



## Assignment of master's thesis

<b>Title:</b>	Interactive Productivity-Enhancing iOS Application
<b>Student:</b>	Bc. Petr Šmejkal
<b>Supervisor:</b>	Ing. Marek Suchánek, Ph.D. et Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Software Engineering
<b>Department:</b>	Department of Software Engineering
<b>Validity:</b>	until the end of summer semester 2024/2025

### Instructions

The goal of this diploma thesis is to create a mobile application for the iOS platform that is intended to increase users' productivity in an interactive way. The resulting application will incorporate a range of productivity-enhancing features, including interval-based studying techniques, long-term planning capabilities, user performance comparisons, and more. It will satisfy both immediate study objectives and long-term educational goals. Furthermore, users will have the flexibility to customize the application's settings to align with their specific study needs.

Follow the principles of software engineering in the following steps:

- Examine the domain of studying, self-improvement, and productivity-enhancing techniques, employ conceptual modeling techniques, and elucidate the essential processes involved.
- Evaluate existing solutions for these issues, highlighting their strengths and weaknesses.
- Define the application requirements and specify use cases for the new application.
- Design the mobile application according to the requirements, using SwiftUI for the frontend technology and Google Firebase for the backend technology.
- Implement, document, and thoroughly test the application prototype.
- Assess the outcomes and propose potential further development.







**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

Master's thesis

## **Interactive Productivity-Enhancing iOS Application**

*Bc. Petr Šmejkal*

Department of Software Engineering  
Supervisor: Ing. Marek Suchánek, Ph.D. et Ph.D.

May 9, 2024



---

## Acknowledgements

I would like to acknowledge and express my deepest gratitude to my supervisor, Ing. Marek Suchánek, Ph.D. et Ph.D., for his invaluable guidance, patience, and expertise throughout the research and writing of my Master's thesis. His insights and suggestions were crucial to its completion. I also extend heartfelt thanks to my family and friends for their unwavering support and guidance throughout my studies.



---

## Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 9, 2024

Czech Technical University in Prague

Faculty of Information Technology

© 2024 Petr Šmejkal. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

### **Citation of this thesis**

Šmejkal, Petr. *Interactive Productivity-Enhancing iOS Application*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

---

## Abstrakt

Tato diplomová práce popisuje celý proces vývoje interaktivní mobilní aplikace pro zlepšování produktivity pro platformu iOS s využitím standardních postupů softwarového inženýrství. Tento proces zahrnuje analýzu domény, analýzu konkurenčních řešení, specifikaci požadavků a případů užití, návrh, implementaci s využitím *SwiftUI* a *Firebase*, testování a vytvoření dokumentace. Práce je zakončena zhodnocením dosažených výsledků a nastíněním možného budoucího rozvoje. Výsledkem je kompletní aplikace připravená pro budoucí rozšíření.

**Klíčová slova** mobilní aplikace, iOS vývoj, Swift, SwiftUI, Firebase, Pomodoro technika, produktivita, metody výuky, softwarové inženýrství, testování software, dokumentace

---

# Abstract

This Master's thesis outlines the comprehensive development process of an interactive productivity-enhancing mobile application for the iOS platform, employing established Software Engineering methodologies. This process includes domain analysis, evaluation of existing solutions, specification of requirements and use cases, design, implementation with *SwiftUI* and *Firebase*, testing, and documentation. The thesis is concluded with the presentation of the results and an outline of possible future development of the application. The outcome is a complete application that is ready for future extensions.

**Keywords** mobile application, iOS development, Swift, SwiftUI, Firebase, Pomodoro technique, productivity, learning methods, software engineering, software testing, documentation



---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Goals of the Thesis</b>	<b>3</b>
<b>2 Learning Approaches and Cognitive Processes</b>	<b>5</b>
2.1 Psychological Foundations of Learning . . . . .	5
2.1.1 Learning Theories . . . . .	5
2.1.1.1 Behaviorism . . . . .	5
2.1.1.2 Cognitivism . . . . .	6
2.1.1.3 Constructivism . . . . .	6
2.1.1.4 Other Learning Theories . . . . .	7
2.1.2 Cognitive Load Theory . . . . .	7
2.1.3 Motivation in Learning . . . . .	7
2.1.3.1 Intrinsic vs. Extrinsic Motivation . . . . .	7
2.1.3.2 Self-Determination Theory . . . . .	8
2.1.4 Memory and Retention . . . . .	8
2.1.5 Colors . . . . .	8
2.1.6 Other Concepts . . . . .	8
2.2 Effective Learning Methods . . . . .	9
2.2.1 Active Learning . . . . .	9
2.2.1.1 Foundations of the Method . . . . .	10
2.2.1.2 Evidence-based Studies . . . . .	11
2.2.1.3 Advantages and Disadvantages . . . . .	11
2.2.1.4 Summary of Active Learning . . . . .	12
2.2.2 Spaced Repetition . . . . .	12
2.2.2.1 Foundations of the Method . . . . .	12
2.2.2.2 Evidence-based Studies . . . . .	13
2.2.2.3 Advantages and Disadvantages . . . . .	14
2.2.2.4 Summary of Spaced Repetition . . . . .	15
2.2.3 Collaborative Learning . . . . .	15
2.2.3.1 Foundations of the Method . . . . .	15
2.2.3.2 Evidence-based Studies . . . . .	16
2.2.3.3 Advantages and Disadvantages . . . . .	17
2.2.3.4 Summary of Collaborative Learning . . . . .	17
2.2.4 Gamification of Learning . . . . .	18

2.2.4.1	Foundations of the Method . . . . .	18
2.2.4.2	Evidence-based Studies . . . . .	20
2.2.4.3	Advantages and Disadvantages . . . . .	20
2.2.4.4	Summary of Gamification in Learning . . . . .	21
2.2.5	Pomodoro Technique . . . . .	22
2.2.5.1	Foundations of the Method . . . . .	22
2.2.5.2	Evidence-based Studies . . . . .	22
2.2.5.3	Advantages and Disadvantages . . . . .	22
2.2.5.4	Summary of Pomodoro Technique . . . . .	23
2.2.6	Other Methods . . . . .	23
2.3	Individual Differences . . . . .	23
2.4	Learning and Technology . . . . .	24
2.5	Pomodoro Technique Research . . . . .	24
2.5.1	Introduction . . . . .	24
2.5.2	Motivation and Research Questions . . . . .	25
2.5.3	Methods . . . . .	25
2.5.3.1	Interview Study . . . . .	26
2.5.3.2	Survey Study . . . . .	26
2.5.4	Results and Discussion . . . . .	27
2.5.4.1	Interview Study Findings . . . . .	27
2.5.4.2	Survey Study Findings . . . . .	28
2.5.4.3	Summary of Results . . . . .	31
2.5.5	Research Conclusion . . . . .	31
2.6	Summary of Learning Approaches and Cognitive Processes . . . . .	31
<b>3</b>	<b>Domain Analysis</b>	<b>33</b>
3.1	Competition Analysis . . . . .	33
3.1.1	Forest: Focus for Productivity . . . . .	33
3.1.1.1	How It Works . . . . .	34
3.1.1.2	Key Features . . . . .	34
3.1.1.3	Advantages and Disadvantages . . . . .	35
3.1.1.4	Summary of Forest Analysis . . . . .	35
3.1.2	Focus To-Do: Focus Timer&Tasks . . . . .	36
3.1.2.1	How It Works . . . . .	36
3.1.2.2	Key Features . . . . .	36
3.1.2.3	Advantages and Disadvantages . . . . .	37
3.1.2.4	Summary of Focus To-Do Analysis . . . . .	37
3.1.3	Study Bunny: Focus Timer . . . . .	38
3.1.3.1	How It Works . . . . .	38
3.1.3.2	Key Features . . . . .	38
3.1.3.3	Advantages and Disadvantages . . . . .	39
3.1.3.4	Summary of Study Bunny Analysis . . . . .	39
3.2	Domain Model . . . . .	40
3.2.1	Domain Model Notations . . . . .	40
3.2.2	Productivity-Enhancing Application Domain . . . . .	42
3.2.2.1	Storylines . . . . .	43
3.2.2.2	Guilds . . . . .	45
3.2.2.3	Board . . . . .	46
3.2.3	Summary of Domain Model . . . . .	46
3.3	Application Requirements . . . . .	47

3.3.1	Frameworks for Requirement Definition . . . . .	47
3.3.1.1	SMART . . . . .	47
3.3.1.2	FURPS . . . . .	47
3.3.1.3	MoSCoW . . . . .	48
3.3.2	Requirements Specification . . . . .	48
3.3.3	Summary of Application Requirements . . . . .	48
3.4	Application Use Cases . . . . .	48
3.4.1	Use Case Diagram Notation . . . . .	49
3.4.1.1	Use Case . . . . .	49
3.4.1.2	Association . . . . .	50
3.4.1.3	Actor . . . . .	50
3.4.1.4	System . . . . .	51
3.4.1.5	Include . . . . .	51
3.4.1.6	Extend . . . . .	51
3.4.1.7	Dependency . . . . .	52
3.4.1.8	Generalization . . . . .	52
3.4.1.9	Realization . . . . .	52
3.4.1.10	Collaboration . . . . .	53
3.4.2	Use Case Specification . . . . .	53
3.4.3	Summary of Application Use Cases . . . . .	53
3.5	Summary of Domain Analysis . . . . .	54
<b>4</b>	<b>Technology Analysis</b>	<b>55</b>
4.1	Firestore Backend . . . . .	55
4.1.1	Firestore Authentication . . . . .	55
4.1.2	Firestore Cloud . . . . .	56
4.1.3	Other Firestore Services . . . . .	56
4.2	Swift Language . . . . .	56
4.2.1	Key Features . . . . .	57
4.2.2	Automatic Reference Counting . . . . .	58
4.2.2.1	Weak References . . . . .	58
4.2.2.2	Unowned References . . . . .	60
4.3	iOS SDK . . . . .	62
4.4	SwiftUI . . . . .	63
4.4.1	How It Works . . . . .	65
4.4.1.1	View Identity . . . . .	65
4.4.1.2	View Lifetime . . . . .	65
4.4.1.3	View Dependencies . . . . .	65
4.5	Xcode . . . . .	66
4.6	Swift Package Manager . . . . .	67
4.7	Testflight and AppStore . . . . .	67
4.7.1	Testflight . . . . .	67
4.7.2	AppStore . . . . .	68
4.8	Summary of Technology Analysis . . . . .	68
<b>5</b>	<b>Design</b>	<b>69</b>
5.1	Choosing Architecture and Design Patters . . . . .	69
5.1.1	Clean Architecture . . . . .	70
5.1.2	MVVM . . . . .	71
5.1.3	Architecture in Practice . . . . .	72

5.1.3.1	Domain Layer . . . . .	72
5.1.3.2	Data Layer . . . . .	72
5.1.3.3	Presentation Layer . . . . .	73
5.2	Database Schema . . . . .	74
5.2.1	ER Diagram Notation . . . . .	74
5.2.1.1	Types, Keys and Fields . . . . .	74
5.2.1.2	Cardinality and Ordinality . . . . .	74
5.2.2	ER Diagram . . . . .	75
5.3	User Interface Design . . . . .	76
5.3.1	Usability . . . . .	76
5.3.2	Colors . . . . .	77
5.3.3	Wireframes . . . . .	78
5.3.3.1	Authentication . . . . .	78
5.3.3.2	Navigation Bar . . . . .	79
5.3.3.3	Storylines . . . . .	79
5.3.3.4	Guilds . . . . .	80
5.3.3.5	Board . . . . .	80
5.3.3.6	Settings . . . . .	81
5.3.4	Miscellaneous . . . . .	82
5.4	Summary of Design . . . . .	82
<b>6</b>	<b>Implementation</b>	<b>83</b>
6.1	Initial Setup . . . . .	83
6.1.1	Jira and BitBucket Setup . . . . .	83
6.1.1.1	Jira Setup . . . . .	84
6.1.1.2	BitBucket Setup . . . . .	87
6.1.1.3	Jira and BitBucket Integration . . . . .	90
6.1.2	Xcode Project Setup . . . . .	91
6.1.3	Firebase Setup . . . . .	93
6.1.3.1	Create New Firebase Project . . . . .	93
6.1.3.2	Register iOS App . . . . .	93
6.1.4	AppStore Connect Setup . . . . .	95
6.2	Core Components . . . . .	97
6.2.1	NSLogging . . . . .	97
6.2.2	Dependency Injection . . . . .	98
6.2.3	String Catalog . . . . .	98
6.2.4	Navigation Routing . . . . .	99
6.3	Typical Workflow . . . . .	102
6.3.1	Issue and Branch Creation . . . . .	102
6.3.2	Module Implementation . . . . .	102
6.3.3	Committing, Reviewing, Merging and Deployment . . . . .	107
6.4	Storylines Module . . . . .	109
6.5	Other Modules . . . . .	110
6.6	Summary of Implementation . . . . .	111
<b>7</b>	<b>Testing and Documentation</b>	<b>113</b>
7.1	Testing . . . . .	113
7.1.1	Unit Testing . . . . .	113
7.1.1.1	Characteristics of Unit Testing . . . . .	113
7.1.1.2	Mocking . . . . .	114

7.1.1.3	Unit Testing Example . . . . .	115
7.1.2	User Interface Testing . . . . .	118
7.1.3	Usability Testing . . . . .	118
7.1.3.1	Usability Testing Scenarios . . . . .	120
7.1.3.2	Changes Made Based On Feedback . . . . .	120
7.1.4	Performance Testing . . . . .	121
7.1.4.1	Launch Performance Testing . . . . .	121
7.1.5	Network Performance Testing . . . . .	122
7.1.6	Summary of Testing . . . . .	123
7.2	Documentation . . . . .	124
7.2.1	Code Commentary Format . . . . .	124
7.2.2	DocC . . . . .	124
7.2.3	Summary of Documentation . . . . .	125
<b>8</b>	<b>Results and Future Development</b>	<b>127</b>
8.1	Results . . . . .	127
8.1.1	Analysis, Design, Implementation and Testing . . . . .	127
8.1.2	Requirements Fulfillment . . . . .	128
8.1.3	Summary of the Results . . . . .	128
8.2	Future Development . . . . .	128
8.2.1	Usability Testing Feedback . . . . .	128
8.2.2	Remaining Requirements . . . . .	128
8.2.3	Other Possible Enhancements . . . . .	129
8.2.4	Summary of Future Development . . . . .	129
	<b>Conclusion</b>	<b>131</b>
	<b>Bibliography</b>	<b>133</b>
<b>A</b>	<b>Requirement Specification</b>	<b>145</b>
A.1	Non-Functional Requirements . . . . .	145
A.2	Functional Requirements . . . . .	146
<b>B</b>	<b>Use Case Specification</b>	<b>151</b>
<b>C</b>	<b>Wireframes</b>	<b>159</b>
<b>D</b>	<b>Final Application Interface Screenshots</b>	<b>169</b>
<b>E</b>	<b>Usability Testing Scenarios</b>	<b>191</b>
<b>F</b>	<b>List of Abbreviations</b>	<b>199</b>
<b>G</b>	<b>Contents of Attachments</b>	<b>201</b>



---

## List of Figures

2.1	Complexity of Tasks Performed During a Lecture [1] . . . . .	9
2.2	Possible Lecture Organization According to Active Learning [1] . .	10
2.3	The Forgetting Curve [2] . . . . .	13
2.4	The Forgetting Curve with Repeated Review of Material [3] . . . .	14
2.5	Basic Steps in Collaborative Learning [4] . . . . .	16
2.6	Gamification Taxonomy [5] . . . . .	19
2.7	Work approaches of the Survey Participants . . . . .	28
2.8	Survey Participant's Satisfaction With Their Study Approaches . .	29
2.9	Pomodoro Timer Usefulness For Different Tasks . . . . .	29
2.10	Influence of the Pomodoro Technique . . . . .	30
2.11	Comparison of Gaminified and Non-Gamified Pomodoro Technique	30
3.1	Screenshots of Forest: Focus for Productivity [6] . . . . .	34
3.2	Screenshots of Focus To-Do: Focus Timer&Tasks [7] . . . . .	36
3.3	Screenshots of Study Bunny: Focus Timer [8] . . . . .	38
3.4	Class Example; based on [9] . . . . .	40
3.5	Association Example; based on [9] . . . . .	40
3.6	Aggregation Example; based on [9] . . . . .	41
3.7	Composition Example; based on [9] . . . . .	41
3.8	Generalization Example; based on [9] . . . . .	42
3.9	Productivity-Enhancing Application Domain . . . . .	43
3.10	Storylines Subdomain . . . . .	44
3.11	Guilds Subdomain . . . . .	45
3.12	Board Subdomain . . . . .	46
3.13	Requirement Specification Format . . . . .	48
3.14	Requirements Specification Overview . . . . .	49
3.15	Use Case Notation; based on [10] . . . . .	50
3.16	Association Notation; based on [10] . . . . .	50
3.17	Actor Notation; based on [10] . . . . .	50
3.18	System Notation; based on [10] . . . . .	51
3.19	Include Notation; based on [10] . . . . .	51
3.20	Extend Notation; based on [10] . . . . .	52
3.21	Dependency Notation; based on [10] . . . . .	52
3.22	Generalization Notation; based on [10] . . . . .	52
3.23	Realization Notation; based on [10] . . . . .	53

3.24	Collaboration Notation; based on [10]	53
3.25	Use Case Specification Format	53
3.26	Use Case Diagram	54
4.1	Cloud Firestore Collections and Documents Example	56
4.2	Swift Weak Referencing Example [11]	59
4.3	Relation Between <code>Person</code> and <code>Apartment</code> Objects [11]	59
4.4	Discarding Reference to <code>Person</code> Object [11]	60
4.5	Discarding Reference to <code>Apartment</code> Object [11]	60
4.6	Swift Unowned Referencing Example [12]	61
4.7	Relation Between <code>Customer</code> and <code>CreditCard</code> Objects [12]	61
4.8	Relation Between <code>Customer</code> and <code>CreditCard</code> Objects [12]	62
4.9	iOS SDK Architecture [13]	63
4.10	SwiftUI Declarative Code Example	64
4.11	Xcode 15.3 IDE	66
5.1	Clean Architecture as Described by Robert C. Martin [14]	71
5.2	MVVM Pattern; based on [15]	72
5.3	Architecture of the Feature Module	73
5.4	Fields, Private and Foreign Keys and Types Notation [16]	74
5.5	Cardinality and Ordinality Notation [16]	75
5.6	ER Diagram for Productivity-Enhancing Application	76
5.7	Primary and Secondary Colors for the UI Design	77
5.8	Wireframes of the Authentication Screens	78
5.9	Wireframe of the Navigation Bar	79
5.10	Wireframes of the Home and Storylines Tabs	79
5.11	Wireframes of the Guilds Tab	80
5.12	Wireframes of the Board Tab	81
5.13	Wireframe of the Settings Tab	81
6.1	Jira Team-Managed vs. Company-Managed Projects [17]	85
6.2	Jira Setup Overview	85
6.3	Jira Kanban Board After Project Creation	86
6.4	Jira Issue Creation	87
6.5	BitBucket Project Creation	88
6.6	BitBucket Repository Creation	89
6.7	BitBucket Initial Page	89
6.8	Jira-BitBucket Integration	90
6.9	Xcode Project Creation	91
6.10	Xcode Project Git Setup	92
6.11	Adding Firebase SDK Dependency to Xcode	94
6.12	<code>AppDelegate</code> Class	94
6.13	<code>AppDelegate</code> Connection to <code>App Struct</code>	95
6.14	Bundle Identifier Registration	95
6.15	AppStore Connect App Registration	96
6.16	Skeleton of <code>MemorizifyLogger</code>	97
6.17	Xcode String Catalog	99
6.18	<code>Router</code> Class Skeleton	99
6.19	<code>ComponentRoute</code> Enumeration	100
6.20	<code>Router</code> Class Initialization and Passing	100



6.21	NavigationStack Usage Example	101
6.22	User Model	102
6.23	IncreaseScoreUseCase Protocol	103
6.24	Update of UserRepository Protocol	103
6.25	Update of UserRepository Implementation	104
6.26	Update of UserRepository Implementation	105
6.27	UserRepository and IncreaseScoreUseCase Registration	105
6.28	UserScoreViewModel Implementation	106
6.29	UserScoreView Implementation	107
6.30	Merged BitBucket Pull Request	108
6.31	Testflight Distribution	109
6.32	Storyline Custom Encoding and Decoding	110
6.33	Memorizify Jira Board Overview	111
6.34	Memorizify BitBucket PRs Overview	111
7.1	Implementation of OnboardingRepository	115
7.2	Mock of KeychainProvider	116
7.3	Unit Testing of OnboardingRepository	117
7.4	UI Testing of Reset Password Feature	119
7.5	Usability Test Scenario Format	120
7.6	Implemented Usability Testing Changes	121
7.7	Xcode Network Performance Presets	122
7.8	Memorizify Network Conditions Handling	123
7.9	Xcode Code Comment Format	124
7.10	Xcode DocC Referencing	125
C.1	Initial Home	159
C.2	Sign Up	159
C.3	Log In	160
C.4	Reset Password	160
C.5	Home Tab	161
C.6	Plain Timer	161
C.7	Storylines Tab	162
C.8	Storyline Detail	162
C.9	Storyline Setup	163
C.10	Storyline Timer	163
C.11	Guilds Tab	164
C.12	Guild Detail	164
C.13	Create Guild	165
C.14	Invite Friend	165
C.15	Board Tab	166
C.16	Board Detail	166
C.17	Settings Tab	167
C.18	Navigation Bar	167
D.1	Launch Screen	169
D.2	Onboarding 1st Page	169
D.3	Onboarding 2nd Page	170
D.4	Onboarding 3rd Page	170
D.5	Initial Home	171

D.6 Sign Up . . . . .	171
D.7 Sign Up Done . . . . .	172
D.8 Log In . . . . .	172
D.9 Reset Password . . . . .	173
D.10 Password Reset Done . . . . .	173
D.11 Home Tab . . . . .	174
D.12 Storyline Menu . . . . .	174
D.13 Plain Timer Setup . . . . .	175
D.14 Plain Timer . . . . .	175
D.15 Storylines Tab . . . . .	176
D.16 Storyline Detail . . . . .	176
D.17 Storyline Setup . . . . .	177
D.18 Storyline Timer . . . . .	177
D.19 Guilds Tab . . . . .	178
D.20 Guild Invitation . . . . .	178
D.21 Create Guild . . . . .	179
D.22 Update Guild . . . . .	179
D.23 Guild Detail . . . . .	180
D.24 Guild Detail (Dark) . . . . .	180
D.25 Leader Actions . . . . .	181
D.26 Invite Friend . . . . .	181
D.27 Member Context Menu (Leader's Point of View) . . . . .	182
D.28 Member Context Menu (Common Member's Point of View) . . . . .	182
D.29 Board Tab . . . . .	183
D.30 Board Detail . . . . .	183
D.31 Board Tab (iPad OS) . . . . .	184
D.32 Board Detail (iPad OS) . . . . .	185
D.33 Settings Tab . . . . .	186
D.34 Change Password . . . . .	186
D.35 Change Language . . . . .	187
D.36 Change Notifications . . . . .	187
D.37 Delete Account 1st . . . . .	188
D.38 Delete Account 2nd . . . . .	188
D.39 Delete Account 3rd . . . . .	189
D.40 Push Notification . . . . .	189
D.41 Loading Placeholders . . . . .	190
D.42 No Connection . . . . .	190

---

# Introduction

In our contemporary world full of distractions, from video games to social media, staying focused and productive can be a difficult challenge. These distractions not only hinder progress but can also lead individuals to abandon their goals entirely. In such an environment, maintaining concentration and productivity becomes increasingly difficult for many.

However, there are evidence-based solutions designed to combat these issues. Research in various scientific disciplines has led to the development of strategies that can help individuals reclaim their focus and enhance productivity. Leveraging these insights, technology offers powerful tools to support these strategies, transforming theoretical solutions into practical applications.

This leads to the introduction of a specialized mobile application developed for the iOS platform. The app utilizes scientifically-backed methods to enhance focus and productivity, integrating gamification elements through engaging storylines that users can advance through using a Pomodoro timer. Additionally, it facilitates the formation of guilds, allowing users to either compete in completing storyline goals or simply study together, while monitoring each other's progress.

The thesis will detail the journey from analysis and design to implementation and testing of this application, conducted according to standard processes of Software Engineering. It showcases the functionality and development of the app as a strategic tool for improving productivity in a distraction-filled environment.



---

## Goals of the Thesis

The main goal of this thesis is to develop an interactive productivity-enhancing mobile application for the iOS platform, following the standard processes of Software Engineering. This process involves domain analysis, evaluation of existing solutions, specification of requirements and use cases, design, implementation using *SwiftUI* and *Firebase*, testing, and documentation. These partial goals contribute to the fulfillment of the main goal.

Initially, a comprehensive analysis of the psychological principles underlying learning processes is required, along with an exploration of effective learning methods. Additionally, a detailed examination of the domain of interactive productivity-enhancing applications is essential. These analyses will guide the specification of requirements and use cases. Subsequently, a technology assessment is crucial to fully understand the capabilities of the tools employed. The design phase will then involve selecting appropriate architectures and design patterns, as well as the development of the user interface. Implementation will follow, supported by thorough testing and comprehensive documentation. The thesis will conclude with an evaluation of the results and possibilities for future development.



---

# Learning Approaches and Cognitive Processes

This chapter explores foundational learning approaches and cognitive processes that shape educational practices. It examines key theories and reveals how they influence the acquisition, processing, and retention of knowledge. By understanding these theories, it is possible to enhance educational strategies and teaching methods, ultimately improving learning outcomes. This discussion aims to provide insights into effectively integrating these theories into practical educational settings, offering a deeper understanding of their impact on both teaching and learning.

## 2.1 Psychological Foundations of Learning

The section explores the psychological foundations that shape how people learn, focusing on key theories and methods. These perspectives provide valuable insights into the processes of acquiring, processing, and retaining knowledge, influencing modern educational practices. Understanding these theories helps educators and learners optimize learning strategies and improve educational outcomes.

### 2.1.1 Learning Theories

This section delves into the fundamental psychological theories that shape modern educational practices, exploring how different schools of thought — *behaviorism*, *cognitivism*, and *constructivism* — contribute to our understanding of learning processes. Each theory offers a distinct perspective on how individuals acquire, process, and retain knowledge, influencing various educational strategies and teaching methods. By examining these theories, it is possible to appreciate the diverse approaches to education and their implications for both teaching and learning in various contexts.

#### 2.1.1.1 Behaviorism

*Behaviorism* is a psychological approach that emphasizes the study of observable behaviors over internal mental states, seeking a level of scientific rigor

comparable to that in the physical sciences. Initiated by John B. Watson, behaviorism was a response to the introspective methods prevalent at the time, which Watson criticized as unscientific. He insisted that psychology should only deal with behaviors that are observable and measurable, dismissing the use of introspection and subjective accounts of consciousness as unreliable. Watson's stance was significantly influenced by Ivan Pavlov's work on conditioning, which he saw as essential for a scientific approach to studying human behavior. He advocated that through conditioning, simple responses could be developed into complex behaviors, emphasizing the powerful role of environmental influence in human development. This idea was highlighted by Watson's assertion that he could train any child to become any specialist, highlighting his belief in the power of environmental conditioning. [18, 19]

As the dominant force in psychology from the 1920s through the early 1960s, behaviorism shaped various educational theories and practices. The framework of behaviorism includes key theories like *Ivan Pavlov's Classical Conditioning* and *B.F. Skinner's Operant Conditioning*. These theories explain learning through environmental interactions that modify behavior via consequences, thus shaping and maintaining behaviors. Despite challenges from the rise of cognitive psychology, the principles of behaviorism remain integral to many educational methods today, emphasizing the enduring influence of environmental factors on the learning process. [18, 19]

### 2.1.1.2 Cognitivism

*Cognitivism* emerged as a significant learning theory, countering behaviorism's focus on observable behaviors by emphasizing the importance of internal mental processes. This approach views learners as active participants who process information, rather than passive recipients of stimuli. Cognitivism posits that the mind functions like a complex machine, where information is received, processed, stored, and retrieved. Key elements of this theory include the understanding of how attention and memory work together to influence learning. Cognitivists study how learners perceive, think, understand, and remember information, emphasizing strategies like organization, metaphorical thinking, and problem-solving to facilitate deeper learning. Through understanding these cognitive processes, educators can design more effective teaching methods that enhance comprehension and retention, ultimately leading to more meaningful educational experiences. [18, 20]

### 2.1.1.3 Constructivism

*Constructivism* in learning theories posits that learners actively construct their own understanding and knowledge of the world, through experiencing things and reflecting on those experiences. This approach emphasizes that learning is inherently personal, influenced by the learner's own perspectives and interpretations. Constructivists argue that people learn by integrating new information with our existing cognitive frameworks and knowledge bases, thus making learning a highly individualized process. [21]

Educational strategies based on constructivism focus on hands-on, activity-based teaching and learning that encourage students to discover principles for themselves and to construct knowledge by working to solve realistic prob-



lems. The role of the teacher shifts from the dispenser of information to a guide or facilitator, providing support and scaffolding to students as they build their own understanding. This method encourages collaboration and the sharing of perspectives among students, enhancing the depth and breadth of learning through social interaction. [21]

### 2.1.1.4 Other Learning Theories

Several learning theories complement the main educational frameworks. *Connectivism* deals with learning through digital and social networks, reflecting modern technological impacts. *Social Learning Theory*, by Albert Bandura, emphasizes learning through observation and modeling others. *Experiential Learning*, proposed by David Kolb, involves a cyclic process of gaining knowledge through direct experience, reflection, conceptualization, and active experimentation. These theories offer diverse perspectives on how people absorb, process, and retain information, shaping contemporary educational practices. [18]

### 2.1.2 Cognitive Load Theory

*Cognitive Load Theory (CLT)*, developed by John Sweller, focuses on how cognitive resources are utilized during learning and the impact on the ability to process new information efficiently. CLT distinguishes three types of cognitive load: intrinsic (related to the complexity of the material itself), extraneous (how information is presented), and germane (the construction and automation of schemas). [22, 23]

In the realm of mobile apps, CLT has profound implications for design and usability. Effective mobile app design should minimize extraneous load by reducing clutter and simplifying navigation, and maximize germane load by encouraging deep cognitive processing. By carefully managing these elements, app designers can enhance usability and facilitate smoother learning experiences. Apps that effectively balance cognitive loads help users learn and retain information more efficiently, thus enriching the learning experience. [24]

### 2.1.3 Motivation in Learning

This section explores the psychological underpinnings of learning in the context of mobile applications, focusing on intrinsic and extrinsic motivation and *Self-Determination Theory (SDT)*. The section delve into how mobile apps can effectively harness these motivational strategies to enhance learning experiences. Understanding these foundational theories helps in designing mobile apps that not only engage users but also sustain their interest and facilitate deeper learning through well-structured interactive elements. [18]

#### 2.1.3.1 Intrinsic vs. Extrinsic Motivation

Recent research highlights that mobile apps can enhance learning by balancing intrinsic and extrinsic motivational elements. *Intrinsic Motivation* in mobile learning is driven by engaging content that resonates with users' interests or challenges them at a cognitive level, promoting deeper engagement and satisfaction. Conversely, *Extrinsic Motivation* can be fostered through the use

of gamified elements such as digital badges or progress trackers, which provide external rewards and serve as performance feedback to encourage continuous engagement and achievement. [25]

### 2.1.3.2 Self-Determination Theory

*Self-Determination Theory (SDT)* suggests that for mobile apps to be effective in learning, they need to support autonomy, competence, and relatedness. Autonomy can be promoted by offering customizable learning paths that allow users to control their learning experience. Competence is encouraged through adaptive challenges and real-time feedback that help users gauge their progress. Lastly, relatedness can be enhanced through social features like community forums or group challenges, which connect learners with peers and mentors, thereby increasing their engagement and motivation to learn. [25]

### 2.1.4 Memory and Retention

In mobile apps, enhancing memory retention can be effectively achieved through strategies like *Spaced Repetition* and *Retrieval Practice*. For detailed overview of spaced repetition method refer to Section 2.2.2.

Spaced repetition involves presenting information at gradually increasing intervals to reinforce learning over time, while retrieval practice focuses on recalling information to strengthen memory retention. These methods are supported by research indicating that engaging with material repeatedly over spaced intervals and actively retrieving information helps in solidifying memories, making them more resistant to forgetting. [26, 27]

### 2.1.5 Colors

Colors play a significant role in psychology as they can influence mood, behavior, and perceptions. In the field of environmental psychology, colors are known to have specific psychological effects—for example, blue often induces feelings of calmness and serenity, while red can evoke emotions of passion and urgency. These effects are not just limited to static environments but are also impactful in dynamic digital interfaces. [28]

In the context of mobile applications, the psychology of colors is critical in user interface design. Choosing the right color scheme can enhance user experience, influence user behavior, and increase engagement. For instance, using blue might promote trust and reliability, which is ideal for banking or healthcare apps, whereas using bright and energetic colors like orange or red might be more effective in fitness apps to motivate users. By understanding the psychological impact of different colors, developers can create more effective and appealing mobile applications. [29]

### 2.1.6 Other Concepts

Psychology in learning integrates various other concepts. *Attention and Information Processing* examines how UI/UX design influences user attention and information processing. *Neuroscience of Learning* applies brain-based insights to enhance app functionality. *Social Learning Theory* leverages app-

based social features to promote interactive learning. *Metacognition and Self-regulated Learning* foster reflective learning practices and goal setting. Lastly, *Psychological Impact of Technology* explores both the benefits and potential issues of mobile learning, such as the digital divide. [30]

## 2.2 Effective Learning Methods

Effective learning methods are crucial for enhancing educational outcomes and personal growth. This section distinguishes between methods, techniques, and strategies within education, clarifying their roles: methods encompass overarching theories like active learning or spaced repetition, techniques are specific practices within these methods, and strategies are personal adaptations by learners [31]. By exploring these various approaches, backed by historical evidence and contemporary research, educators and learners can significantly improve engagement, retention, and comprehension. The discussions that follow will detail *Active Learning*, *Spaced Repetition*, *Collaborative Learning*, *Gamification of Learning*, and other effective strategies to enrich the educational experience.

### 2.2.1 Active Learning

As many people have experienced, traditional lectures typically involve a professor standing in front of a blackboard, explaining a topic while students sit at their desks and listen. This educational method is considered outdated by many [32]. The issue is that the length of such classes is often too long for students to remain concentrated and motivated throughout the entire lecture. The following Figure 2.1 illustrates the difficulty of typical tasks performed during a lecture.



Figure 2.1: Complexity of Tasks Performed During a Lecture [1]

However, there are newer approaches to education, one of which is *Active Learning*. As the name suggests, active learning (sometimes also referred as student-centered learning [33]) involves students actively participating in the learning process, rather than passively attending lectures. This approach is commonly used when a group of students is assigned the same task to complete. Typically, students are encouraged to collaborate and work together to accomplish the task. The term active learning does not have a single, precise definition, as it encompasses various learning strategies including group work, project-based methods, learning through play, and technology-based learning. Despite the diversity of these approaches, they share a common element: active participation. This means that students take an active role in their own learning process. [1] Time in a lecture can be organized for example as shown in Figure 2.2 below.

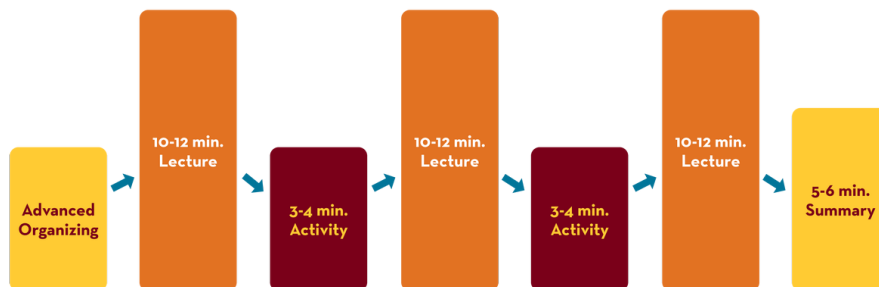


Figure 2.2: Possible Lecture Organization According to Active Learning [1]

### 2.2.1.1 Foundations of the Method

The concept of active learning has existed for centuries, but the specific term active learning as it is understood today has been in use since the late 1970s. During this period, the method gained significant popularity, leading to its endorsement by educators, researchers, cognitive psychologists, and instructional designers. These professionals advocated for educational models that emphasized a greater level of student participation in the learning process [33]. Prominent educators who contributed significantly to this shift include Paulo Freire [33, 34], who profoundly redefined student-teacher interactions. Others who have influenced this method include John Dewey and Maria Montessori, each bringing unique perspectives to active and experiential learning approaches [33, 35, 36].

To more clearly delineate the essence of active learning, the following principles are commonly suggested to guide the active learning methodology:

1. **Purposive:** The significance of the task is aligned with the students' interests and concerns.
2. **Reflective:** Encouragement for students to contemplate the implications and significance of their learning.

3. **Negotiated:** The process of learning involves students and teachers collaboratively determining the goals and methods.
4. **Critical:** Emphasis on students recognizing diverse approaches and methods to absorb the content.
5. **Complex:** Tasks are compared to real-world complexities while fostering deeper analytical thinking among students.
6. **Situation-driven:** Learning tasks are shaped by situational needs, ensuring relevance and applicability.
7. **Engaged:** Activities mirror real-life tasks, enhancing the practical application of learning. [37]

### 2.2.1.2 Evidence-based Studies

Numerous studies support the effectiveness of active learning, demonstrating robust and comprehensive results across various disciplines and contexts over an extended period. Generally, these studies indicate that student groups employing active learning methods outperform those using traditional, passive learning approaches [38]. Furthermore, research shows that active learning not only enhances basic study effectiveness but also fosters critical thinking [38], improves learning attitudes [39], boosts collaboration skills [38], increases the ability to solve real-world problems [40], increases knowledge retention [41] and more. Overall, the research strongly suggests that active learning is an effective educational method.

### 2.2.1.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of active learning.

#### Active Learning Advantages:

- Providing context that helps students recognize the relevance of their studies.
- Enhancing the retention of knowledge.
- Deepening understanding and improving the ability to apply knowledge in real-life situations.
- Increasing student engagement, making the learning process more enjoyable.
- Accommodating a wider variety of learning styles. [42]

#### Active Learning Disadvantages:

- Often requiring more time for instructors to prepare effectively.
- Being less efficient than traditional didactic methods for conveying foundational knowledge.
- Potentially causing frustration among students who are unprepared to participate actively. [42]

### 2.2.1.4 Summary of Active Learning

Active learning and technology can seamlessly integrate to create a fun and interactive environment that enhances learning effectiveness. There are numerous ways to utilize technology during active learning. Technology serves as a crucial element for connecting students, allowing them to interact with each other without needing to be physically present. It also enables methods that would not be possible otherwise, such as advanced visualization techniques. Another significant aspect that technology introduces is the element of fun—gamification plays a major role in learning by making it enjoyable, which in turn boosts both attitude and effectiveness. [43, 44]

In conclusion, this section explains what active learning entails, how it is possible to implement its principles, its benefits in learning, and its advantages and disadvantages. It also discusses how active learning can be integrated with technology. This section provides a solid foundation for later designing a productivity-enhancing mobile app.

### 2.2.2 Spaced Repetition

This section explores another learning method known as *Spaced Repetition*. Imagine there is a substantial amount of information that a student needs to learn. To manage this effectively, the information can be broken down into smaller, more manageable chunks. Each chunk can then be learned individually, allowing the student to focus on specific segments of the overall material.

#### 2.2.2.1 Foundations of the Method

Spaced repetition is particularly effective because it prioritizes the mastery of all information based on the learner's familiarity and recall difficulty. The method involves reviewing the chunks that the student knows the least more frequently, ensuring these more challenging areas are reinforced. Conversely, the chunks that are easiest for the student are repeated less often. This strategic approach to repetition helps optimize the learning process, making it both efficient and adaptive to the student's individual needs. [45]

This method has been recognized since the 1880s when it was first studied by Hermann Ebbinghaus, a German psychologist. Ebbinghaus described how people tend to forget information over time in a phenomenon he called the *Forgetting Curve*. He conducted experiments to investigate memory retention and discovered that memory loss occurs rapidly after initial learning. Specifically, he found that up to 75% of learned material can be forgotten within the first 48 hours, with the most significant memory loss occurring within the first hour after learning [46] as shown in the following Figure 2.3. Also note that the student will not remember everything that was presented in the material.

The process of forgetting is a natural aspect of human memory, but it can be disrupted. One initial strategy might be to review the material immediately after learning, such as right after a class. This immediate review can indeed extend the duration for which the student retains the information, but the benefit is typically short-lived. Studies have shown that within about 24 hours, the level of knowledge retention between a student who reviewed the material immediately after class and one who did not is nearly identical [47]. To combat

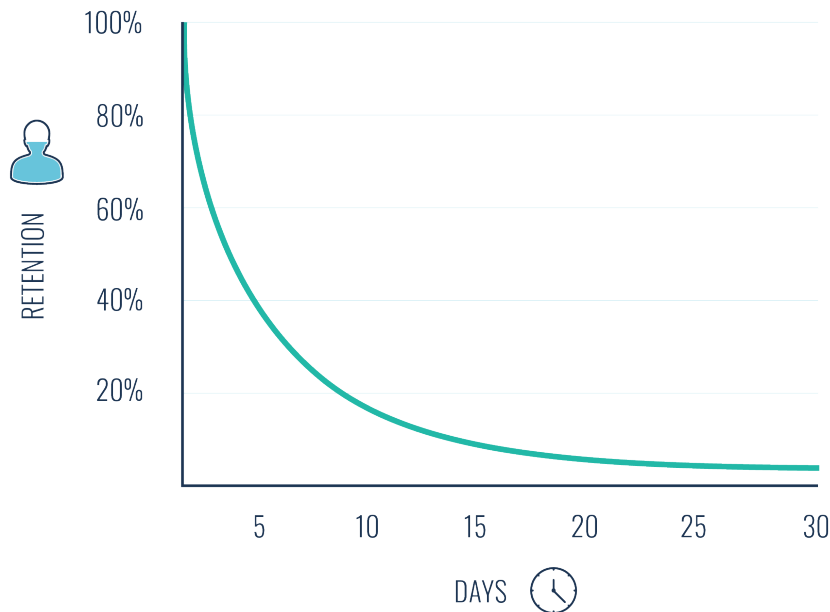


Figure 2.3: The Forgetting Curve [2]

this rapid decline in memory, more effective methods such as spaced repetition can be applied, which strategically spaces out review sessions to enhance long-term retention. [48]

As previously mentioned, this method is based on repetitive learning, with a focus on prioritizing more difficult material. If a repetitive method is applied, it will alter the forgetting curve as depicted in Figure 2.4. It is important to note that, as mentioned earlier, retention does not significantly increase after just one immediate repetition. This method becomes truly effective only when it is performed multiple times, extending the intervals between reviews to better reinforce memory and comprehension.

### 2.2.2.2 Evidence-based Studies

Numerous evidence-based studies highlight the benefits of spaced repetition in learning. The technique is particularly effective in improving long-term memory retention and enhancing recall efficiency [49], which are vital for academic success. By spacing out learning over time, it also helps prevent mental fatigue [50], maintaining cognitive performance throughout extended study periods [51]. Additionally, spaced repetition makes learners to think about their learning process more reflectively, improving their ability to regulate their own learning and making judgements about what, how, and when to study [50]. These benefits collectively contribute to a more efficient and enriching learning experience.

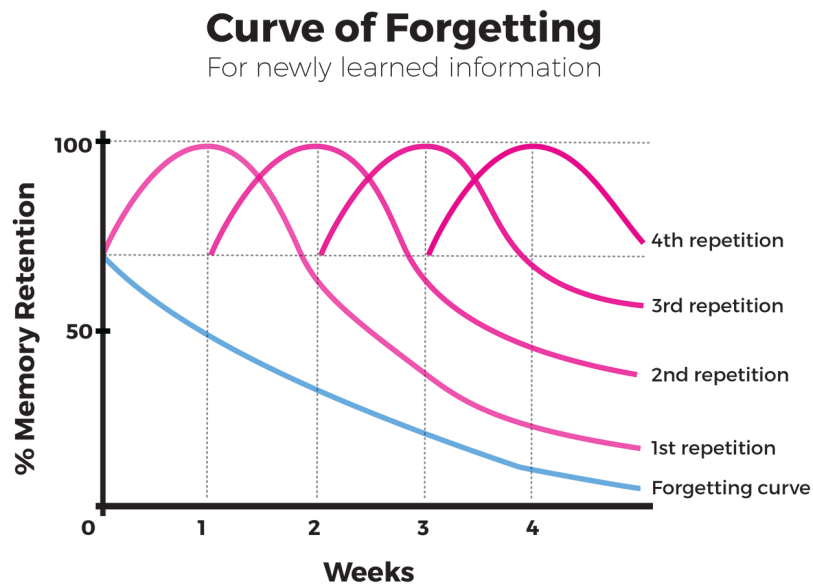


Figure 2.4: The Forgetting Curve with Repeated Review of Material [3]

#### 2.2.2.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of spaced repetition.

##### Spaced Repetition Advantages:

- Spaced repetition enhances flexibility and allows for just a short time of daily practice.
- It is beneficial for visual learners by linking imagery to information.
- Effective for memorizing isolated facts, vocabulary, or rules.
- Numerous implementations and tools are available to assist in creating and using flashcards. [52]

##### Spaced Repetition Disadvantages:

- Spaced repetition requires advance planning and sufficient time allocation.
- It demands long-term commitment to observe noticeable results.
- It is less effective for memorizing larger theories or extensive information.
- Preparations can be time-consuming and necessitates significant creativity. [52]



### 2.2.2.4 Summary of Spaced Repetition

Spaced repetition is a powerful learning strategy that can be effectively integrated with technology, making it highly accessible and customizable. A number of mobile apps and web applications have already successfully adopted this approach, leveraging the benefits of spaced repetition to enhance educational outcomes. [53]

Among the well-known implementations of spaced repetition is the *Leitner System*, a learning method that utilizes flashcards (cards with questions) categorized into different boxes, typically three. All cards start in the first box, which is considered the easiest. During a session, a learner reviews a card from the first box; if they answer correctly, the card moves to the next box, which is considered more difficult. If the answer is incorrect, the card remains in the current box or moves back to an easier one, if applicable. The system is designed so that cards in more difficult boxes are reviewed more frequently, ensuring that the learner spends more time on challenging material, thus reinforcing their understanding and retention.

In conclusion, spaced repetition is a proven and effective learning method that strategically utilizes the human memory process to enhance information retention and recall. By scheduling reviews to counteract the natural tendency to forget, it allows learners to concentrate more frequently on challenging material, thus enhancing overall learning efficiency. Additionally, this method integrates well with technology, as evidenced by the wide array of tools available for its implementation.

### 2.2.3 Collaborative Learning

One of the popular methods of learning is *Collaborative Learning*, which can occur either peer-to-peer or in larger groups. Peer learning involves students working in pairs or small groups to achieve a specified goal. This approach allows students to teach each other as they work toward solving the given problem. [54]

Collaborative learning originated in the 1960s in the UK, when researchers such as Basil Bernstein and Michael Young began to create environments to study how language and social structure influenced learning, with a particular focus on interaction and group work. In the 1980s, this concept was further formalized, providing a robust foundation for other researchers to build upon. This groundwork allowed for the expansion and diversification of collaborative learning concepts during the 1990s, including the integration of modern technologies. Recently, this approach has continued to evolve with contemporary advancements. [55]

#### 2.2.3.1 Foundations of the Method

Students work together to solve problems and share knowledge, benefiting from diverse perspectives and enhancing their understanding through mutual interaction. This form of learning integrates various theoretical and methodological insights from multiple disciplines, emphasizing the importance of group dynamics in education. Students are encouraged to engage actively with the content, leading to deeper understanding, retention of the material, and the development of critical thinking and communication skills. Research supports the effec-

tiveness of collaborative learning in fostering higher cognitive processes and social skills, crucial for academic and professional success. With the rise of digital platforms, collaborative learning has expanded to include computer-supported environments, making it accessible to a wider audience and providing new opportunities for interaction and learning across geographical boundaries. [56] Five basic steps to achieve a collaborative learning environment are depicted in the following Figure 2.5.

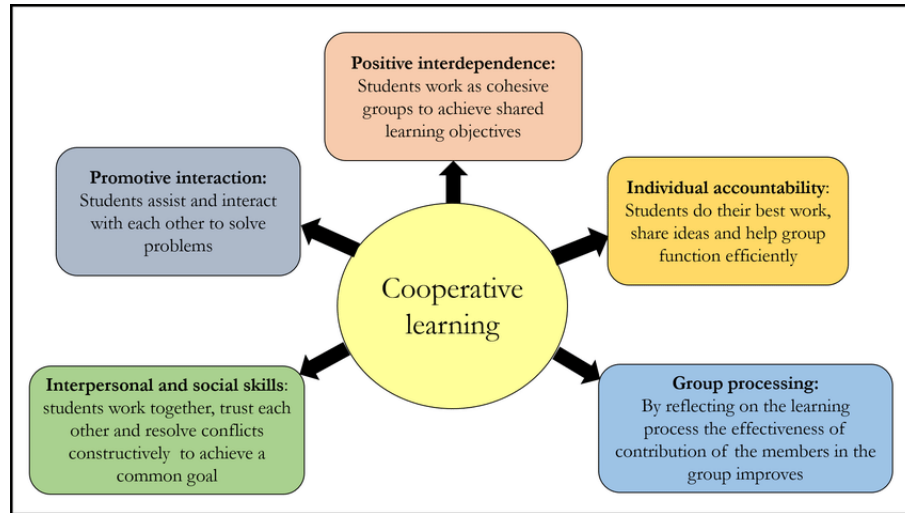


Figure 2.5: Basic Steps in Collaborative Learning [4]

### 2.2.3.2 Evidence-based Studies

Numerous studies support the effectiveness of collaborative learning, demonstrating its superiority over traditional learning methods in various aspects. Research consistently shows that collaborative learning enhances communication skills, exposes students to diverse perspectives, and significantly improves retention rates. [57]

Another study employs evolutionary game theory to analyze collaborative learning dynamics, revealing that perceived academic value and social benefits motivate students, while time and effort costs deter them. This theoretical evidence provides insights into the factors that influence collaborative learning, aiding educational theorists and policymakers in enhancing its effectiveness. [58]

Furthermore, this approach fosters deeper understanding and engagement with the material, as students actively participate and learn from each other's insights and experiences. This collective learning environment not only boosts academic performance but also cultivates essential social skills and critical thinking abilities, making it a valuable educational strategy. [59]

### 2.2.3.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of collaborative learning.

#### Collaborative Learning Advantages:

- Collaborative learning requires students to communicate, debate, and negotiate fostering essential skills for workplace readiness as they reach consensus on various topics.
- Through collaborative efforts, students interact with peers from different cultures and backgrounds, gaining valuable insights and observing diverse learning strategies.
- Collaboration allows students to learn from each other, broadening their perspectives and deepening their understanding. This process includes providing peer feedback, which serves as an effective check on each other's work. [60]

#### Collaborative Learning Disadvantages:

- Introverted students often prefer time to think and process information internally. They may find it challenging to participate in social settings where immediate verbal interaction and openness are required.
- Effective group work doesn't happen automatically; it requires structured guidance. Educators should teach students about positive interdependence, managing diverse learning styles, and ensuring inclusivity.
- Evaluating group work can lead to perceived inequities. Some students might feel that their peers do not deserve a share of the group's high marks due to minimal contribution, while others may worry about underperformers affecting their own grades. [60]

### 2.2.3.4 Summary of Collaborative Learning

Technology in collaborative learning has two sides. On one hand, it can be extremely beneficial, opening up new possibilities and experiences such as communication tools, team management tools, and collaboration platforms like 360Learning<sup>1</sup>, EdApp<sup>2</sup>, or Slack<sup>3</sup>. These technologies can enhance efficiency, attitude, and retention, and can help bridge the gap between students, among other benefits. On the other hand, technology can sometimes obstruct the learning process, which some studies have indeed shown. Additionally, setting up a collaborative learning environment may be challenging due to limited resources, such as when every participant requires a tablet, computer, or specific software. [61]

This section offers a comprehensive overview of collaborative learning, including its definition, history, applications, and the evidence-based studies

---

<sup>1</sup>The full company name is 360Learning UK Ltd

<sup>2</sup>The full company name is EdApp Pty Ltd.

<sup>3</sup>The full company name is Slack Technologies owned by Salesforce Inc.

that support this learning method. Additionally, it outlines the advantages and disadvantages of collaborative learning. This information will serve as part of a solid foundation for designing a productivity-enhancing mobile application.

### 2.2.4 Gamification of Learning

This section is about *Gamification of Learning*, which is a method designed to enhance students' motivation and engagement by integrating elements typical of video games or any other game environment into educational content. This approach aims to make the learning process more enjoyable and encourages learners to engage with the material consistently. [62]

Gamification in education has been present in various forms for many years but has seen significant development over the last 50 years. In the 1970s, the release of multiplayer games with text-only interfaces marked the beginning of social online gaming, sparking the initial interest in the gamification of learning. This potential was recognized in the 1980s by academic researchers, who then began to further develop and research gamification. The concept of gamification as it is known today emerged in the early 2000s, with badge systems and social gamification becoming more prominent. Around the 2010s, gamification experienced a surge in popularity, driven by increased accessibility to suitable technology and the broader expansion of gamification strategies. [63]

#### 2.2.4.1 Foundations of the Method

Gamification in learning refers to the strategic application of game-like elements and mechanics in non-game contexts to enhance engagement, motivation, and problem-solving. Traditionally associated with fun and games, these elements are leveraged to encourage learning, foster behavioral change, and solve various problems in innovative ways. As various thought leaders and organizations have defined, gamification incorporates game mechanics such as levels, points, and badges into everyday activities to make them more interactive and enjoyable. This process is aimed at capturing and retaining interest by transforming mundane tasks into compelling challenges that offer feedback and rewards. By integrating game dynamics into learning or work environments, gamification seeks to stimulate participation, enhance performance, and promote continuous engagement. Through its psychological appeal, gamification turns routine interactions into stimulating experiences, driving individuals to achieve specific outcomes while enjoying the process. [62]

The term *Gamification Taxonomy* was introduced to classify and organize the various elements and approaches within gamification. This taxonomy is essential for differentiating the diverse strategies and components used to apply game mechanics in non-game contexts. When a gamification taxonomy is clearly defined, it can be effectively utilized in implementing gamification as a learning method in specific cases. It's important to note that there is not just one definitive gamification taxonomy; rather, there are multiple taxonomies, and they are continuously evolving as research progresses. [64]

One of the taxonomies introduced by a group of researchers in 2019 defines the taxonomy across five dimensions: *Performance*, *Ecological*, *Social*, *Personal*, and *Fictional* [5]. Each dimension encompasses its own unique *aspects* (also referred as *elements* [5]), as depicted in Figure 2.6.

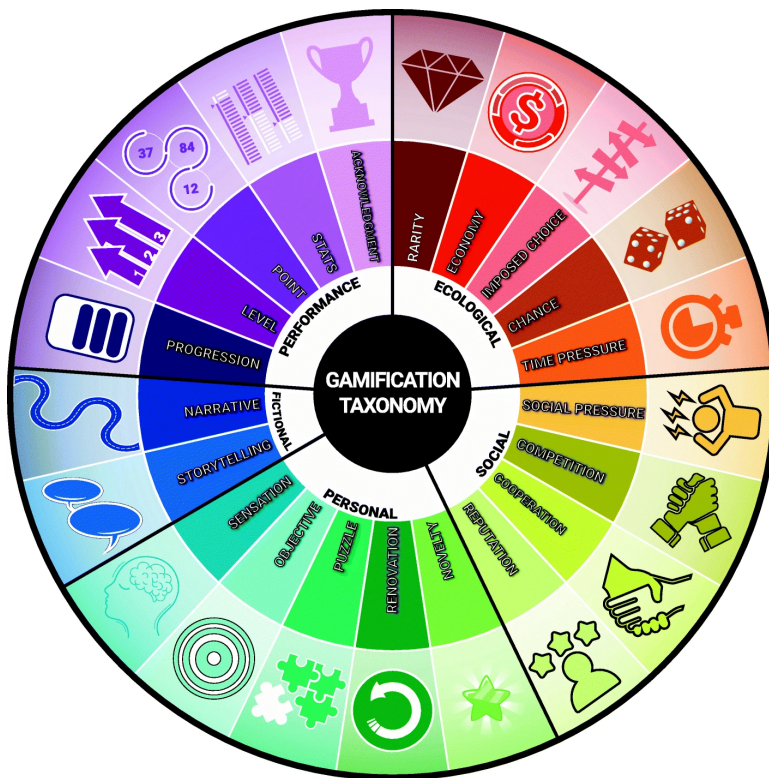


Figure 2.6: Gamification Taxonomy [5]

1. **Performance:** This dimension focuses on elements that provide feedback to learners about their progress and achievements. This includes points, progression metrics, levels, statistical feedback, and acknowledgments. These elements are crucial as they help learners understand the impact of their actions and their progress within the learning environment. Without such feedback, learners may feel disoriented and unsure about the effectiveness and relevance of their activities, as there is no mechanism to indicate success or areas for improvement. This dimension consists of *Acknowledgement*, *Level*, *Progression*, *Point* and *Stats* aspects.
2. **Ecological:** The *Ecological* dimension in gamification pertains to the environment in which the gamification is implemented, represented through various properties that affect user interaction. Key elements in this dimension include *Chance*, *Imposed Choice*, *Economy*, *Rarity*, and *Time Pressure*. These components are essential for creating a dynamic and engaging environment, as they foster interactions and challenge the user. Without these ecological elements, the gamification setting may appear dull and fail to stimulate active participation or sustained interest from users.

3. **Social:** This dimension in gamification focuses on the interactions among learners within the environment. Key elements of this dimension include *Competition*, *Cooperation*, *Reputation*, and *Social Pressure*. These elements are crucial for fostering a sense of community and engagement among participants. They enable learners to interact, compete, collaborate, and build reputations within the learning context. A lack of social elements can lead to isolation among students, as it prevents them from engaging with peers and benefiting from the motivational and educational advantages of social interactions.
4. **Personal:** The *Personal* dimension relates to the individual learner and their direct engagement with the environment. This dimension includes elements such as *Sensation*, *Objective*, *Puzzle*, *Novelty*, and *Renovation*. These elements are integral in providing a tailored and meaningful experience that resonates with the learner's interests and goals. They help to stimulate personal motivation and satisfaction by aligning the gamification elements with individual needs and preferences. A deficiency in these personal elements can lead to demotivation, as the system may fail to provide meaningful and relevant experiences that captivate and engage the learner.
5. **Fictional:** The last dimension is a hybrid category that intersects with both the user and the environment. It primarily involves the use of *Narrative* and *Storytelling* to connect the user's experience with the broader context of the activity. This dimension is crucial for providing a cohesive and immersive setting that explains the rationale behind tasks and enhances the overall quality of the user experience. When fictional elements are lacking, there can be a significant loss of context and meaning, making it unclear why tasks need to be performed. This absence can diminish the depth and impact of the experience, directly affecting user engagement and satisfaction. [5]

### 2.2.4.2 Evidence-based Studies

There are numerous studies supporting the positive effects of gamification on enhancing interest in learning [65], as well as improving cognitive, motivational, and behavioral learning outcomes [66], among others. However, some of these studies also indicate that the results can be variable, meaning that they depend on the context and subject matter, which may lead to differing outcomes [67].

### 2.2.4.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of gamification in learning.

#### Gamification Learning Advantages:

- Gamification transforms learning into an enjoyable and engaging activity, encouraging wholehearted participation and fostering a competitive yet friendly atmosphere among learners. This emotional connection to the content significantly increases knowledge retention.

- Through the use of game elements like badges and rewards, gamification motivates learners to achieve course objectives and continue progressing through learning materials. These rewards provide a sense of accomplishment and can enhance the learner’s self-esteem and perception of skill.
- Gamification allows for instant feedback, helping learners understand what they know and where they need to improve. Tools like leaderboards also offer additional feedback, showing learners how they compare to peers, which can motivate further improvement and engagement.
- Gamification accommodates different learning speeds, allowing learners to engage with content at a pace that suits them best. This personalized approach helps maximize the learning experience by aligning with each individual’s comfort level and capacity. [68]

### **Gamification Learning Disadvantages:**

- Effective gamification requires that games are interactive and engaging, and not merely quizzes disguised as games. Creating truly engaging game content that resonates with and motivates learners can be challenging and requires a deep integration of learning objectives with game design.
- The use of points, badges, and leaderboards, often referred to as *pointsification*, can sometimes lead to negative effects such as lack of motivation, irrelevance, and even worsened performance. There are also concerns about ethical issues, such as gaming the system and cheating.
- Introducing gamification in educational and training settings can inadvertently foster a competitive environment that may not align with organizational goals that emphasize cooperation and teamwork. This misalignment can create a culture clash where individual achievements are prioritized over collaborative success. [68]

#### **2.2.4.4 Summary of Gamification in Learning**

All the previously mentioned methods facilitate more engaging, interactive learning experiences that keep learners motivated and effective. Gamification of learning method and its strategies, such as reward systems, levels, and interactive challenges, naturally complement digital environments, making them ideal for implementing in educational technologies. Technology together with gamification have the potential to revolutionize education, providing a dynamic and effective approach to teaching complex subjects. [69]

The section on gamification of learning discusses how integrating game-like elements into educational settings boosts student engagement and motivation. It applies these elements across various dimensions—performance, ecological, and social—to improve effectiveness. Studies generally support gamification’s positive impact on learning outcomes, though results can vary with context and implementation. This section can serve as reference material when designing a productivity-enhancing mobile application.

### 2.2.5 Pomodoro Technique

This section begins with a comprehensive introduction to the *Pomodoro Technique*, providing detailed insights into its origins and development. It explores the foundational concepts behind the timer, its historical context, and its primary functionalities. Additionally, the section explains how the *Pomodoro Timer* can be effectively implemented in various settings, discussing its benefits in enhancing productivity and focus. The introduction aims to equip readers with a solid understanding of how this time management tool can be integrated into their daily routines for optimal performance.

#### 2.2.5.1 Foundations of the Method

The Pomodoro technique, as detailed by its creator, F. Cirillo, is a productivity method that emphasizes the division of work into finite intervals known as Pomodoros. This technique is designed to help individuals break down their workday into manageable segments, thereby facilitating a structured approach to tasks. Each Pomodoro session typically lasts for 25 minutes, followed by a short break (traditionally 5-minutes). This structure not only aids in maintaining concentration but also allows for tracking and recording the number of Pomodoros spent on each task. By seeing the amount of time dedicated to different activities, users can gauge their productivity and plan their schedules more effectively, providing a tangible measure of task completion and progress. [70]

#### 2.2.5.2 Evidence-based Studies

Research has explored various aspects of the Pomodoro technique's effectiveness, particularly in combating common workplace issues like procrastination and lack of focus. For instance, a study conducted by Dizon et al. observed that while the Pomodoro technique helped nursing students reduce procrastination, it also slightly diminished their motivation [71]. Meanwhile, another study by Sarkar et al. suggested that the technique is beneficial in managing mental fatigue among developers during intensive coding sessions [72]. These studies highlight that while the Pomodoro technique can significantly enhance focus and reduce the inclination to procrastinate, the impacts on motivation can vary, suggesting the need for additional motivational elements such as gamification to further optimize its benefits.

#### 2.2.5.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of Pomodoro technique.

##### **Pomodoro Technique Advantages:**

- Forces regular breaks before feeling overwhelmed, using breaks for simple, relaxing activities to maintain focus.
- Encourages breaking tasks into smaller, manageable parts, improving productivity and planning.



- Helps in better planning and dedicating more effort to tasks, especially useful in multi-step projects.
- Timers provide a sense of urgency that can motivate and help maintain focus. [73]

### **Pomodoro Technique Disadvantages:**

- The short, 25-minute intervals can disrupt focus just as engagement increases, though longer intervals can be adapted.
- Particularly in public or office settings, external disturbances can disrupt the technique's effectiveness.
- Without prior planning, it can be challenging to decide what to work on, making spontaneous task initiation difficult.
- The structured nature of scheduled work and break times can be restrictive for those who prefer flexibility or struggle with routine adherence. [73]

### **2.2.5.4 Summary of Pomodoro Technique**

This section explored the Pomodoro technique, initially outlining its development by Francesco Cirillo to improve productivity by structuring work into 25-minute intervals, each followed by short breaks. It discusses the method's benefits in maintaining focus and organizing tasks, but also notes potential drawbacks, such as reduced motivation and the disruptive nature of its rigid timing in certain environments. Research suggests the technique's effectiveness in reducing procrastination and managing fatigue, though it may not suit everyone due to its structured approach. The section highlights the technique's utility and challenges, suggesting a need for flexible application.

### **2.2.6 Other Methods**

In addition to active learning, spaced repetition, collaborative learning, gamification in learning, and the Pomodoro technique, several other effective learning methods can be utilized. *Dual Coding* combines verbal and visual information to enhance memory and understanding. *Interleaved Practice*, which involves mixing different topics or forms of practice, can improve problem-solving skills and long-term retention. *Elaborative Interrogation* encourages learners to generate explanations for why stated facts are true, fostering deeper engagement with the material. These methods diversify learning approaches and can enhance educational outcomes across various disciplines. [74, 75]

## **2.3 Individual Differences**

Individual differences in learning refer to the unique ways in which individuals acquire, process, and retain information. These differences encompass a wide range of factors, including cognitive abilities, personality traits, motivational tendencies, and socio-cultural backgrounds. Understanding these variations is crucial for designing effective educational interventions that cater to diverse

learning needs. For instance, some individuals may excel in visual learning tasks, while others may prefer auditory or kinesthetic approaches.

Moreover, factors such as attentional control, working memory capacity, and learning styles play significant roles in shaping how individuals engage with educational material. Learning is a socially mediated process, influenced by interactions with peers, teachers, and the broader socio-cultural context. [76] Thus, a comprehensive understanding of individual differences in learning requires consideration of both cognitive and socio-cultural factors. By acknowledging and accommodating these differences, educators can create inclusive learning environments that support optimal learning outcomes for all students. [77]

### 2.4 Learning and Technology

The integration of technology into learning environments has revolutionized educational practices, offering diverse avenues for knowledge acquisition and engagement. Digital tools provide personalized learning experiences tailored to individual preferences and abilities, fostering self-directed learning [78]. Virtual simulations, multimedia resources, and online collaboration platforms enhance understanding and retention by catering to different learning styles and preferences [79]. Furthermore, technology facilitates access to vast repositories of information, enabling learners to explore topics in depth and at their own pace [80]. However, the effective utilization of technology in education requires careful consideration of factors such as digital literacy, access to resources, and equitable distribution of learning opportunities [81]. By leveraging technology thoughtfully, educators can create dynamic learning environments that empower students to thrive in the digital age.

### 2.5 Pomodoro Technique Research

This section offers a detailed overview of a research study conducted by the author of this thesis and his classmates in Montreal, Canada, in December 2022. The environment of exchange students from various countries, along with local Canadian students who attended the same classes, provided a unique opportunity to conduct research with a broad audience featuring diverse opinions, cultural differences, and attitudes. The research was carried out by Petr Šmejkal, Lucia Čahojová, and Diederik Ponfoort. At the time, the topic was specifically chosen to provide a foundational basis for developing the productivity-enhancing application discussed in this thesis. The output document of the research is included in the electronic attachments, as detailed in Appendix G.

Initially, the concept of the research is introduced, followed by the presentation of the motivation and research questions. This is followed by a description of the methodologies employed, concluding in a discussion of the results obtained from these methods.

#### 2.5.1 Introduction

The recent surge in the popularity of tools designed to enhance focus and motivation can be attributed not only to their increased accessibility and user-

friendliness but also to the positive experiences reported by their users. Among these tools, the Pomodoro technique stands out as a particularly influential method. This approach has evolved over time, adapting to modern needs through various applications, each offering different features from engaging user interfaces and customizable settings to elements of gamification, catering to a diverse range of preferences and enhancing user engagement.

The necessity for such tools has become more pronounced in today's environment, where distractions are ubiquitous, particularly through digital channels like social media, streaming services, and video games. These distractions contribute significantly to procrastination, particularly when work and leisure devices overlap. This research aims to delve deeper into the motivations and habits that drive user engagement with productivity tools, seeking insights that could guide the development of applications that resonate more deeply with user needs. This focus is especially pertinent in the post-pandemic era, where many continue to work from home, necessitating a reevaluation of work habits and the promotion of strategies that support sustained concentration and effective task management.

### 2.5.2 Motivation and Research Questions

Our environment is filled with distractions like social media, streaming apps, and video games, which often lead to procrastination as they compete with our work tasks, especially since most of our work is done on phones and computers. To address the challenges of lost focus and the shift in habits post-pandemic, many tools and techniques have been developed as mobile or desktop applications aimed at fostering healthier work habits and improving focus. The study aims to explore people's habits and motivations in detail to tailor these tools to user needs more effectively. Specifically, it assesses the impact of the Pomodoro technique and its gamified versions, which might include interactions with virtual animals or plants, to see if these can enhance focus and productivity more than traditional methods.

#### Research Questions

- **RQ1:** What are peoples' approaches to accomplishing their academic or work tasks?
- **RQ2:** How does the Pomodoro Technique help people accomplish their academic/work tasks?
- **RQ3:** How does someone's motivation change when the Pomodoro technique with the element of gamification is used?

### 2.5.3 Methods

This section describes the methods used to evaluate the Pomodoro technique's effectiveness through two phases: an *interview* and a *survey study*. The interview involved four participants discussing their experiences with the technique in work and study environments, analyzed qualitatively to identify themes. The survey, involving eight participants, delved deeper into usage patterns of both

traditional and gamified Pomodoro techniques, exploring participant strategies, satisfaction, and specific applications to understand its practical benefits comprehensively.

### 2.5.3.1 Interview Study

In a recent study focused on exploring the efficacy of the Pomodoro technique among office workers and students, four participants were recruited through convenience sampling. These individuals, averaging 29 years in age and having varied experience with the Pomodoro technique—from one week to a few months—participated in semi-structured online interviews that lasted about 30 minutes each. All participants were either pursuing or had completed higher education degrees and were fluent in English. The interviews aimed to capture their experiences with the Pomodoro technique, delving into their current work or study environments and any modifications they might consider beneficial to their productivity methods.

The interview process was structured to gradually deepen into the subject matter, beginning with a warm-up session where participants described their professional or academic background and their familiarity with the Pomodoro technique. The discussion then moved on to how participants handle their workload and their typical work or study environments. Following this, the conversation focused on their specific experiences with the Pomodoro technique, including a comparison between the traditional and a gamified version of the technique, which incorporates elements like caring for a virtual plant or animal. For those unfamiliar with the gamified version, it was briefly explained, and their expectations were discussed.

Data from these interviews were analyzed through *qualitative content analysis*, using an *inductive approach* to identify themes across the conversations. Codes were assigned to specific interview sections, allowing for the comparison and extraction of common themes. These findings were summarized in a codebook, providing a structured overview of the insights gained regarding the use and perception of the Pomodoro technique among the participants. This methodological approach helped to pinpoint the potential benefits and preferences regarding productivity techniques among young professionals and students.

### 2.5.3.2 Survey Study

A follow-up survey study was designed to delve deeper into the effectiveness of the Pomodoro technique based on findings from prior interviews. This survey collected data from eight participants, predominantly in their early twenties, with educational backgrounds ranging from high school to master's degrees. The gender distribution included five females and three males. The survey aimed to understand participants' usage patterns with both the traditional and gamified versions of the Pomodoro technique, exploring frequency and context of use.

The survey methodology employed *convenience sampling* and was structured into four distinct sections. The initial part gathered demographic data and provided background on the Pomodoro technique. The subsequent sections delved into participants' current strategies for managing workloads, their

satisfaction with these methods, and specific usage details about the Pomodoro technique, both traditional and gamified. Questions focused on the frequency of use, perceived stress levels, task suitability, and overall work-life balance. The final section invited participants to compare their experiences with both versions of the Pomodoro technique, aiming to identify preferences and effectiveness.

Data from the survey were analyzed by correlating responses to several research questions designed to assess how participants manage their workloads, the perceived utility of the Pomodoro timer, and the comparative effectiveness of its gamified version. The analysis involves creating visual representations of the data, such as graphs, to facilitate a clear understanding of trends and conclusions. This structured approach aims to provide insights into the practical application and benefits of the Pomodoro technique in real-world settings.

### 2.5.4 Results and Discussion

This section presents results from studies on the Pomodoro technique's effectiveness in managing work and academic tasks. Through interviews and a survey, varied task management strategies and environmental and their impact on productivity were explored. The findings illuminate how individuals tailor their work habits to enhance efficiency and satisfaction, highlighting preferences from structured task lists to gamification elements.

#### 2.5.4.1 Interview Study Findings

The findings from an interview study exploring the use of the Pomodoro technique revealed a variety of themes related to environment, task management, and overall satisfaction with personal work habits. Participants highlighted the importance of their environment in facilitating productivity, with preferences varying from quiet spaces like libraries to dynamic ones like offices, though each setting had its own set of distractions and benefits. Regarding task management, approaches varied significantly; some participants utilized structured methods like checklists and to-do lists to feel productive, while others had no specific strategy and relied on situational motivation. This varied approach also extended to the Pomodoro technique itself, where opinions on its effectiveness depended on individual work habits and task types.

The study further delved into the nuances of using the Pomodoro timer and its gamified versions. While some found the traditional timer format stressful and interruptive, particularly when it cut into work flow or creative processes, others appreciated the structure it provided, especially for intensive tasks like studying for exams. The gamified version, which included elements like growing virtual trees, was particularly motivating for some as it added a layer of visual progress and environmental contribution. However, others found these features distracting, preferring simpler, less intrusive methods to manage work periods.

In conclusion, while the Pomodoro technique and its gamified enhancements received mixed reviews, they were generally more favored for academic tasks over creative professional work. Participants' satisfaction with their productivity techniques varied, with many acknowledging room for improvement in managing procrastination and work-life balance. These insights reflect the complex

interplay between personal preferences, work environment, task type, and productivity strategies, underscoring the importance of tailoring time management tools to individual needs.

#### 2.5.4.2 Survey Study Findings

In this survey study, it was explored that different approaches individuals adopt to manage their academic or work-related tasks, addressing the first research question. The predominant strategies identified were using task lists, establishing routines, and prioritizing tasks according to importance. The data, as depicted in Figure 2.7, shows that task lists are the most favored method, utilized by the majority of participants.

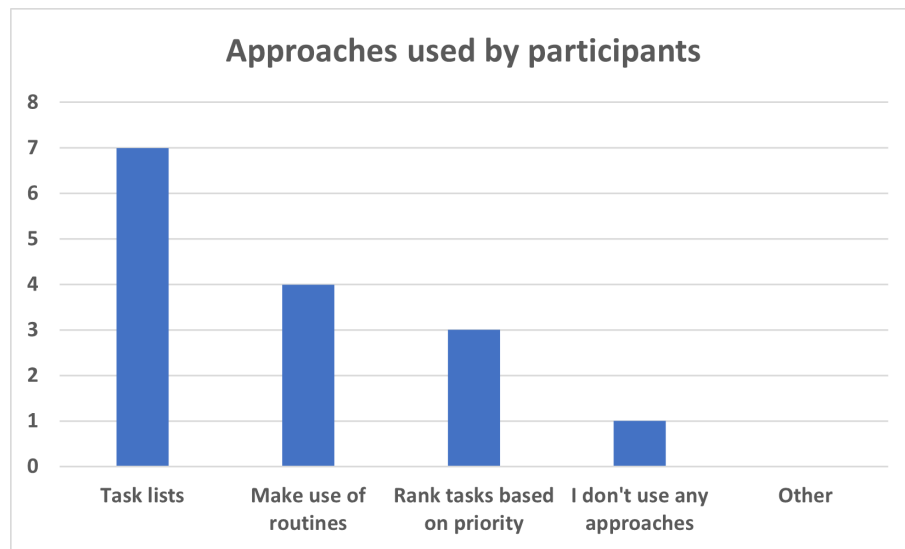


Figure 2.7: Work approaches of the Survey Participants

Furthermore, the survey probed into the participants' satisfaction with their chosen methods. As illustrated in Figure 2.8, the responses ranged from neutral to strongly satisfied, indicating a general approval of their methods for managing tasks.

The second research question delved into the effectiveness of the Pomodoro technique in aiding task accomplishment. Participants evaluated the applicability of this technique to different types of tasks, particularly highlighting its benefits for structured tasks like reading and writing, as shown in Figure 2.9.

However, it was noted that creative tasks might not be as suitable for this method. Additionally, the impact of the Pomodoro technique on enhancing focus, motivation, and overall workflow efficiency was positively acknowledged by the respondents. The influence of the Pomodoro technique on these aspects is further detailed in Figure 2.10, suggesting its utility in maintaining concentration and aiding in workflow integration.

The third research question explored the impact of incorporating gamification into the Pomodoro technique on participants' motivation and focus.

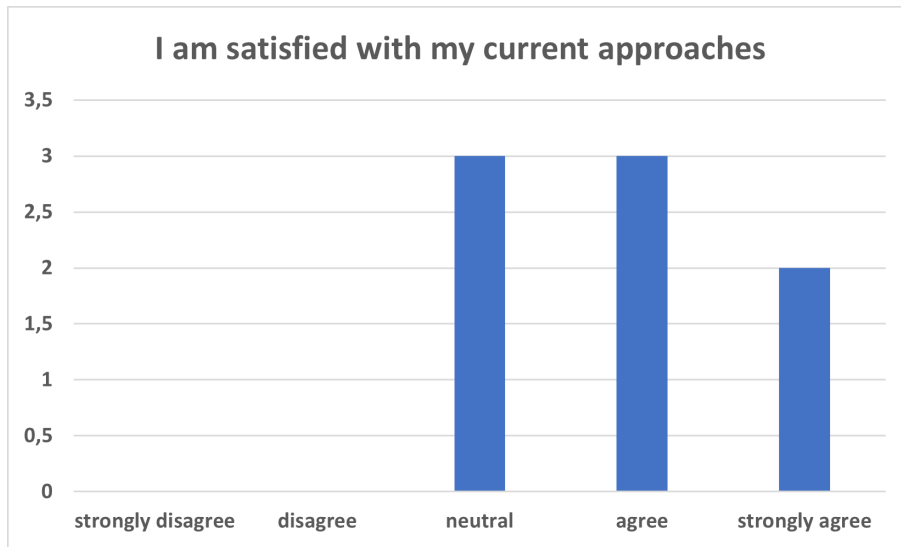


Figure 2.8: Survey Participant’s Satisfaction With Their Study Approaches

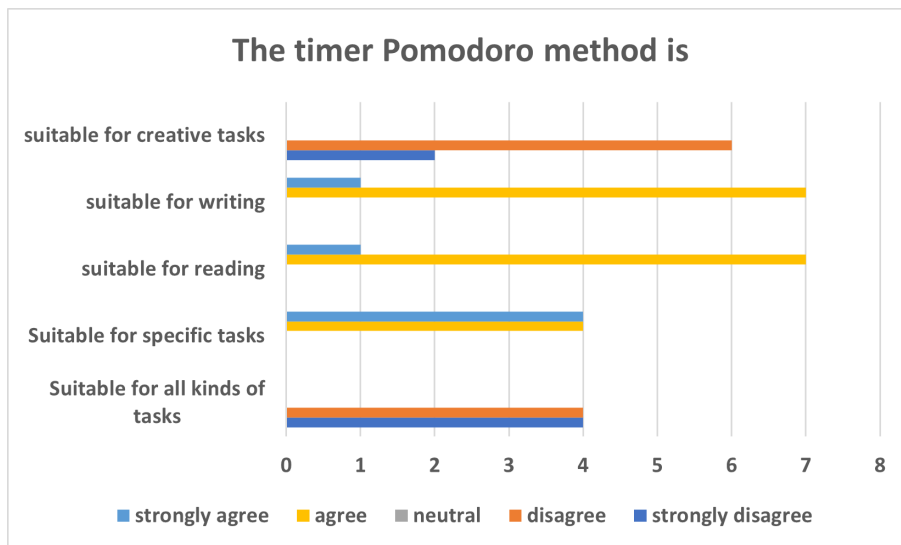


Figure 2.9: Pomodoro Timer Usefulness For Different Tasks

The survey results, as depicted in Figure 2.11, show that the gamified version of the Pomodoro technique has a predominantly positive impact, with half of the participants strongly affirming an increase in their efficiency and motivation compared to the classical version. This indicates that the addition of game-like elements can significantly enhance the effectiveness of the Pomodoro technique, making task completion not only more engaging but also

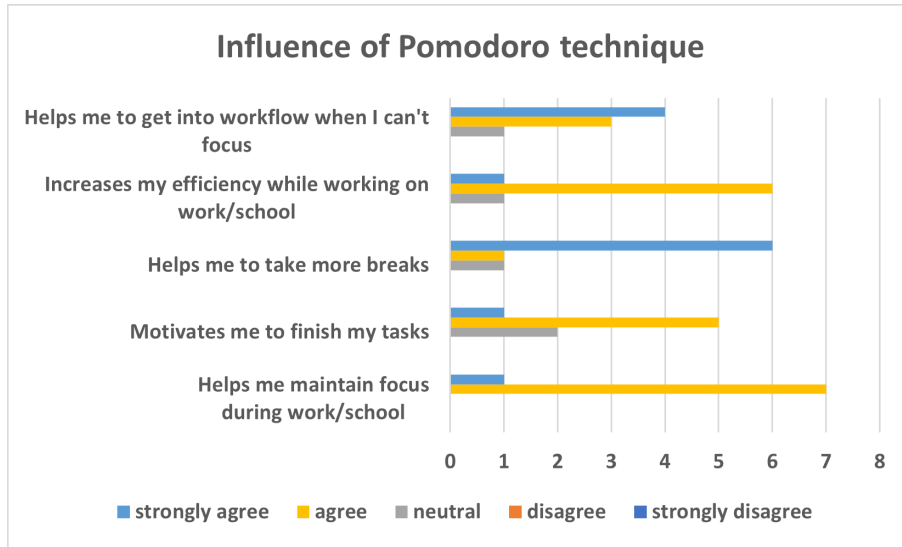


Figure 2.10: Influence of the Pomodoro Technique

more productive.

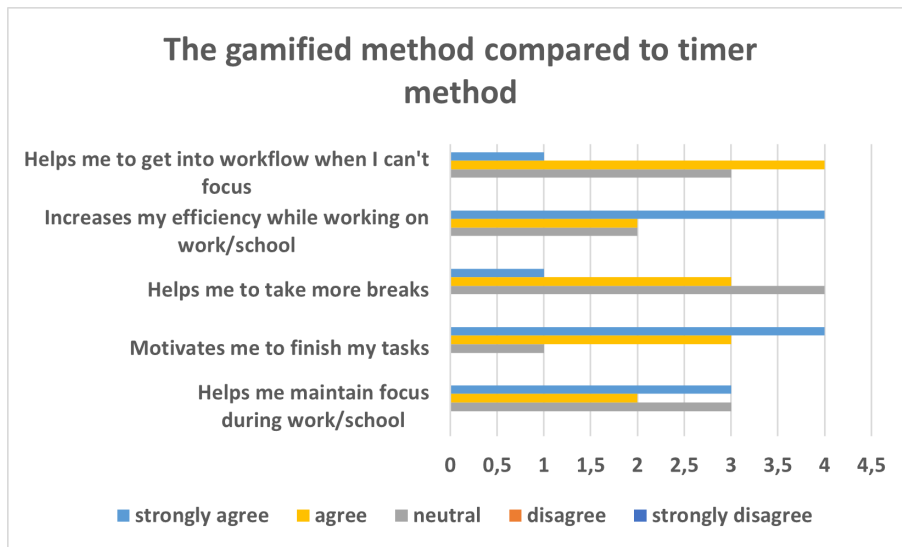


Figure 2.11: Comparison of Gamified and Non-Gamified Pomodoro Technique

In summary, the study not only highlighted preferred learning strategies and their satisfaction levels but also validated the effectiveness of the Pomodoro technique for certain types of tasks. The integration of this technique into regular work routines has been shown to positively affect focus and efficiency.



### 2.5.4.3 Summary of Results

The findings indicate that while many are somewhat satisfied with their current work approaches, there is room for improvement, particularly for those working from home who struggle with initiating tasks. The Pomodoro technique and its variations, including gamified elements, can enhance productivity and make work more engaging. However, this technique is not universally suitable. Therefore, it is suggested to explore and incorporate variety of methods to find what best enhances workflow. A potential improvement would be developing applications that offer not only the Pomodoro technique but also other task-based methods, allowing users to tailor their productivity tools to fit various tasks, such as creative work or memorizing, within a single, versatile application. This could simplify usage and help in developing more effective work habits.

### 2.5.5 Research Conclusion

The study explored work behaviors and task management, focusing on the Pomodoro technique and its gamified version to assess their impact on motivation. Initial insights from semi-structured interviews with four participants were validated through a survey with eight participants. While the Pomodoro technique showed varied effectiveness, the gamification element emerged as a significant motivator, offering visual progress tracking. Limitations included a small sample size and participant bias, suggesting the need for larger, more diverse studies. Future research could explore specific task contexts and develop versatile productivity applications.

## 2.6 Summary of Learning Approaches and Cognitive Processes

This chapter has explored foundational learning approaches and cognitive processes, emphasizing their significant impact on educational practices. It highlights how deep understanding and strategic integration of these theories can enhance learning strategies and teaching methods, thereby improving educational outcomes. Additionally, the chapter includes a tailored study on the Pomodoro technique, extending beyond typical Software Engineering analyses to inform the design of a productivity-enhancing iOS application. This comprehensive exploration serves as a robust foundation for future application development.



---

## Domain Analysis

This chapter delves into the domain analysis necessary for the development of a productivity-enhancing iOS application. Domain analysis is a crucial step in Software Engineering as it helps to understand the competitive landscape, identify user needs, and define system requirements comprehensively. By evaluating existing applications and their features, this analysis informs the design and development process, ensuring the final product is well-positioned in the market and meets user expectations effectively. This chapter covers several key areas, including a detailed competition analysis, domain modeling, application requirements specification, and use case scenarios, each critical for laying a solid foundation for the application's success.

### 3.1 Competition Analysis

This section presents an analysis of the competition and similar solutions (for the iOS platform) related to the mobile application discussed in this thesis. By examining existing apps in the market, this research aims to provide insights that will inform the further design and development of the mobile app. Identifying what features are well-received by users and which ones are not, drawing valuable lessons from both. The analysis of competitors is a fundamental step in the standard Software Engineering process, ensuring that the application is well-positioned to meet user needs and stand out in a marketplace. This thorough evaluation not only helps in enhancing the app's functionality but also in adopting best practices and avoiding common pitfalls observed in similar applications. The selection of competitors was strategic, focusing on market leaders and innovators to derive lessons from their successes and shortcomings, and to benchmark against established standards in the industry.

#### 3.1.1 Forest: Focus for Productivity

Currently ranked number one in the Productivity app category on the App Store, the app *Forest: Focus for Productivity* is highly popular. Forest is designed to enhance users' focus and time management effectively with a simple yet engaging concept: users grow a virtual forest by staying focused and avoiding non-essential phone use. A detailed breakdown and review of its features and functionality will follow in this section.

#### 3.1.1.1 How It Works

Users start by setting a timer in the Forest app, choosing how long they want to focus, which can vary from 10 minutes to several hours based on the task. Once the timer begins, a virtual tree starts to grow on the user’s screen and will continue to grow as long as the user remains in the app and refrains from using their phone for other activities. Each successful focus session contributes a new tree to the user’s virtual forest, allowing them to visually track their accumulated focused time through the development of a lush virtual forest over time. [82, 6]

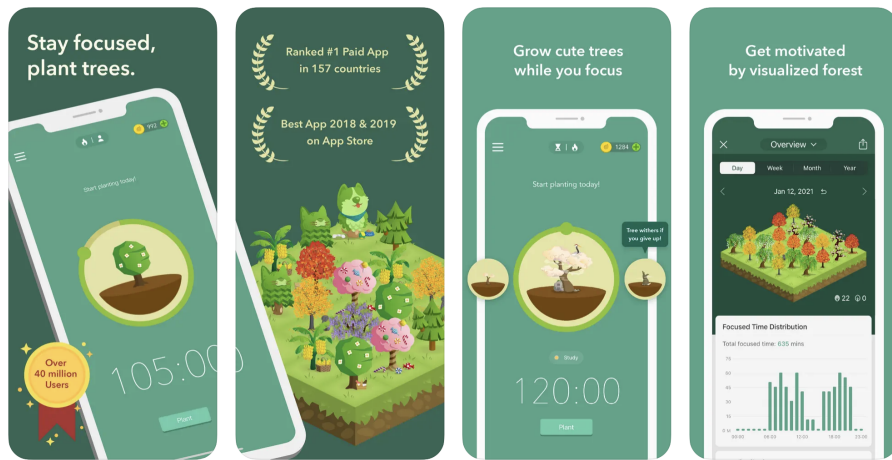


Figure 3.1: Screenshots of Forest: Focus for Productivity [6]

#### 3.1.1.2 Key Features

- **Virtual Forest:** The core feature of the app is the virtual forest that grows as users spend time away from their phone. This provides a visual incentive to stay focused and avoid distractions.
- **Variety of Tree Species:** Users can unlock different species of trees as they accumulate focus time, adding variety and interest to the virtual forest.
- **Statistics and Tracking:** Forest provides detailed statistics about users’ focus habits, including daily, weekly, and monthly summaries. This allows users to track their progress and set goals.
- **Rewards and Achievements:** The app offers rewards and achievements to motivate users. These can include new tree species or other virtual items as users reach certain milestones.
- **FocusWhitelist:** Users can create a whitelist of apps that they can access without stopping the tree from growing. This is useful for those who need certain tools or apps for their work.

- **Collaborative Focus:** Users can join rooms with friends or colleagues to focus together. This feature helps in creating a community of focused individuals, providing mutual support and motivation.
- **Real-World Impact:** Through partnerships with organizations focused on reforestation and environmental sustainability, Forest allows users to spend virtual coins earned during focus sessions to plant real trees around the world.
- **Forest combines gamification with productivity**, making it an engaging tool for those looking to improve their focus and manage their time more efficiently. Its simple yet effective design has made it a popular choice among students, professionals, and anyone looking to reduce distractions and enhance productivity. [83]

#### 3.1.1.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of the Forest app.

##### Forest Advantages:

- Eliminates interruptions from incoming messages and notifications.
- Offers a variety of extra trees.
- Real-world trees are planted when enough coins are collected.

##### Forest Disadvantages:

- User interface lacks intuitiveness.
- Requires a large number of coins to access extra trees or plant an actual tree.
- Includes a whitelist option that permits exceptions, which may dilute its effectiveness.

#### 3.1.1.4 Summary of Forest Analysis

The section presents an analysis of Forest: Focus for Productivity, a top app in the Productivity category. It outlines the app's core functionality, encompassing its virtual forest concept and key features like diverse tree species, focus statistics, rewards, and collaborative sessions. The advantages and disadvantages of the app are assessed, with particular attention to its ability to minimize interruptions and its interface usability. This analysis adheres to standard processes in Software Engineering and will inform the design of the productivity-enhancing mobile application.

#### 3.1.2 Focus To-Do: Focus Timer&Tasks

The analysis of the *Focus To-Do: Focus Timer&Tasks* iOS application explores its fundamental operational approach, which revolves around enhancing users' productivity and focus. This examination delves into the core mechanisms driving the app's functionality, emphasizing its overarching goal of facilitating improved task management and concentration among its users. [7]

##### 3.1.2.1 How It Works

Users initiate their productivity sessions by setting timers for their tasks, prompting the app's focus timer to activate. Through features such as task lists and notification blocking, distractions are effectively minimized, enabling users to concentrate fully on their designated tasks. Progress is meticulously tracked through visual indicators, including completed tasks and time allocation for each activity, offering invaluable insights into users' productivity habits. Additionally, the app provides users with a suite of tools for task organization, prioritization, and scheduling, empowering them to efficiently manage their time and accomplish their goals with heightened focus and efficacy. [7, 84]

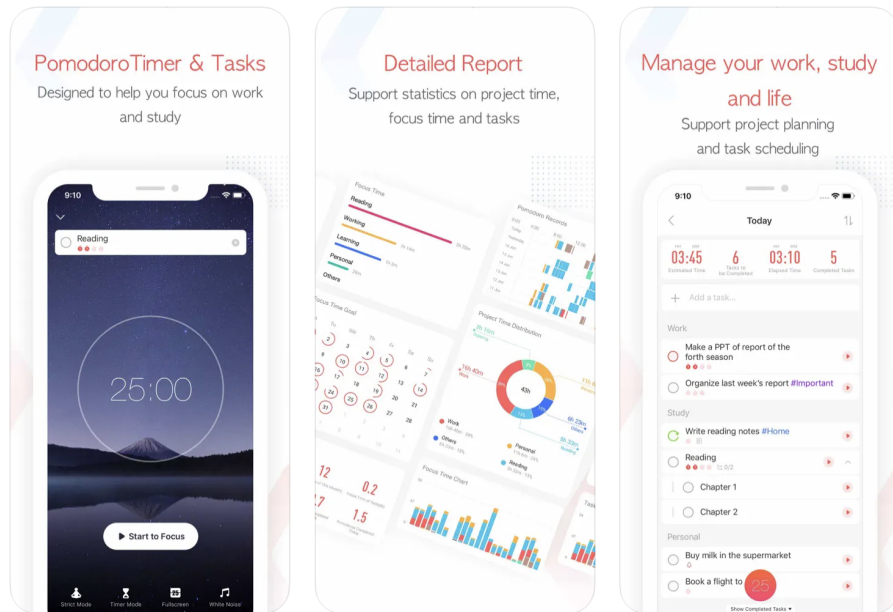


Figure 3.2: Screenshots of Focus To-Do: Focus Timer&Tasks [7]

##### 3.1.2.2 Key Features

- **Focus Timer:** Utilizes a timer-based system to help users concentrate on their tasks for specific durations.
- **Task Lists:** Allows users to create, organize, and prioritize their tasks, ensuring clarity and focus on what needs to be accomplished.

- **Notification Blocking:** Enables users to minimize distractions by temporarily blocking notifications during focus sessions.
- **Progress Tracking:** Provides visual indicators and progress statistics to help users monitor their productivity and stay motivated.
- **Task Organization:** Offers tools for task categorization, setting reminders, and scheduling activities to enhance task management and efficiency.
- **Pomodoro Technique Support:** Integrates the popular Pomodoro technique, which involves timed work intervals followed by short breaks, to optimize productivity.
- **Focus Insights:** Provides insights into users' focus habits and productivity trends through analytics and reports, facilitating self-improvement and goal setting. [84, 85]

#### 3.1.2.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of the Focus To-Do app.

##### Focus To-Do Advantages:

- Features like focus timers and task lists help minimize distractions.
- Application offers efficient organization, prioritization, and scheduling of tasks.
- Visual indicators and statistics offer valuable productivity insights.

##### Focus To-Do Disadvantages:

- Some users find the design chaotic and distracting, making it difficult to focus effectively.
- Some users have experienced instances where all their tasks and study hours were deleted, leading to frustration and dissatisfaction.
- The app lacks flexibility for users to revert to the older, simpler red version, indicating a need for improved customization options.

#### 3.1.2.4 Summary of Focus To-Do Analysis

This section starts by detailing the functionality of the Focus To-Do app, highlighting tools that boost productivity. It covers key features such as focus timers and task lists, which are designed to minimize distractions and enhance task management. The section concludes with a table that summarizes the app's strengths and weaknesses, providing a comprehensive view of its effectiveness in promoting focus and efficiency. The analysis employs standard Software Engineering processes and will inform the design of a productivity-enhancing iOS app.

### 3.1.3 Study Bunny: Focus Timer

This section introduces the *Study Bunny* app, an engaging iOS application that enhances the studying experience by blending gamification with task management. The app features a customizable digital bunny to motivate users during study sessions. Equipped with tools like timers and progress tracking, Study Bunny turns routine learning into a fun and rewarding activity, promoting effective study habits through interactive incentives. [8]

#### 3.1.3.1 How It Works

The Study Bunny app motivates users by integrating gamification elements into its task management system. Users start by setting specific study goals and tasks, which are represented by a digital bunny that they can interact with and nurture. Completing tasks earns users coins, which can be used to buy accessories for their bunny or unlock new features. The app includes a timer to encourage focused study sessions using techniques like the Pomodoro technique and provides statistics to monitor progress over time. This playful approach not only makes studying more engaging but also aids in maintaining focus and managing time efficiently. [8, 86]



Figure 3.3: Screenshots of Study Bunny: Focus Timer [8]

#### 3.1.3.2 Key Features

- **Customizable Digital Bunny:** Users can personalize their digital bunny, enhancing engagement and motivation during study sessions.
- **Task Management:** Allows users to set specific study goals and tasks, organizing their learning schedule effectively.
- **Reward System:** Completing tasks earns coins that can be spent on accessories for the bunny or to unlock new app features.



- **Focus Timer:** Integrates a timer with options like the Pomodoro technique to help users maintain focus and manage study time efficiently.
- **Progress Tracking:** Provides visual statistics to track and analyze users' study habits and progress over time. [86, 87]

#### 3.1.3.3 Advantages and Disadvantages

Below is a list of the advantages and disadvantages of the Study Bunny app.

##### Study Bunny Advantages:

- Engages users with a customizable digitalbunny, making the learning process fun and interactive.
- Includes a built-in timer that supports focused study sessions through the Pomodoro technique.
- Completing tasks earns rewards, encouraging engagement and motivation.
- Features an intuitive design that simplifies task management and progress tracking.

##### Study Bunny Disadvantages:

- The app has restricted customization options, which may not satisfy all user preferences as well as offer displays distractive ads which interrupt studying.
- The reliance on rewards might not foster intrinsic learning habits, potentially affecting long-term engagement.
- Study Bunny can be demanding on device resources, potentially draining battery life and affecting performance on older devices. As well as occasional glitches and bugs appear in the app.

#### 3.1.3.4 Summary of Study Bunny Analysis

This section explores the Study Bunny app, an iOS application that enhances studying through gamification and task management. It highlights features like a customizable digital bunny, a Pomodoro timer, rewards, and progress tracking, which collectively make learning engaging and effective. The section concludes with a table summarizing the app's pros and cons, providing a clear view of its utility in promoting focused and enjoyable study habits. This analysis employs standard processes of Software Engineering and underscores the potential of gamification to improve educational tools and learning experiences.

## 3.2 Domain Model

A domain model describes the entities within a specific domain and their relationships, while remaining implementation-agnostic unlike class diagrams. It also includes the properties of these entities, offering a conceptual overview of the domain's structure for clearer communication and effective system design.

The following text in this section uses *UML (Unified Modeling Language)* according to the *ISO/IEC 19505-2:2012* standard, created by the *International Organization for Standardization*. [88]

### 3.2.1 Domain Model Notations

This subsection of the introduces and explains the different types of relationships that can exist between entities in a domain model. These relationships are crucial for understanding how entities interact with one another within a system, covering various forms of connections from basic associations to more complex hierarchical dependencies.

- **Class:** In UML, each entity is depicted as a *class*, featuring key attributes without methods to ensure the model remains platform-independent. Classes are illustrated as rectangles split into two parts: the upper section for the class name and the lower for its attributes. [9] An example of such a class is shown in Figure 3.4.

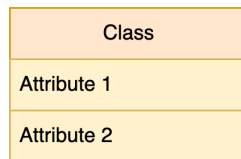


Figure 3.4: Class Example; based on [9]

- **Association:** In UML, *Association* represents a relationship between two independent entities, depicted as a solid line. Typically bi-directional, this can be modified to uni-directional by adding an arrow, showing that only one entity maintains a reference to the other. The example of a Car and Driver in Figure 3.5 illustrates such an association. [9]

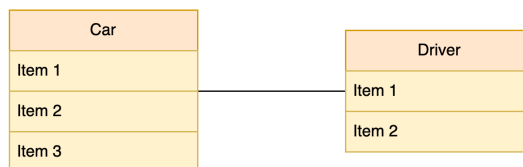


Figure 3.5: Association Example; based on [9]

- **Aggregation:** *Aggregation* in UML depicts the relationship between a whole and its parts, illustrated as a solid line with an empty diamond at the whole entity, like a section of an article. This entity holds a collection, and the part entities, such as articles in Figure 3.6, can exist independently and belong to multiple collections. The line's endpoints indicate the relationship's multiplicity, showing that a section can contain numerous articles, and each article must belong to at least one section. [9]

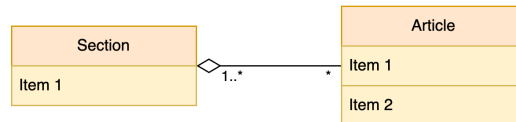


Figure 3.6: Aggregation Example; based on [9]

- **Composition:** *Composition* in UML represents a stronger relationship than aggregation, indicated by a filled diamond shape on the line connecting entities. It shows a dependency where parts cannot exist without the whole; if the whole is removed, its parts are also removed. For instance (as depicted in Figure 3.7), in a composition between Order and Order Item, the order item is meaningless without its associated order, unlike parts in an aggregation that can exist independently. The multiplicity for the whole in a composition is always 1, emphasizing its exclusive relationship with its parts. [9]

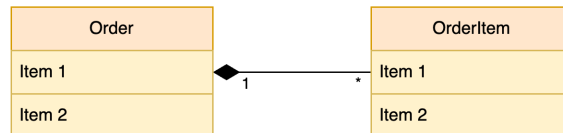


Figure 3.7: Composition Example; based on [9]

- **Generalization:** *Generalization* in UML is depicted as a solid line with an empty arrow, representing inheritance where one entity adopts the properties and behavior of another. For instance, in a model where Square and Circle classes inherit from a Shape class, the arrow points towards the Shape, indicating the source of inheritance. See Figure 3.8 to visualize the mentioned example. [9]

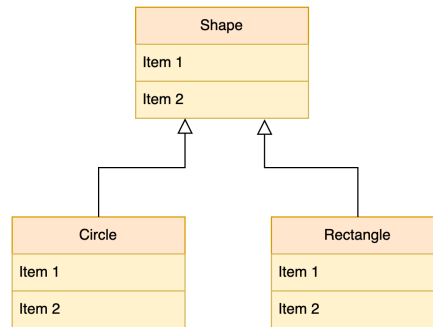


Figure 3.8: Generalization Example; based on [9]

- **Multiplicity:** *Multiplicity* in UML specifies the number of instances in relationships like association, aggregation, and composition. It determines how entities like sections and articles relate in terms of quantity. Here are the key points:
  1. **Specific Value (1):** Indicates an exact number of instances, such as one section or article.
  2. **Asterisk (\*) or N:** Represents any number of instances, including zero.
  3. **Interval (1..\*):** Defines a range using two numbers, commonly separated with dots, allowing for expressions such as 1..\*, 2..6, or 0..1 to specify minimum and maximum counts.

Multiplicity can also be combined in formats such as "1, 2, 3, 7..\*", covering specific numbers or any number starting from a threshold. Moreover, multiplicity defaults to 1. [9]

In summary, this section offers a foundational understanding of the entities and relationships within a specific domain, utilizing UML according to the ISO/IEC 19505-2:2012 standards. It outlines various types of relationships crucial for system design, including association, aggregation, composition, and generalization. This comprehensive framework will now be used in this section to model a domain for the productivity-enhancing application that is the subject of this thesis. This domain model can later be used as a foundation for other diagrams.

### 3.2.2 Productivity-Enhancing Application Domain

This subsection provides a detailed analysis of the domain for the productivity-enhancing application, thoroughly examining the various components within the domain. It describes the purpose of each entity and explores the relationships among them. By doing so, it offers insights into how these entities interact and contribute to the overall functionality of the application. The domain model is divided into three subdomains, as shown in Figure 3.9. Detailed overview of each subdomain follows.

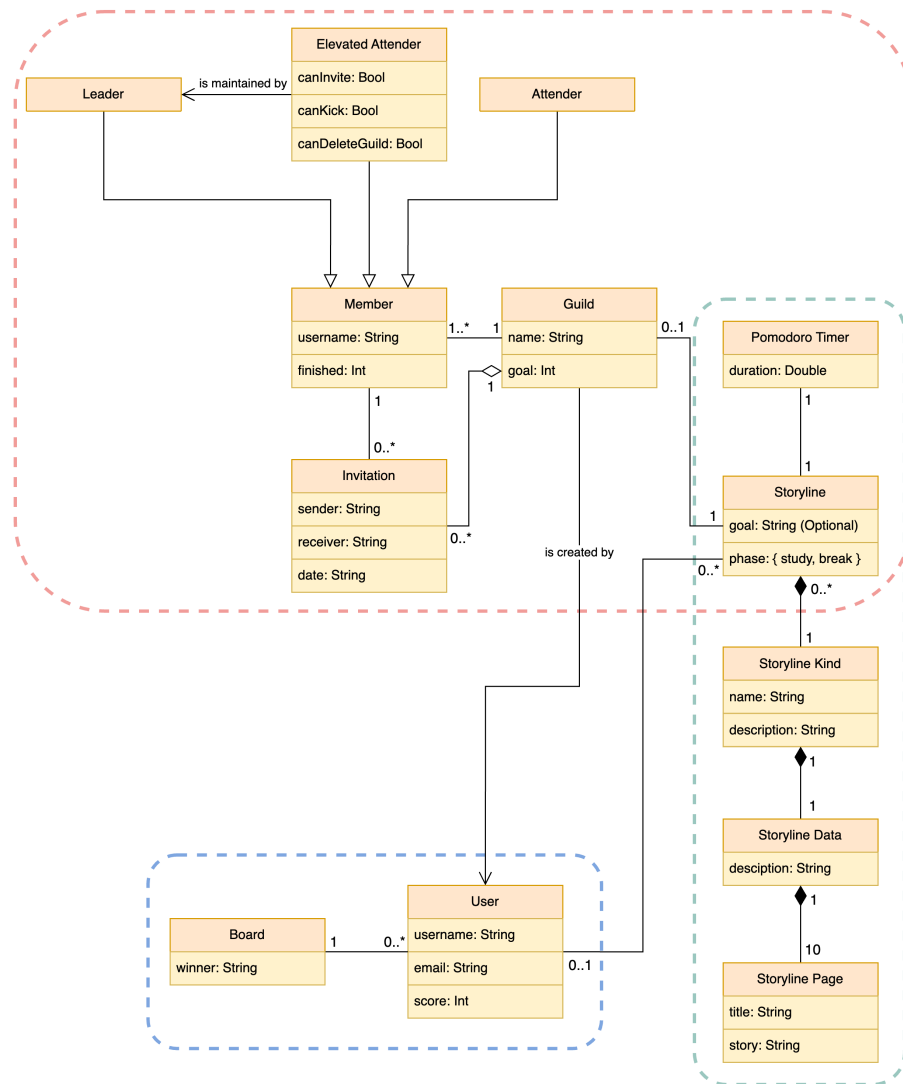


Figure 3.9: Productivity-Enhancing Application Domain

### 3.2.2.1 Storylines

The first subdomain, labeled as *Storylines* and outlined with a green dashed line in the diagram shown in Figure 3.10, integrates a Pomodoro timer with captivating storylines, thereby adding a gamification element to the traditional Pomodoro technique. To find out more about Pomodoro technique see Section 2.2.5. This integration aims to enhance productivity by making the process of time management more engaging and enjoyable for the user. A detailed description of each entity within this subdomain follows, providing further insight into their roles and interactions within the system.

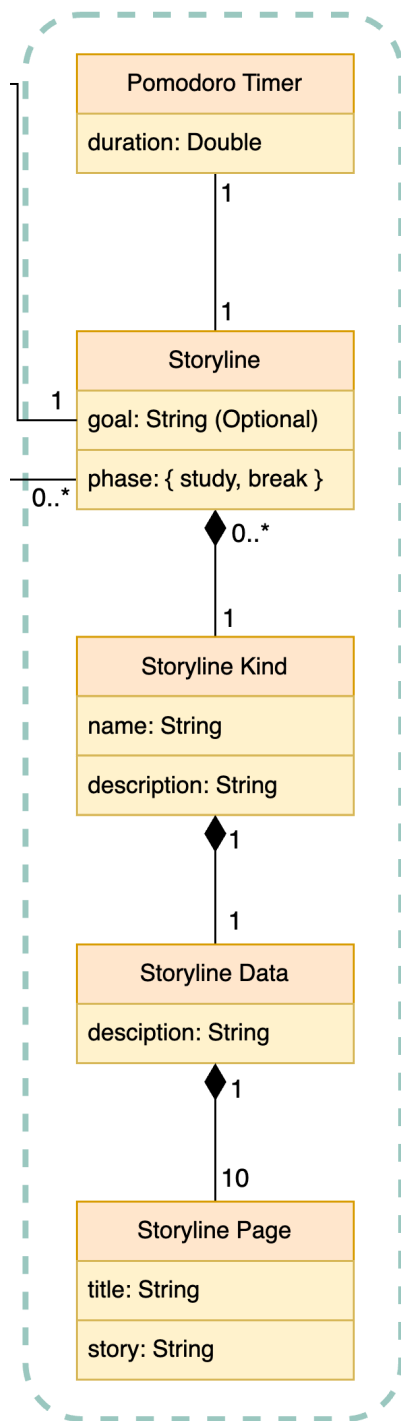


Figure 3.10: Storylines Subdomain

1. **Storyline:** The **Storyline** entity is the central entity of this sub-domain, representing a storyline as a whole. The **goal** property of the **Storyline** is the total score that the user sets as their target to achieve. The **Phase** enum property determines whether the storyline will appear during the **break** phase or the **study** phase of the Pomodoro technique. A storyline belongs either to the **User** entity or to the **Guild** entity, depending on context.

2. **Storyline Kind, Storyline Data, Storyline Page:** These three entities together constitute the content of a storyline. A **Storyline** entity cannot exist without those three entities.

The **Storyline Kind** entity defines the general category or topic of the storyline, identified by its **name** property. Each **Storyline** is associated with one **Storyline Kind**, but multiple **Storyline** entities can belong to the same kind.

The **Storyline Data** entity pertains to the specific content of the storyline, described by the **description** property, which provides a brief summary. There is always a one-to-one relationship between **Storyline Data** and **Storyline Kind** entities.

The **Storyline Page** represents an individual segment of the story, characterized by **title** and **story** properties. There are always ten pages, and each page belongs exclusively to one **Storyline Data** entity.

3. **Pomodoro Timer:** The **Pomodoro Timer** entity is a countdown timer used in the Pomodoro technique. The timer is configured with a **duration** setting. Each **Storyline** entity is associated with one timer, and each timer is linked exclusively to one storyline.

### 3.2.2.2 Guilds

*Guilds* represent another subdomain of the application. This subdomain facilitates the connection between users, allowing them to form guilds, which are groups of members who can either compete or cooperate to achieve a set goal. This functionality supports active and collaborative learning, which is further detailed in Chapter 2. The following text provides a detailed description of the Guilds subdomain, its entities, and their relationships, supported by Figure 3.11.

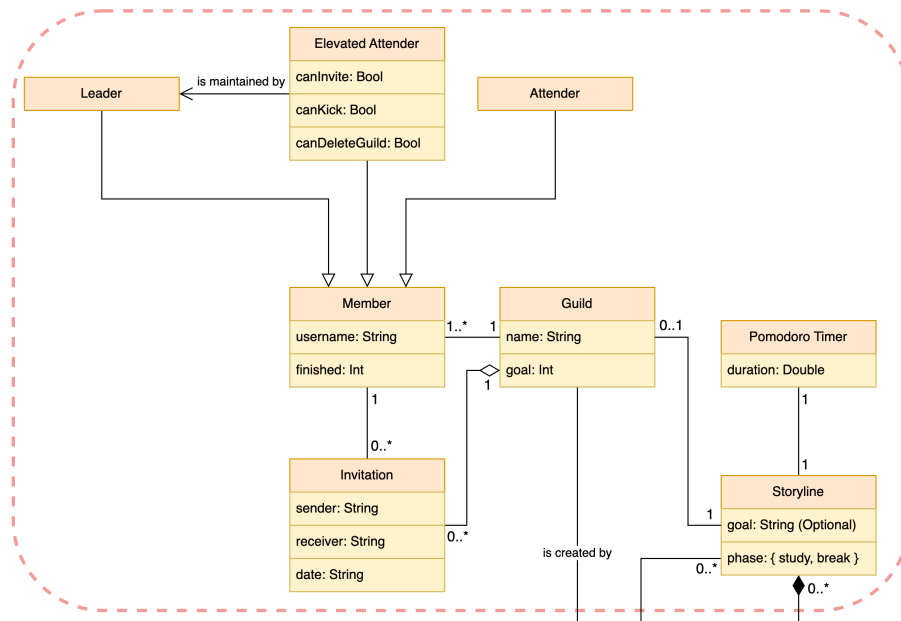


Figure 3.11: Guilds Subdomain

1. **Guild:** The **Guild** entity is the main entity of this subdomain. It is characterized by its **name** and **goal** properties. This entity is closely linked with the **Storyline** entity, which was discussed earlier in Section 3.2.2.1. Each guild is required to have a specific storyline associated with it. A guild also has its members (at least one).
2. **Member:** In the Storylines subdomain, the **Member** entity represents an individual member of a guild. Each member is associated with a specific guild and is described by the **username** property and the **finished** property. The **finished** property indicates how much of the guild's goal the member has completed.

There are different types of the **Member** entity based on their privileges. The **Leader** has full control over the guild. The **Elevated Attender** possesses rights determined by the guild leader. The **Attender** is a regular member with basic privileges.

3. **Invitation:** The `Invitation` entity is employed to invite new members to a guild. This entity possesses properties such as `sender`, `receiver`, and `date`. Any number of invitations can be linked to a guild. An invitation entity is meaningless without a guild association.

### 3.2.2.3 Board

The *Board* subdomain is a feature that enables users to compare their performances with each other. This comparative element is designed to foster a sense of competition and community among users, enhancing engagement and motivation within the application.

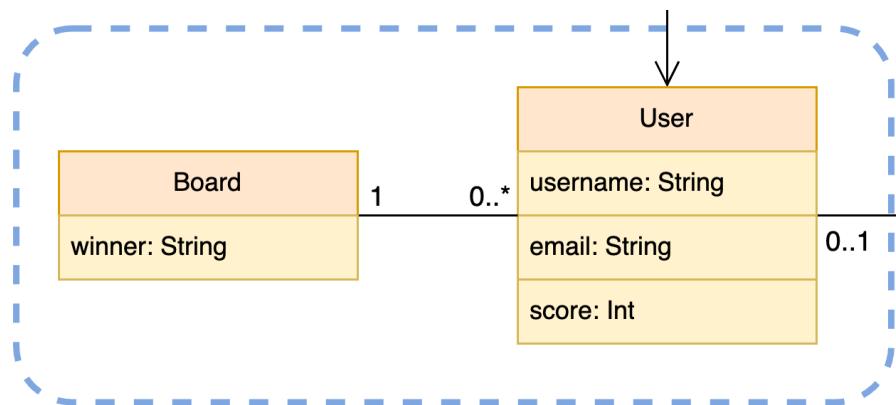


Figure 3.12: Board Subdomain

1. **User:** The `User` entity encapsulates information about the user, including properties such as `username` and `email`. Additionally, the `score` property represents all the points a user has gained by completing sessions using the Pomodoro timer during study periods. Each user is part of a board, which ranks all users based on their scores.
2. **Board:** The `Board` entity is a ranked list of users, ordered by the `score` property. Each board can include multiple users, but each user is associated with only one board.

### 3.2.3 Summary of Domain Model

In summary, this section explores the domain of a productivity-enhancing application, focusing on three subdomains: Storylines, Guilds, and Board. The Storylines subdomain enhances time management through gamification, Guilds enables collaborative and competitive user groups, and Board offers a competitive platform for performance comparison. Each subdomain is designed with specific entities to optimize user engagement and functionality within the application. This domain model will later serve as a foundation for the proper design of the application and for developing other diagrams and models.



### 3.3 Application Requirements

Before beginning the design of the application, it is essential to gather requirements to accurately define the behavior of the final product. This initial step not only helps in estimating the effort required for the project but also in setting clear boundaries for its scope. [89]

Gathering requirements involves distinguishing between functional and non-functional requirements, which are both critical for the development process. *Functional requirements* describe what the system should do, detailing the behaviors, functionalities, and interactions that the software must support. [89]

In contrast, *non-functional requirements* focus on how the system performs certain operations and qualities it must have, such as security, usability, reliability, and performance. These two categories of requirements together ensure that the application is both effective in fulfilling its intended purpose and efficient in its operation, thereby aligning with user expectations and technical specifications. [89]

#### 3.3.1 Frameworks for Requirement Definition

There are multiple frameworks available for defining and categorizing requirements. For the purposes of this thesis, the *FURPS*, *MoSCoW*, and *SMART* frameworks will be utilized. Each of these frameworks offers a unique approach to requirement analysis and prioritization, which will aid in structuring and clarifying the requirements effectively for the project. These models will help ensure that the requirements are comprehensive, well-defined, and aligned with the project's goals.

##### 3.3.1.1 SMART

The *SMART* criteria is a framework designed to set clear, well-defined, and most importantly achievable goals. It stands for Specific, *Measurable*, *Achievable*, *Relevant*, and *Time-bound*. Each component of the SMART framework helps in creating effective and actionable objectives. Goals are made specific to eliminate ambiguity, measurable to track progress, achievable to ensure they are realistic, relevant to align with broader business objectives, and time-bound to provide a deadline. This approach is particularly valuable in project management and personal development contexts, as it provides a structured and clear methodology for goal-setting, ensuring that all objectives are not only clear and practical but also aligned with the overarching priorities of the project or organization. [90]

##### 3.3.1.2 FURPS

The *FURPS* model is an acronym that stands for Functionality, Usability, Reliability, Performance, and Supportability - categories that encompass a comprehensive range of software qualities. Developed by Hewlett-Packard, *FURPS* is widely used in software development to help define and categorize system requirements. Beyond these primary categories, it also considers design constraints, implementation requirements, interface requirements, and physical requirements. By employing *FURPS*, developers and project managers can ensure a holistic approach to software design, addressing both functional require-

ments and user experience aspects, which are crucial for the successful deployment and operation of software systems. [91]

#### 3.3.1.3 MoSCoW

The *MoSCoW* method is a prioritization technique used in project management and requirement analysis to categorize the importance of various deliverables into four groups: Must have, Should have, Could have, and Won't have. This method helps project managers and teams to clearly distinguish between essential and optional features, ensuring that critical requirements are addressed first and resources are allocated efficiently. By focusing on these priorities, the MoSCoW method aids in creating a structured approach to managing tasks and expectations, streamlining the development process and facilitating clearer communication among stakeholders regarding project objectives and deliverables. [92]

#### 3.3.2 Requirements Specification

The specification of requirements for the productivity-enhancing application follows the frameworks described in the previous section. All requirements are specified in the format shown in Figure 3.13. To view all the requirements for the application, refer to Appendix A.

**Identifier:** Identifier of the requirement.

**Name:** Name of the requirement.

**Description:** Description of the requirement.

**FURPS:** Categorization according to the FURPS framework [91].

**MoSCoW:** Categorization according to the MoSCoW framework [92].

Figure 3.13: Requirement Specification Format

To see a detailed diagram of both the functional and non-functional requirements, please refer to Figure 3.14.

#### 3.3.3 Summary of Application Requirements

In summary, this section provides an overview of the requirements categorization frameworks used in this thesis, as well as the specification of the requirements for the productivity-enhancing application. These requirements were specified by employing commonly used processes in Software Engineering and will later be used to properly design the application.

### 3.4 Application Use Cases

Use cases depict specific scenarios that demonstrate how functional requirements are applied within the system. They provide a detailed view of the application's capabilities and the specific business processes it supports. It is crucial that these use cases comprehensively cover all functional requirements to ensure a complete understanding of the system's functionality and its operational context. [93]

FUNCTIONAL REQUIREMENTS	NON-FUNCTIONAL REQUIREMENTS
FR1: Account Creation	NF1: iOS 17.0+ Compatibility
FR2: Log In	NF2: iPadOS 17.0+ Compatibility
FR3: Log Out	NF3: User-Friendly Interface
FR4: Email Verification	NF4: Application Extensibility
FR1: Account Creation	NF5: WatchOS Compatibility
FR5: Reset Password	FR16: Create Guild
FR6: Change Password	FR17: Update Guild
FR7: Change Username	FR18: Delete Guild
FR8: Change Email	FR19: Invite Guild Member
FR9: Account Deletion	FR20: Remove Guild Member
FR10: English Localization	FR21: Board
FR11: Czech Localization	FR22: Board Sorting
FR12: Plain Timer	FR23: Notifications
FR13: Create Storyline	FR24: Calendar Progress Tracker And Organizer
FR14: Delete Storyline	FR25: Authentication Process Deeplinks
FR15: Update Storyline	FR26: In-App Purchases
FR27: Dynamic Island Support	

Figure 3.14: Requirements Specification Overview

### 3.4.1 Use Case Diagram Notation

This section explores *Use Case Diagram Notation*, detailing key components and their interactions within a system. It covers elements like *Use Cases*, *Associations*, *Actors*, and *System* boundaries, as well as relationships such as *Include*, *Extend*, and *Dependency*, all depicted through concise figures and descriptions. Each part adheres to Unified Modeling Language (UML) standards (ISO/IEC 19505-2:2012 standard), providing a clear framework for understanding and implementing system specifications effectively.

#### 3.4.1.1 Use Case

A *use case* outlines the sequence of actions a system performs to deliver a valuable, observable outcome for actors or stakeholders involved with the system. [88, 10]

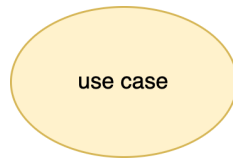


Figure 3.15: Use Case Notation; based on [10]

### 3.4.1.2 Association

An *association* in modeling represents a semantic relationship through tuples that link typed instances, known as links. Each link corresponds to association ends, defined by properties related to specific types. Navigation of an association depends on whether an end property is owned or navigable, determining if it can be accessed from the opposite ends. [88, 10]

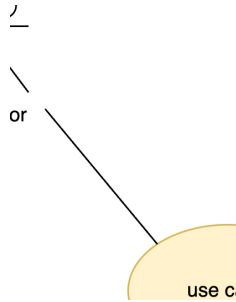


Figure 3.16: Association Notation; based on [10]

### 3.4.1.3 Actor

An *actor* in a system represents a role played by a user or another system that interacts with a subject, but is external to it. This role, which involves exchanging signals and data, does not necessarily correspond to a specific physical entity but rather to a facet of an entity relevant to the use cases. An actor can represent humans, external hardware, or other systems, and a single physical entity may assume multiple actor roles or vice versa. [88, 10]

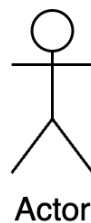


Figure 3.17: Actor Notation; based on [10]

#### 3.4.1.4 System

If a subject or *system* boundary is depicted, the use case ellipse is visually placed inside the system boundary rectangle. This placement indicates that the use case is relevant to that classifier, but not necessarily that it is owned by it. [88, 10]

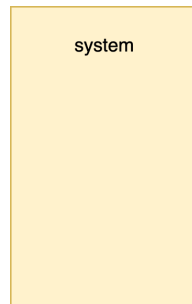


Figure 3.18: System Notation; based on [10]

#### 3.4.1.5 Include

An *include* relationship specifies that one use case incorporates the behavior defined in another use case. [88, 10]

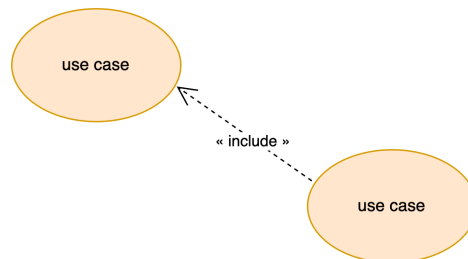


Figure 3.19: Include Notation; based on [10]

#### 3.4.1.6 Extend

An *extend* use case relationship specifies that the behavior of one use case can enhance another, typically at designated extension points in the extended use case. While the extended use case functions independently, the extending use case often describes supplementary behavior that adds value under certain conditions but may not be standalone. [88, 10]

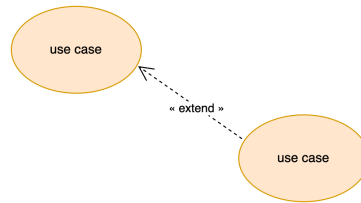


Figure 3.20: Extend Notation; based on [10]

#### 3.4.1.7 Dependency

A *dependency* is a relationship in modeling that indicates one or more model elements rely on other elements for their specification or implementation. This relationship shows that the full meaning or structure of the dependent elements is contingent on the elements they depend on. [88, 10]

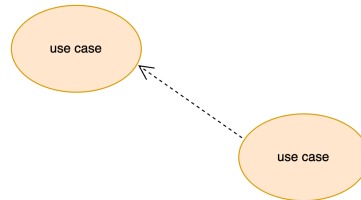


Figure 3.21: Dependency Notation; based on [10]

#### 3.4.1.8 Generalization

A *generalization* is a relationship where a specific classifier inherits features from a more general classifier, meaning each instance of the specific classifier is also considered an instance of the general classifier. [88, 10]

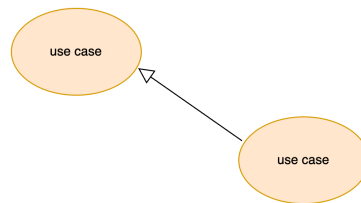


Figure 3.22: Generalization Notation; based on [10]

#### 3.4.1.9 Realization

*Realization* is an abstraction relationship where one set of model elements (the supplier) provides a specification, and another set (the client) implements it. This relationship facilitates modeling processes such as stepwise refinement, optimizations, transformations, and framework composition. [88, 10]

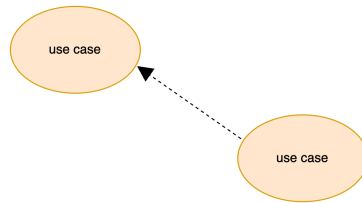


Figure 3.23: Realization Notation; based on [10]

#### 3.4.1.10 Collaboration

A *collaboration* describes a structure where various elements each perform specialized functions to collectively achieve a desired functionality. It aims to explain how a system operates, focusing only on relevant aspects and omitting details like the identities or specific classes of the participating instances. [88, 10]

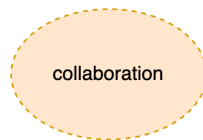


Figure 3.24: Collaboration Notation; based on [10]

### 3.4.2 Use Case Specification

This subsection details the specification of use cases for the productivity-enhancing mobile application. The use cases are specified according to standardized processes in Software Engineering. The textual format of the use cases is as shown in Figure 3.25.

**Identifier:** Identifier of the use case.

**Name:** Name of the use case.

**Description:** Description of the use case.

**Requirements Covered:** Identifiers of requirements that the use case covers.

Figure 3.25: Use Case Specification Format

Figure 3.26 illustrates all the use cases within the system using a *UML Use Case diagram*, conforming to the ISO/IEC 19505-2:2012 standard [88]. To see the full textual use case specification, refer to Appendix B.

### 3.4.3 Summary of Application Use Cases

This section outlines the use case scenarios for a productivity-enhancing mobile application, demonstrating how functional requirements are implemented within the system. By employing UML notations and various relationship types, it depicts the application's functionalities and interactions with users.

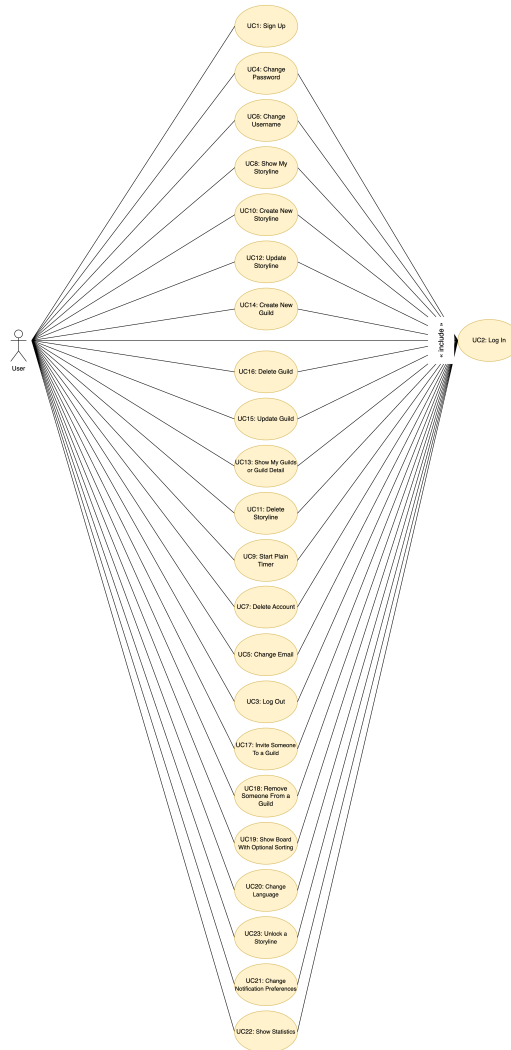


Figure 3.26: Use Case Diagram

This comprehensive coverage ensures a full understanding of the system’s capabilities. The use cases are specified following standard processes in Software Engineering.

### 3.5 Summary of Domain Analysis

This chapter has conducted a thorough domain analysis for a productivity-enhancing iOS application, adhering to standard Software Engineering processes. Together with Chapter 2, it establishes a strong foundation for the future design of a productivity-enhancing mobile application and fulfills one of the requirements specified in the thesis assignment.



---

## Technology Analysis

This chapter provides an overview of various technologies relevant to iOS development, offering insights into the tools and frameworks essential for building mobile applications. It discusses Firebase services, which will be utilized to design the backend of a productivity-enhancing application as specified in the thesis assignment. Additionally, the chapter mentions features of the *Swift* language, *iOS SDK* architecture, *SwiftUI*, *Xcode*, *Swift Package Manager*, and *App Store* tools, providing a comprehensive understanding of the technological landscape for iOS development.

### 4.1 Firebase Backend

*Firebase* services, developed by Google<sup>4</sup>, offer a comprehensive suite of tools for mobile and web application development. These services are categorized into three main groups: *Build*, *Release & Monitor*, and *Engage*. The *Build* services provide the backend infrastructure essential for app development, while the *Release & Monitor* services focus on the deployment of applications and their ongoing monitoring. The *Engage* services aim to enhance user engagement through features like analytics, A/B testing, and messaging campaigns. [94] A detailed overview of specific services Authentication, and Firestore Cloud which fall under the *Build* category, is provided in the following sections. These services were chosen because they collectively establish a solid foundation for building mobile applications.

#### 4.1.1 Firebase Authentication

*Firebase Authentication* offers a comprehensive backend service that facilitates secure user authentication across various platforms. It provides SDKs and pre-built UI libraries supporting numerous authentication methods, including email, phone numbers, and federated identity providers like Google and Facebook. Integrated with *Firebase* services and adhering to standards like OAuth 2.0, it ensures seamless backend integration. For enhanced security, *Firebase Authentication* with *Identity Platform* adds features such as multi-factor authentication and enterprise-level support, making it a versatile tool for managing user identities in both mobile and web applications. [95]

---

<sup>4</sup>The full company name is Google LLC.

### 4.1.2 Firestore Cloud

*Cloud Firestore* is a scalable *NoSQL* database from Firebase and Google Cloud, designed for mobile and web development. It ensures real-time data synchronization and robust offline support, allowing apps to remain responsive regardless of network connectivity. Built on Google Cloud's infrastructure, Firestore offers strong consistency, automatic data replication, and transaction support, making it highly scalable. It supports complex data structures through *documents* and *collections*, facilitating expressive querying and real-time updates. [96]

As mentioned, Firestore organizes data using documents and collections rather than tables. In this structure, documents are stored within collections, and these documents can hold references to other collections, thereby forming complex, hierarchical data structures. This allows for greater flexibility in data organization and retrieval. An example of such a data structure is illustrated in Figure 4.1.

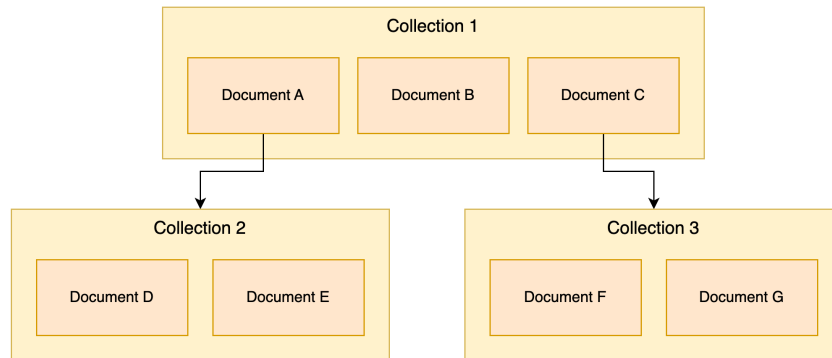


Figure 4.1: Cloud Firestore Collections and Documents Example

### 4.1.3 Other Firebase Services

Firebase offers a range of services beyond Authentication and Firestore, including *Firebase Cloud Messaging* for notifications, *Firebase Hosting* for web hosting, *Realtime Database* for live data syncing, *Firebase Functions* for serverless backend code, *Google Analytics* for comprehensive app analytics and more. [94]

## 4.2 Swift Language

*Swift*, currently in its 5.10 version, is a powerful programming language suitable for a wide range of applications, from mobile devices to server environments. It skillfully blends contemporary language features with the insights of an open-source community, making it both safe and fast—qualities that attract both novice and seasoned developers. Swift proactively prevents common programming errors like uninitialized variables, out-of-bounds array indices, and integer

overflows, thus boosting both its safety and reliability. The language is in constant evolution, regularly incorporating new features that enhance its utility and ease of use. [97]

The language's syntax is intuitive and optimized for performance, supported by a comprehensive standard library that adheres to modern programming standards. This simplicity in coding, combined with functional depth, ensures that Swift's straightforward code also results in optimal performance without sacrificing readability. This balance underscores Swift's dedication to delivering performance without compromise, making it a forward-looking choice for developers interested in building enduring and influential software applications. [98]

Expanding beyond Apple's ecosystem, Swift is a versatile, multi-platform language that also supports Linux and is making strides toward compatibility with Windows. It integrates with Objective-C, allowing for mixed-language projects that can modernize legacy code or incorporate existing Objective-C libraries. Swift Playgrounds, an innovative educational tool, democratizes language learning, making Swift accessible and engaging. The language also emphasizes protocol-oriented programming, enhancing code reusability and flexibility. Optimized by the LLVM compiler for both compile-time and runtime performance, Swift ensures efficient execution on modern hardware. Furthermore, its open-source model encourages community contributions through *Swift.org*<sup>5</sup> and *GitHub*<sup>6</sup>, fostering a collaborative environment that propels Swift forward in the competitive world of programming languages. [98]

### 4.2.1 Key Features

- **Simplified Syntax:** Swift allows entire programs to be written with less syntactic complexity, eliminating the need for a separate library to handle common tasks like outputting text or managing strings.
- **Type Inference:** Swift provides powerful type inference capabilities where the compiler can deduce the type of variable or constant without explicit type annotations.
- **String Interpolation:** Swift supports advanced string interpolation which enables embedding values directly within strings using a simple and readable syntax.
- **Memory Management:** Swift uses automatic reference counting (ARC) for memory management without the need for programmers to explicitly free up memory. This helps in managing the lifecycle of objects through strong and weak references.
- **Control Structures:** Swift supports a range of control flow structures including `for-in`, `while`, and `repeat-while` loops, alongside `if` and `switch` statements which can use pattern matching to execute more complex checks.
- **Functions and Closures:** Functions in Swift are first-class types. Swift supports closures, which are blocks of functionality that can be passed around and used in the code.

---

<sup>5</sup>Swift.org is the official resource and community hub for the Swift programming language.

<sup>6</sup>GitHub is a platform for hosting and collaborating on software projects.

- **Optionals and Error Handling:** Swift introduces optionals to handle the absence of a value, and robust error handling using `try`, `catch`, `throw`, and `throws` keywords, improving the safety and robustness of the code.
- **Protocols and Extensions:** Swift allows for defining protocols which are interfaces that define a blueprint of methods, and extensions which help in adding additional functionality to existing classes, structures, or types.
- **Generics:** Swift supports generic programming, enabling users to write flexible, reusable functions and types that can work with any type, subject to constraints defined by the developer.
- **Concurrency:** Swift provides modern features for handling concurrency including `async` and `await` syntax, which simplify writing asynchronous code. [97]

### 4.2.2 Automatic Reference Counting

Swift employs *Automatic Reference Counting (ARC)* to manage memory efficiently, thus enhancing both application safety and performance. ARC automatically tracks and manages the life cycle of objects in memory, deallocating them when no longer needed. This mechanism simplifies memory management by eliminating manual processes, reducing the likelihood of errors such as memory leaks. Despite its advantages, ARC can encounter issues with reference cycles, which are resolved using weak and unowned references to prevent memory leaks, thereby ensuring robust memory management across applications. [99, 100]

#### 4.2.2.1 Weak References

In Swift, *weak references* are used to prevent strong reference cycles that could lead to memory leaks, as described in the language's memory management strategy. A weak reference does not hold a strong claim on the object it references, allowing ARC to deallocate the referenced object when there are no more strong references to it. This characteristic of weak references ensures that they do not prevent the garbage collection of objects that are no longer in use, thereby enhancing memory management. Further technical detail reveals that when a property is declared as `weak`, ARC automatically sets the reference to `nil` when the instance it points to is deallocated. This automatic nil-setting feature prevents the reference from becoming a dangling pointer, thus avoiding potential runtime crashes associated with accessing deallocated memory. This aspect of weak references is crucial for the stability and reliability of applications, especially those with complex data models involving multiple relationships among objects. [11, 100, 101]

```

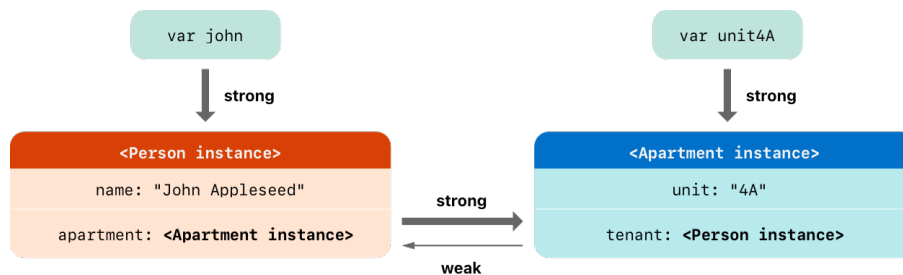
class Person {
    let name: String
    init(name: String) { self.name = name }
    var apartment: Apartment?
}

class Apartment {
    let unit: String
    init(unit: String) { self.unit = unit }
    weak var tenant: Person?
}

```

Figure 4.2: Swift Weak Referencing Example [11]

Weak references are particularly useful in relationships where one object does not own the other, as demonstrated in the example Figure 4.2 involving a `Person` and an `Apartment`. In this scenario, while a `Person` might live in an `Apartment`, the `Apartment` does not own the `Person`. By declaring the `tenant` property of the `Apartment` as a weak reference, Swift sets this reference to `nil` automatically when the `Person` instance is deallocated. This automatic nullification helps avoid dangling pointers, which could lead to crashes or unexpected behaviors if accessed. [101] If each of the two classes are initialized and assigned to each other, the relationship between them will appear as shown in Figure 4.3.

Figure 4.3: Relation Between `Person` and `Apartment` Objects [11]

Furthermore, weak references must be declared as optional variables because their value can become `nil` when the instance they refer to is deallocated. This property of weak references also means that checks for their existence must be handled just like any other optional in Swift, ensuring runtime safety and robustness in the application's memory management practices. Notably, property observers do not fire when ARC sets a weak reference to `nil`, indicating the non-intrusive nature of ARC's memory management. This behavior exemplifies how Swift's memory management is designed to be both efficient and safe, reducing the cognitive load on developers and allowing them to focus

more on application logic rather than memory management intricacies. [101]

The deallocation can be demonstrated by discarding the reference to the object of `Person` class<sup>7</sup>. If this reference is released, the object is deallocated because it only has a weak reference to the `Apartment` instance; there is no other instance holding a strong reference to the `Person` object. [101] In this example, the objects in memory would appear as shown in Figure 4.4.

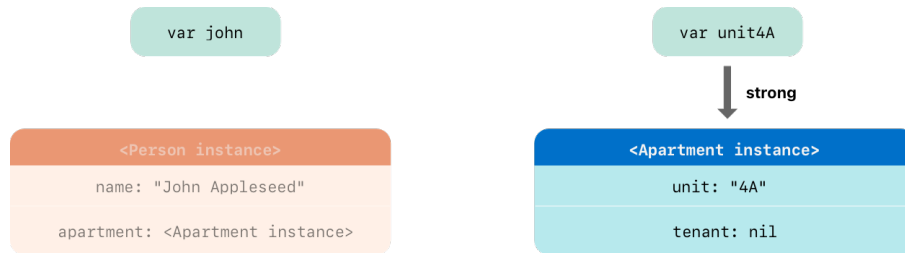


Figure 4.4: Discarding Reference to `Person` Object [11]

Finally, when reference to `Apartment` object is removed, there would be no objects left allocated in the memory [101], as depicted in Figure 4.5.

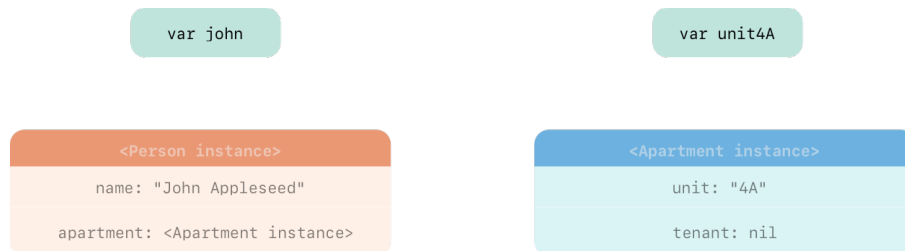


Figure 4.5: Discarding Reference to `Apartment` Object [11]

#### 4.2.2.2 Unowned References

*Unowned references* are similar to weak references in that they do not create a strong hold on the object they refer to. However, they differ in a crucial aspect: unowned references are used when the other instance they reference has the same or longer lifetime. Therefore, unlike weak references, an unowned reference is always expected to hold a value and is not made optional by Automatic Reference Counting, which does not set it to `nil`. [12]

Unowned references should only be used when there is certainty that the referenced instance will not be deallocated while the reference is still in use. Accessing an unowned reference after the object it points to has been deallocated will lead to a runtime error. This emphasizes the importance of understanding the lifecycle of objects within the application to avoid unwanted crashes. [12]

<sup>7</sup>Note that to fully understand the context, the `Person` object is strongly referenced by the `john` variable, and the `Apartment` object is strongly referenced by the `unit4A` variable.

```

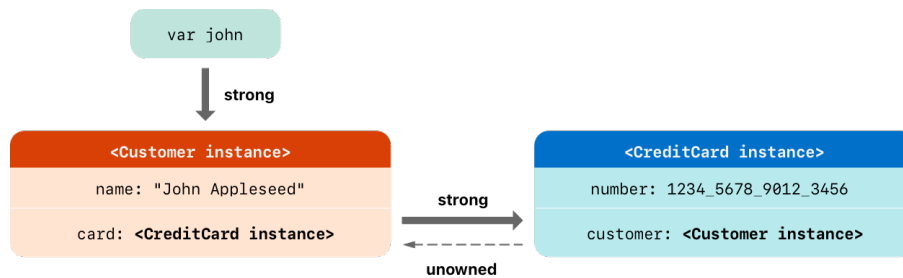
class Customer {
    let name: String
    var card: CreditCard?
    init(name: String) {
        self.name = name
    }
}

class CreditCard {
    let number: UInt64
    unowned let customer: Customer
    init(number: UInt64, customer: Customer) {
        this.number = number
        this.customer = customer
    }
}

```

Figure 4.6: Swift Unowned Referencing Example [12]

The usage of unowned references is demonstrated with two classes, `Customer` and `CreditCard`, which model a relationship where a credit card always references a customer, but not necessarily vice versa. This setup helps avoid strong reference cycles, with the `CreditCard` class having an `unowned` reference to a `Customer`. A `CreditCard` instance is always linked to a `Customer` instance at initialization, ensuring that the reference is valid as long as the credit card exists. [101] This example is shown in Figure 4.6, followed by depiction of memory of such case in Figure 4.7.

Figure 4.7: Relation Between `Customer` and `CreditCard` Objects [12]

In the provided scenario, as it has already been mentioned, the `Customer` instance forms a strong reference to the `CreditCard` instance, while the instance of `CreditCard` maintains an `unowned` reference to the `Customer` instance. This arrangement ensures that when the `john` variable<sup>8</sup>, is set to `nil`, it effectively breaks the strong reference chain (as shown in Figure 4.8), lead-

<sup>8</sup>Note that the `john` variable holds a strong reference to the `Customer` instance

ing to the deallocation of the `Customer` instance due to the absence of any other strong references. Subsequently, the `CreditCard` instance is also deallocated. [101]

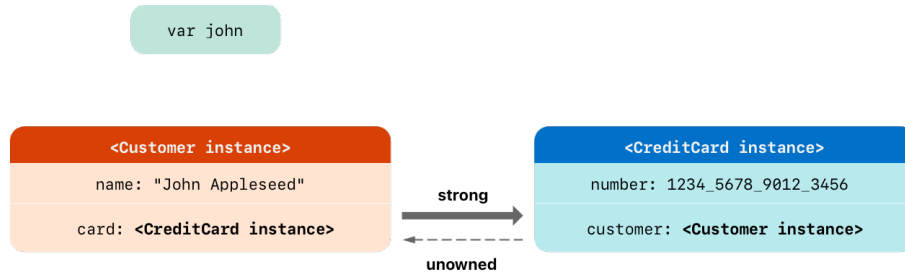


Figure 4.8: Relation Between `Customer` and `CreditCard` Objects [12]

### 4.3 iOS SDK

The *iOS Software Development Kit (SDK)* is fundamental to Apple’s mobile ecosystem, providing a structured framework for developing applications on devices like iPhones, iPads, and iPods. Designed with a layered architecture, the SDK compartmentalizes functionalities to simplify development and enhance security. From the foundational Core OS layer, which manages direct hardware interaction, to the Cocoa Touch layer at the top that enables intuitive user interfaces, each layer is equipped with specific frameworks targeting various app functionalities such as multimedia handling and user interface design. This hierarchical structure not only streamlines the development process but also strengthens app security by reducing exposure of higher-level APIs. [13] A detailed description of the iOS SDK layers follows after Figure 4.9, which depicts the overall architecture.

#### 1. Core OS Layer

- Directly interfaces with device hardware.
- Manages low-level operations such as memory management, file system, and basic networking.
- Houses specific frameworks like *Core Bluetooth* for device connectivity and *Security Services* for app security.

#### 2. Core Services Layer

- Provides high-level APIs abstracted from the *Core OS* services.
- Includes frameworks such as *Core Location* for location services and *Core Data* for data management.
- Facilitates easier access to essential services needed by all apps.

#### 3. Media Layer

- Supports multimedia functionalities, allowing manipulation of audio, video, and graphics.



- Contains frameworks such as *AV Foundation* for audiovisual content and *Core Graphics* for drawing and animations.
- Critical for developing apps with rich multimedia content.

#### 4. Cocoa Touch Layer

- The highest abstraction layer, focused on the user interface and interaction.
- Supports touch input, notifications, and other high-level system services.
- Includes frameworks like *UIKit* for graphical and event-driven apps and *Map Kit* for integrating maps. [13]

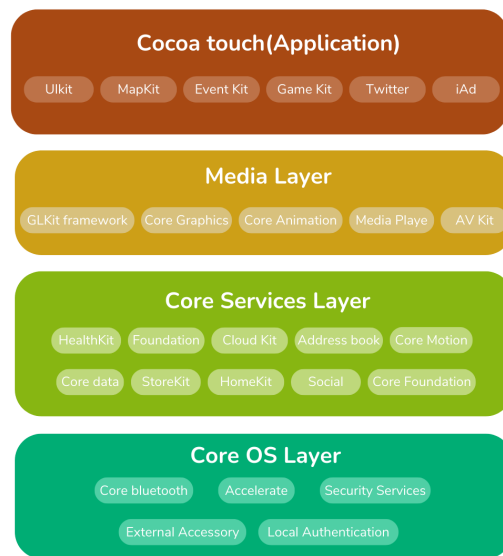


Figure 4.9: iOS SDK Architecture [13]

## 4.4 SwiftUI

*SwiftUI*, built on the Swift programming language, offers a framework for developing visually appealing applications across Apple’s diverse device ecosystem with minimal code. This framework unifies a variety of tools and APIs to ensure consistent user experiences across different devices, such as the MacBook Pro, iPad, and iPhone. SwiftUI introduced sophisticated animation tools that facilitate the creation of complex and smooth transitions. Additionally, data management is streamlined through the use of `@Observable`, which enhances UI responsiveness by updating views only when necessary. Furthermore, SwiftUI has expanded its capabilities to support the creation of spatial applications and interactive widgets, which can be customized for various device contexts. This includes the integration of 3D objects and immersive experiences leveraging *RealityKit*. [102]

In its approach to user interface design, SwiftUI adopts a declarative syntax, simplifying the specification of UI components such as lists and text fields. This syntax shift aids developers in focusing on the end goals of UI, rather than the detailed steps to achieve these goals, which streamlines the development process and enhances maintainability. SwiftUI is also designed to integrate seamlessly with existing Apple frameworks like UIKit and *AppKit*, facilitating its incremental adoption in legacy applications. Complementary to this, Xcode's design tools enhance SwiftUI's utility by providing instantaneous previews and updates, reflecting code changes in real time. This feature establishes a dynamic development environment, where developers can immediately observe the impact of modifications in various configurations, promoting a more effective and iterative process. [102]

```
import SwiftUI

struct ProfileView: View {
    @State private var isFollowing = false

    var body: some View {
        VStack(spacing: 10) {
            Image("profilePicture")
                .resizable()
                .aspectRatio(contentMode: .fill)
                .frame(width: 100, height: 100)
                .clipShape(Circle())
                .shadow(radius: 10)
            Text("John Doe")
                .font(.title)
                .fontWeight(.medium)
            Button {
                isFollowing.toggle()
            } label: {
                Text(isFollowing ? "Unfollow" : "Follow")
                    .foregroundColor(.white)
                    .padding()
                    .background(
                        isFollowing ? Color.red : Color.blue
                    )
                    .cornerRadius(10)
            }
        }
        .padding()
    }
}
```

Figure 4.10: SwiftUI Declarative Code Example

### 4.4.1 How It Works

This section offers an overview of three fundamental mechanisms used in SwiftUI. By understanding these mechanisms, developers can optimize their SwiftUI applications more effectively. They can ensure that views maintain stable identities to prevent unnecessary redraws, manage lifetimes to keep the user interface responsive, and handle dependencies to accurately update the interface based on changes in underlying data. This detailed exploration of SwiftUI's functional principles enables developers to fully leverage its capabilities for creating efficient, dynamic, and responsive applications. [103]

#### 4.4.1.1 View Identity

SwiftUI treats views as value types, unlike UIKit where views are reference types with unique pointers. This difference is crucial for understanding how SwiftUI manages view updates. SwiftUI needs to determine whether a view at a particular moment is the same as another view at a different moment or if they are completely distinct. For instance, consider a scenario where a SwiftUI view displays a user profile. If the user's name in the profile is updated, SwiftUI must decide whether this is still the same view with a modified state or a different view altogether. To make this distinction, SwiftUI uses two types of identities:

- **Explicit Identity:** It is possible to assign explicit identifiers to views using the `.id()` modifier, binding the view's identity to a hashable value. This is particularly useful for dynamic content, such as a list where each entry must be distinguishable from the others for proper updates and animations.
- **Structural Identity:** By default, SwiftUI uses the view's type and its position in the view hierarchy to determine its identity. This automatic mechanism ensures that views are recognized and updated efficiently based on their structural context within the UI. [103]

#### 4.4.1.2 View Lifetime

The *lifetime* of a SwiftUI view is tied to its identity. As long as the identity remains consistent, SwiftUI maintains the view's state across updates. For example, when toggling between different tabs in a view, SwiftUI does not recreate the entire view hierarchy; instead, it reuses the existing views as much as possible, updating only those parts that have changed based on their identities. This efficient management of view lifetimes significantly improves performance by avoiding unnecessary rendering of unchanged content. [103]

#### 4.4.1.3 View Dependencies

*Dependencies* in SwiftUI are the properties and data a view relies on to render its UI. SwiftUI tracks these dependencies to determine when a view needs to be updated. If a dependency changes, SwiftUI re-evaluates the view's body to reflect the change. For example, consider a view displaying a toggle switch bound to a `@State` variable. When the toggle is flipped, the change

## 4. TECHNOLOGY ANALYSIS

in the `@State` variable triggers SwiftUI to recompute the view's body, updating the UI to reflect the new state. [103]

### 4.5 Xcode

Xcode (current latest version is Xcode 15), Apple's *Integrated Development Environment (IDE)*, is a comprehensive tool designed to support the development, testing, and distribution of applications across all Apple<sup>9</sup> platforms. It provides a suite of enhancements that facilitate faster and more efficient app development. These improvements include advanced code completion that references all assets, leading to more robust and error-free coding. Xcode 15 also features interactive previews and live animations, allowing developers to see immediate impacts of their changes, which is especially beneficial in the context of dynamic content creation.

The IDE has been optimized for Apple's multicore silicon architecture, resulting in quicker project builds through a new linker and improved compiler performance. Additionally, Xcode integrates tools such as Git for version control directly within the development environment, enabling developers to manage code changes more effectively without needing to switch contexts. Overall, Xcode is the default IDE for Swift development. There are no other currently supported IDE options for Swift development. [104]

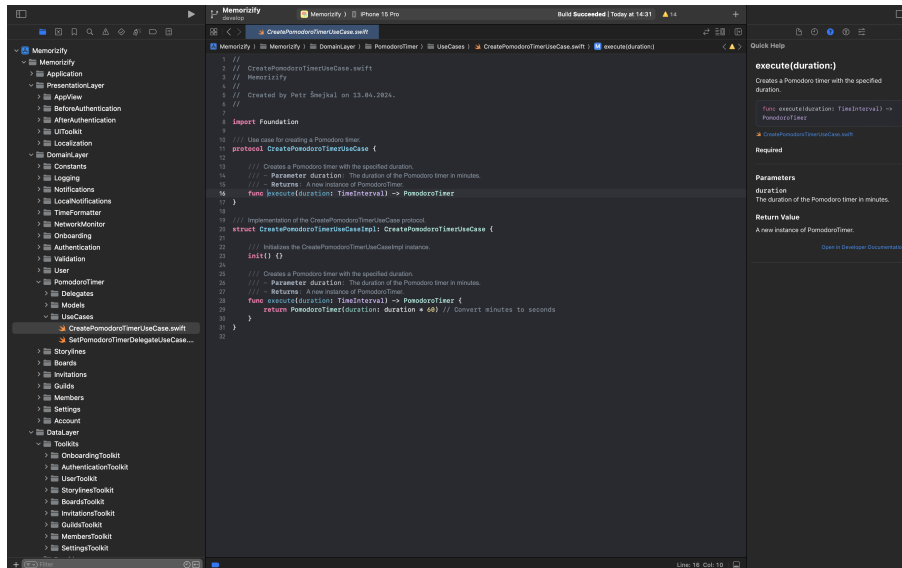


Figure 4.11: Xcode 15.3 IDE

<sup>9</sup>Full company name is Apple Inc.

## 4.6 Swift Package Manager

The *Swift Package Manager (SwiftPM)* serves as a comprehensive tool within the Swift ecosystem, streamlining the management and distribution of Swift code. Integrated with Swift's build system from version 3.0 onwards, SwiftPM automates the downloading, compiling, and linking of dependencies necessary for project development. The essence of SwiftPM lies in its ability to manage complex dependencies efficiently through a defined package system, which includes Swift source files and a manifest file known as `Package.swift`. This manifest dictates the structure of a package, specifying its name, contents, and dependencies, thereby organizing code into reusable modules that can enhance productivity and reduce redundancy across different applications. [105]

Each package can define one or more targets, specifying the products they generate, such as libraries or executables, and listing any other modules they depend on. SwiftPM simplifies the development workflow by automatically handling the dependency graph - a recursive breakdown of each package's requirements. This process ensures that all necessary components, including transitive dependencies, are correctly assembled without manual oversight. The package manager's capability extends to resolving versions and maintaining compatibility across various components, highlighting its role in facilitating a more manageable and robust development environment. This approach not only accelerates the development process by removing manual setup tasks but also enhances project maintainability and scalability. [105]

## 4.7 Testflight and AppStore

This section explores two essential Apple tools that support application development and distribution: *TestFlight* and the *App Store*. TestFlight enables developers to beta test their applications with extensive user feedback before official release, while the App Store provides a comprehensive platform for launching and managing applications globally. Both those tools can be maintained through *AppStore Connect* service. These tools are integral in supporting developers from the initial testing phase through to widespread distribution, optimizing both app functionality and market reach.

### 4.7.1 Testflight

*TestFlight* is an essential tool for developers aiming to beta test their applications with up to 10,000 users before releasing them on the App Store. By uploading a beta version of an application to App Store Connect, it is possible for testers to download the application via the TestFlight app, which facilitates the collection of vital feedback. The platform supports various Apple operating systems including iOS, iPadOS, macOS, and others. Features such as automatic updates ensure that testers always have access to the latest builds, and the capability to manage up to 100 apps simultaneously allows for comprehensive testing scenarios. Additionally, detailed tester metrics are available to enhance engagement and feedback analysis. With facilities for both internal team testing and external public testing, developers can customize the testing environment to optimize application functionality and refinement before its final release on the App Store. [106]

### 4.7.2 AppStore

Apple's *App Store* has become an essential platform for developers aiming to distribute and grow their applications on a global scale. It supports a variety of business models and provides comprehensive tools for app management and marketing, enabling developers to reach a vast audience across Apple devices like iPhones, iPads, and Macs. The App Store offers developers seamless integration with Apple's payment systems and access to a user base that spans over 175 regions and multiple languages. Additionally, it provides promotional opportunities and exclusive analytics to help developers maximize their app's potential. [107]

### 4.8 Summary of Technology Analysis

In summary, this chapter has provided a detailed look at the technologies relevant to iOS development. The findings from this analysis will be essential in shaping the design of the productivity-enhancing mobile app and its infrastructure. By following established Software Engineering practices, this analysis meets a key objective of the thesis, laying a strong groundwork for the next phases of development.

---

# Design

This chapter begins with the selection of architecture and design patterns to be utilized in the productivity-enhancing app, along with an examination of their advantages and trade-offs. Following this, the chapter proceeds to the process of designing a database schema using an entity-relationship diagram. Finally, the chapter delves into the design of the user interface, employing wireframes to visualize interface ideas and concepts, alongside other miscellaneous user interface considerations.

## 5.1 Choosing Architecture and Design Patterns

This section examines the decision-making process for selecting an appropriate *architectural pattern* for the productivity-enhancing mobile app. With several options available in iOS development, it is crucial to choose an architecture that aligns with the project's needs, and long-term maintainability.

Among the considered architectures, *VIPER (View, Interactor, Presenter, Entity, and Router)* stands out for applications where modularity, testability, and clear separation of concerns are crucial. Its component-based design makes it highly suitable for larger teams where responsibilities need to be clearly divided. VIPER's structured approach also simplifies debugging and testing, which are essential for maintaining code quality in complex applications. However, the complexity of VIPER can introduce a steep learning curve and may lead to over-engineering in smaller projects, which can deter its use in more straightforward applications. [108]

On the other hand, *Clean Architecture* is chosen for its flexibility and scalability, which are beneficial for applications expected to evolve over time. This architecture supports easy adaptation to changing requirements and technologies without significant disruptions. Clean Architecture's emphasis on separation of concerns at a higher level than VIPER makes it ideal for projects that anticipate significant business logic changes, ensuring that the app remains robust, testable, and easy to update. Despite these advantages, Clean Architecture can result in increased initial setup time and complexity, potentially slowing down early development stages as developers adapt to its demanding structural requirements. [109]

Another option would be to implement a basic structure for the iOS project by employing a 2-tier architecture, which basically separates the *View* and *View-*

*Model* into one tier, with the remaining components in another tier. This approach is relatively inflexible and is only suitable for very small simple apps [110]. Therefore, it will not be used for the productivity-enhancing application due to its limited scalability and adaptability.

Clean Architecture was chosen for the project due to its flexibility and adaptability, essential for a productivity-enhancing application in its early development phase. This architecture supports easy modifications and integrates user feedback without disrupting the overall structure, making it ideal for an app exploring new concepts. Its ability to evolve as the app transitions from a prototype to a refined product ensures that the project remains manageable and scalable, positioning it well for future expansions.

This architecture will be used in conjunction with the *MVVM (Model-View-ViewModel)* pattern for the presentation layer of the application. This pattern was chosen for its excellent compatibility with SwiftUI, particularly through the use of the `@Observable` property. More details about these two concepts will be discussed in the following sections.

### 5.1.1 Clean Architecture

*Clean Architecture*, much like various other architectural frameworks, is designed to segregate responsibilities effectively. This objective is accomplished by structuring the application into distinct layers, each with a specific role and responsibility. The separation of these layers is governed by the *Dependency Rule*, which ensures that dependencies flow inwards and that the outer layers can interact with the inner layers only through defined interfaces. [111]

The architecture itself consists of four primary layers: *Entities*, *Use Cases*, *Interface Adapters*, and *Frameworks and Drivers*. Each layer serves a unique function, as depicted in Figure 5.1 and further described.

- **Entities:** These are the core business objects of the application.
- **Use Cases:** This layer contains application-specific business rules.
- **Interface Adapters:** This layer converts data between the format most convenient for the use cases and entities, and the format most convenient for some external agency such as the Database or the Web.
- **Frameworks and Drivers:** This outermost layer generally consists of frameworks and tools such as the Database and the Web Framework. [111]

As moving toward the center from the Frameworks and Drivers to Entities, the level of abstraction increases, centralizing and protecting the application's core logic. This arrangement allows changes to external frameworks without significant impact on the core business logic. Such a design ensures a clean separation of concerns, making the system more maintainable and adaptable to change over time. [111]



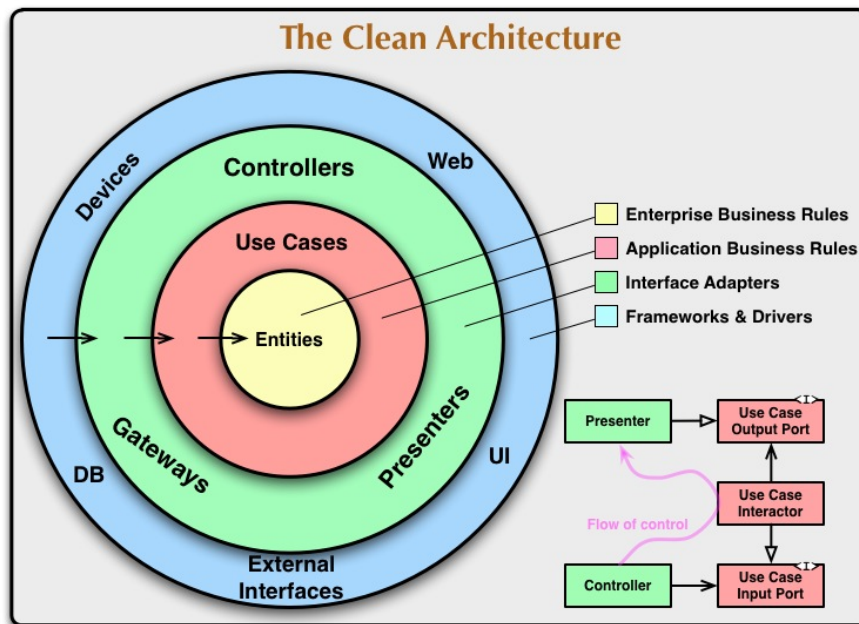


Figure 5.1: Clean Architecture as Described by Robert C. Martin [14]

### 5.1.2 MVVM

The *Model-View-ViewModel (MVVM)* pattern is highly regarded in iOS app development for promoting clean, maintainable, and testable code. It effectively separates the user interface (View) from the application's underlying data (Model) by introducing an intermediary layer known as the ViewModel, which manages presentation logic. This pattern ensures that the user interface layer, where user interactions occur, remains distinctly separate from the data and business logic layers. The ViewModel acts as a conduit between the View and the Model, managing data presentation and handling user interactions to facilitate updates and interactions within the app. [15, 112]

MVVM's structured approach provides numerous benefits, such as enhancing testability by allowing the ViewModel, which contains most of the application logic, to be tested independently from the user interface. This separation simplifies the codebase, making it more organized and easier to maintain. However, the pattern also introduces some complexities, such as the need for additional boilerplate code to bind the View to the ViewModel and potential increases in memory usage due to separate instances of the ViewModel and Model. Despite these challenges, MVVM remains an ideal choice for complex applications due to its ability to facilitate clean modifications and accommodate changing requirements without significant disruptions, making it highly scalable and flexible for evolving project needs [112], which is also required by the productivity-enhancing app.

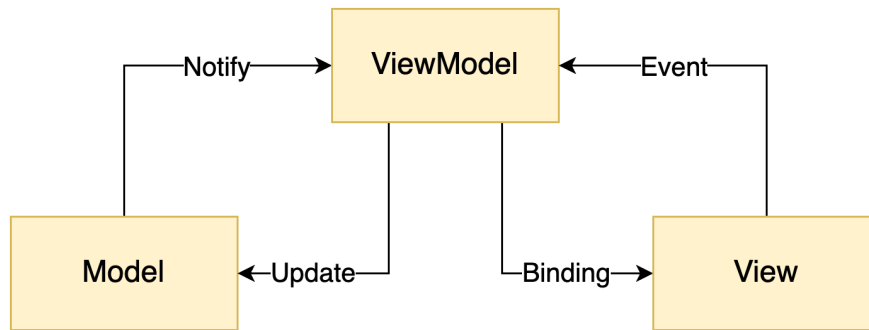


Figure 5.2: MVVM Pattern; based on [15]

### 5.1.3 Architecture in Practice

This section details the application of the previously discussed Clean Architecture and MVVM pattern within the productivity-enhancing app, demonstrating their effectiveness through a *Feature* module that incorporates both frameworks. This methodology will be uniformly applied across all modules of the application. The architecture of the Feature module is depicted in the Figure 5.3, followed by detailed description of all components.

#### 5.1.3.1 Domain Layer

The *domain layer* is the core component of the design, consisting of models and use cases. *Models*, alongside Errors—which represent error states of the module—define the entities within the module, outlining data structures critical for the application’s functionality. *Use cases* further detail the business logic associated with these models, specifying how data is processed and manipulated in response to user interactions or system events. As shown in Figure 5.3, the use cases are reliant on the models, indicating a structured dependency that ensures business rules are correctly implemented and maintained, thereby enhancing the application’s robustness and maintainability.

#### 5.1.3.2 Data Layer

This layer comprises *Toolkits* and *Providers*, with Providers specifically designed to offer straightforward services or simple data as comprehensive toolkits. For the productivity-enhancing application, notable examples include the *Keychain*<sup>10</sup> and *User Defaults*<sup>11</sup> from the iOS SDK, which are utilized for storing small amounts of data. For instance, the *Keychain Provider* offers a simple interface with CRUD operations for this secure storage.

Toolkits leverage these Providers to deliver comprehensive service packages, including repository implementations associated with the module. These

<sup>10</sup>Keychain is a secure storage for saving passwords and other sensitive information.

<sup>11</sup>User Defaults is used for storing lightweight data such as user settings and preferences.

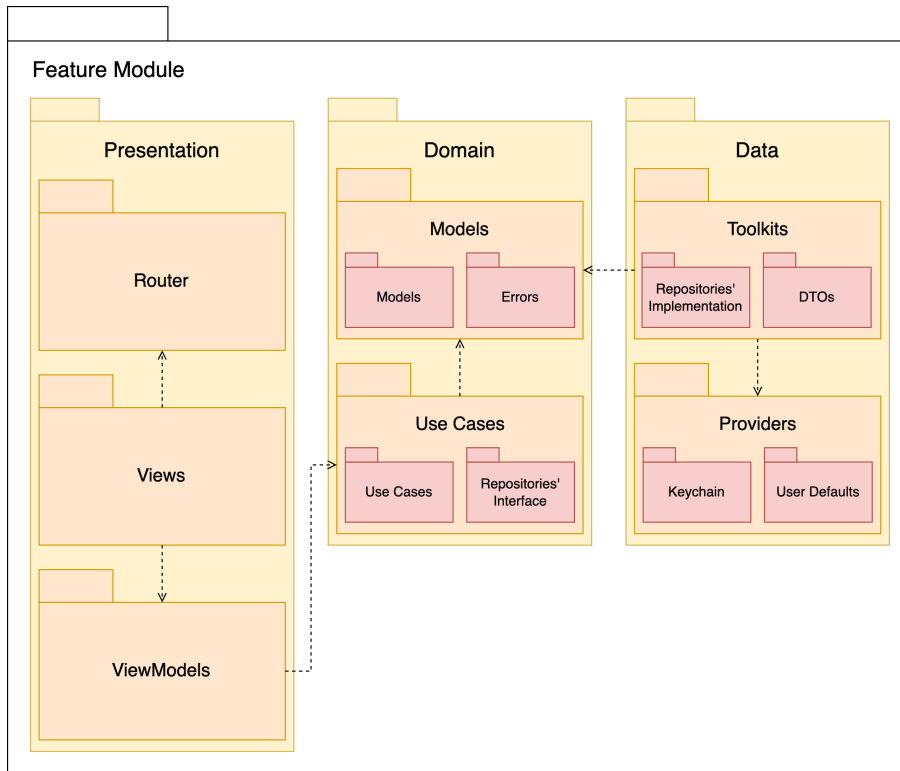


Figure 5.3: Architecture of the Feature Module

repositories ensure robust and consistent data management within the application, thereby enhancing its modularity and scalability for easier maintenance and potential system expansion. For example, a repository might fetch data using both the Keychain and User Defaults Providers to support a specific use case from the domain layer.

### 5.1.3.3 Presentation Layer

The *presentation layer* of the Feature module consists of `Views`, `ViewModels`, and a `Router`. The `Router` functions as a singleton (in SwiftUI often implemented as an `@EnvironmentObject`) and is passed throughout the presentation layer, managing the state of the view hierarchy.

`Views` are responsible for displaying the state contained within `ViewModels` to the user. If user interaction triggers a business logic operation, this is facilitated through use cases that are injected into the `ViewModel` via dependency injection. For instance, in a view displaying app settings, the settings content is maintained in the `ViewModel`. Meanwhile, actions invoked by user inputs are managed through use cases that originate from the domain layer. This structure ensures that the application's business logic remains separate from its user interface.

## 5.2 Database Schema

This section begins by defining the *entity-relationship diagram (ER Diagram)* notation, which is later used to create a database schema. Although Firestore, a document-based database, will be used, a database schema is still beneficial. Employing an ER diagram provides an abstract overview of the data that the application will store. This schema will later guide the implementation of the productivity-enhancing application. The schema is constructed using standard Software Engineering processes.

### 5.2.1 ER Diagram Notation

The physical data model is the deepest level of entity-relationship diagrams and is critical for refining the database structure. It includes comprehensive details of table structures such as column names, data types, constraints, keys, and relationships between tables. [16]

#### 5.2.1.1 Types, Keys and Fields

*Fields* in an ER diagram represent the attributes of the entity, typically visualized as columns within the database. These fields delineate the characteristics of the entity, such as *InterestRate* and *LoanAmount*.

*Keys* in ER diagrams categorize attributes to ensure efficient database organization. They are crucial for linking tables effectively. Primary keys uniquely identify an instance of an entity within a table. Foreign keys link tables together, often representing relationships like one-to-one or one-to-many.

*Types* in an ER diagram refer to the data types assigned to fields, affecting how data is stored and interacted with within the database. This can also encompass the conceptual types of entities involved. [16]

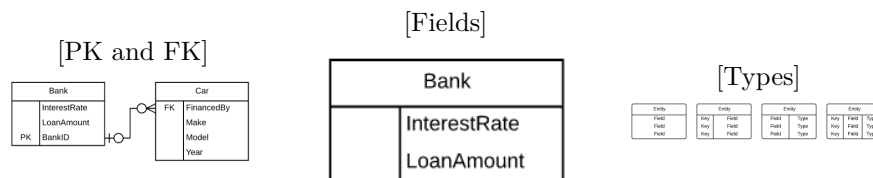


Figure 5.4: Fields, Private and Foreign Keys and Types Notation [16]

#### 5.2.1.2 Cardinality and Ordinality

*Cardinality* and *ordinality* describe the relationship constraints between two entities in a database. Cardinality specifies the maximum number of times an instance of one entity can relate to instances of another entity, while ordinality indicates the minimum required associations. These constraints are depicted in entity-relationship diagrams through the style of lines and endpoints, illustrating the range of possible relationships. This representation aids in visualizing and understanding the database structure's relational dynamics. [16] The notation is shown in Figure 5.5.

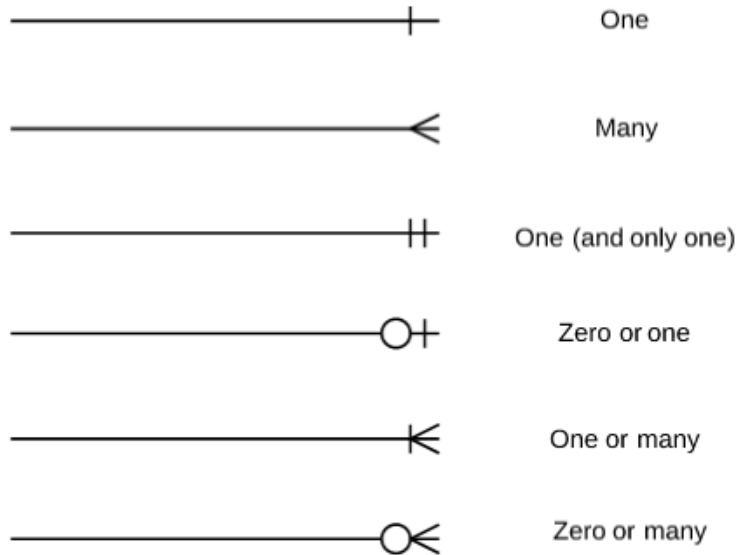


Figure 5.5: Cardinality and Ordinality Notation [16]

### 5.2.2 ER Diagram

This section presents the entity-relationship diagram for the productivity enhancing application, applying concepts previously discussed. It visually outlines the database structure and the relationships between entities, aiding in effective database design and development.

The ER diagram, depicted in Figure 5.6, visually represents the structure of a database designed to facilitate interactions within a productivity-enhancing application, encompassing entities like **User**, **Storyline**, **Guild**, **Invitation**, and **Member**. Each entity is characterized by a unique identifier designated as the *primary key (PK)*, such as `uid` for **User** and `id` for other entities. The **Storyline** entity, which records various user activities, connects to both the **User** and **Guild** entities via foreign keys (`userId` and `guildId`), indicating which user the storyline belongs to and the guild it is associated with if applicable.

Inter-entity relationships within the diagram articulate how users interact with different components of the application. The **User** entity, central to the model, links to **Invitation** and **Member** through the `userId`, signifying that a user can receive multiple invitations. Additionally, **Member** is associated with **Guild** through the `guildId`, illustrating members' affiliation with specific guild.

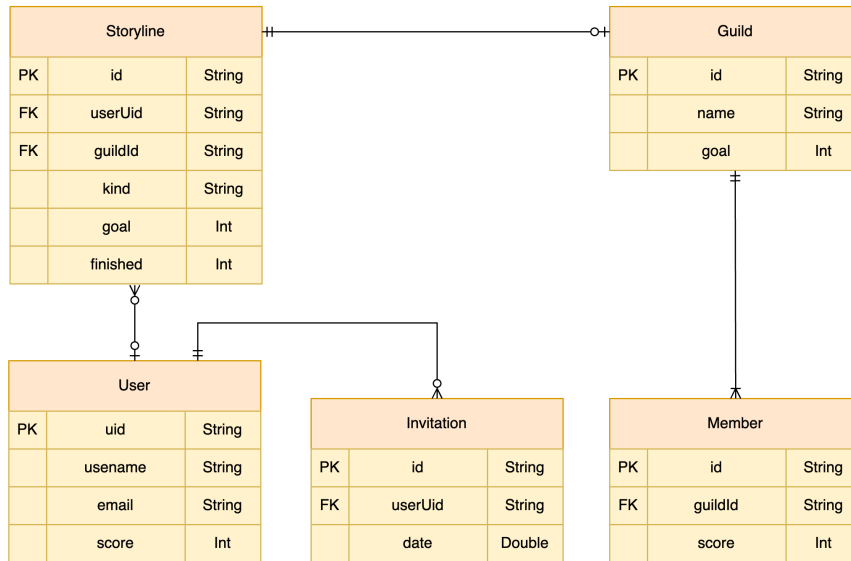


Figure 5.6: ER Diagram for Productivity-Enhancing Application

### 5.3 User Interface Design

In this section, key usability concepts by Jakob Nielsen are explored, followed by the reasoning behind the user interface design choices for the application, establishing the foundation of UI design concepts. Later in the section, wireframes are introduced and described in detail. Finally, miscellaneous user interface decisions are stated, such as support for device rotation, and the introduction and reasoning behind the app's name.

#### 5.3.1 Usability

Understanding *usability* principles is critical in the design of interactive systems, such as mobile applications, to ensure they meet user needs effectively and efficiently. Key concepts from the book *Usability Engineering* by Jakob Nielsen include:

- **Learnability:** This refers to how easily and intuitively new users can understand how to navigate and interact with the system. A user-friendly system should allow users to achieve their objectives with minimal effort and training. For instance, an app that is easy to navigate improves user onboarding and reduces frustration, leading to a better user experience.
- **Efficiency:** Once users have become familiar with the system that they are using, how quickly can they perform tasks? Usability aims to minimize the time and effort required for users to complete their tasks effectively. This involves streamlining user flows and optimizing interface elements to enhance productivity.

- **Memorability:** When users return to the design after a period of not using it, how easily can they reestablish proficiency? A usable system should be easy to remember, reducing the need for retraining and allowing users to quickly pick up where they left off, enhancing user satisfaction.
- **Errors:** This aspect deals with how many errors users make, how severe these errors are, and how easily they can recover from them. A well-designed system should minimize the occurrence of errors and offer clear recovery paths, which helps in maintaining user confidence and reducing frustration.
- **Satisfaction:** Ultimately, how pleasant is it to use the design? User satisfaction is a subjective measure of the overall user experience. A usable system should not only meet the functional needs but also deliver a positive emotional experience, making the interaction enjoyable and engaging. [113]

By integrating these usability principles into the application design, it is possible to create more effective, efficient, and enjoyable applications that align with user expectations and preferences, thereby enhancing overall user engagement and satisfaction.

### 5.3.2 Colors

Based on the analysis of psychological influences of colors discussed in Section 2.1.5, it has been determined that the design of the productivity-enhancing app should embody a friendly, cheerful, and motivating atmosphere. To achieve this, the design will utilize bright, pastel colors. The *primary color* selected is green (#4A887C), which symbolizes balance and harmony, essential for creating a calming and focused environment. The *secondary color*, yellow (#FFF38B), represents warmth and energy, contributing to an uplifting and invigorating user experience. Together, these colors aim to foster a pleasant and productive environment that enhances focus and motivation for its users. This thoughtful choice of color palette is expected to positively influence user engagement and satisfaction by aligning the app's aesthetic with its functional goals. The primary and secondary colors are shown in Figure 5.7.

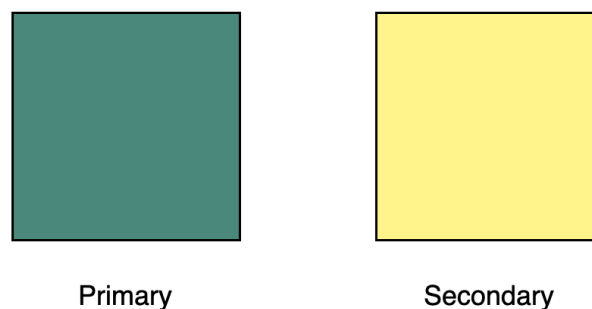


Figure 5.7: Primary and Secondary Colors for the UI Design

### 5.3.3 Wireframes

To sketch a basic concept of the user interface for the productivity-enhancing application, a *lo-fi (low fidelity) wireframing technique* is employed. Wireframing is an essential step in user interface design, establishing the basic structure of the app's interface. By mapping out essential interface elements, it enables effective planning of content and functionality to meet user needs. Wireframes facilitate iterative adjustments that are both cost-effective and time-efficient, ensuring the final design aligns with user expectations and business goals. [114]

The wireframes for this productivity-enhancing application are crafted based on concepts previously defined and analyzed in earlier chapters. These wireframes are created using a tool called *proto.io* [115], a prototyping tool utilized in the design and development of web and mobile applications. It enables the creation of both wireframes and detailed, interactive prototypes that closely simulate the final products. Wireframes of the main screens are presented in the following sections. For a comprehensive view of all wireframes, please refer to Appendix C.

#### 5.3.3.1 Authentication

The design of all screens in this section aims for a clean and minimalist aesthetic, enhanced with images which align with the overall design language of the application. Buttons are always displayed in contrasting colors, utilizing the primary and secondary colors defined earlier in this chapter.

Upon launching the application, the user is presented with the initial authentication screen. Here, they can choose to either log in or sign up. Depending on their choice, they are redirected to the login screen, which also offers access to password recovery if needed, or the sign-up screen. All screens consistently adhere to the established design aesthetic. For a more detailed view, see Figure 5.8.

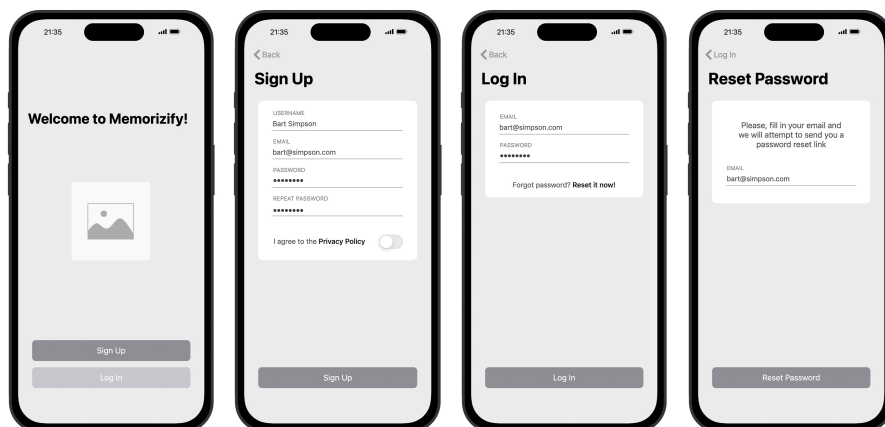


Figure 5.8: Wireframes of the Authentication Screens



### 5.3.3.2 Navigation Bar

After logging in, a standard approach to navigation is employed. The interface features a navigation bar with five tabs at the bottom, as depicted in Figure 5.9. These tabs are *Home*, *Storylines*, *Guilds*, *Board*, and *Settings*, and they are further described in the following sections.



Figure 5.9: Wireframe of the Navigation Bar

### 5.3.3.3 Storylines

The *Storylines* tab showcases available storylines, each represented as a tile with a title and a descriptive image that complements the app's design language. Each tile is interactive; tapping on one opens a detailed view that presents a teaser of the storyline along with a button to initiate a new storyline setup. During this setup phase, users can set goals for the storyline and adjust settings for the Pomodoro timer. Once satisfied with their configurations, users can create the new storyline by tapping a button, which then adds a new tile to the Home tab.

On the Home tab, users can start the storyline timer by tapping this newly added tile. Additionally, each storyline tile on the Home tab is equipped with a context menu for updating or deleting the storyline. The Home tab also features a plain Pomodoro timer, accessible by tapping the Plain Timer tile located at the top of the screen. This design facilitates easy access to both specialized (Storylines) and general (Plain Timer) productivity tools within the app. Wireframes for the Storylines are illustrated in Figure 5.10.



Figure 5.10: Wireframes of the Home and Storylines Tabs

### 5.3.3.4 Guilds

The *Guilds* tab features a similar interface to the storylines list on the Home tab, ensuring familiarity and ease of use throughout the application. At the top of the Guilds tab, the invitations section displays invitations that others have sent to the user, with two buttons available to either accept or decline an invitation.

Below this invitations section, there is a list of guilds that the user is a member of, displayed as tiles. Tapping a guild tile opens a detailed view of the guild. At the top of this detail view, users can see information about the guild, such as its goals and the number of members, as well as a leader actions button. This button, available only to members with sufficient permissions, brings up a context menu offering options to invite members, change guild parameters, or delete the guild, depending on the member's permissions. Below this top information section, a timer section allows setting parameters and starting a Pomodoro timer for the guild. Further down, a list of members displays the scores and names of the users.

Returning to the Guilds tab, below the tiles for the user's guilds, there is an additional tile that leads to a new guild creation screen. To view the wireframes for these features, see Figure 5.11.

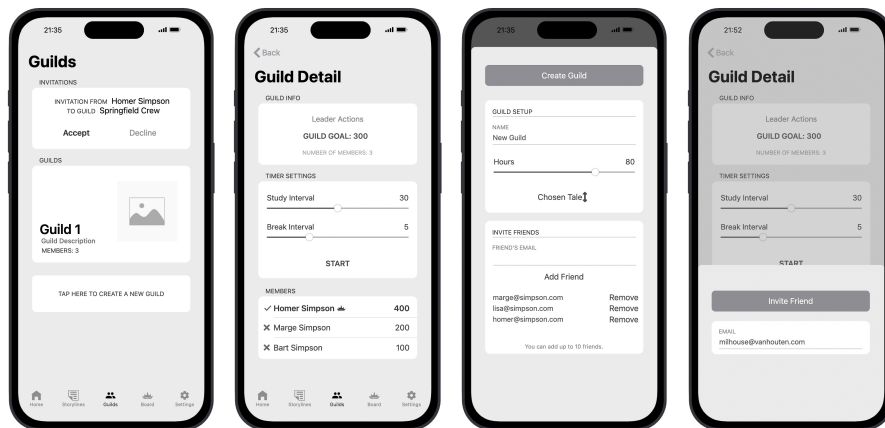


Figure 5.11: Wireframes of the Guilds Tab

### 5.3.3.5 Board

The *Board* tab displays a list of the top 10 users with the highest scores. If there are more than 10 users overall, a *Show More* button appears below the list. This button opens another screen that shows a comprehensive list of all users and their scores. Users can sort this list according to their preferences by using a context menu button located above the list. Wireframes of this section are illustrated in Figure 5.12.

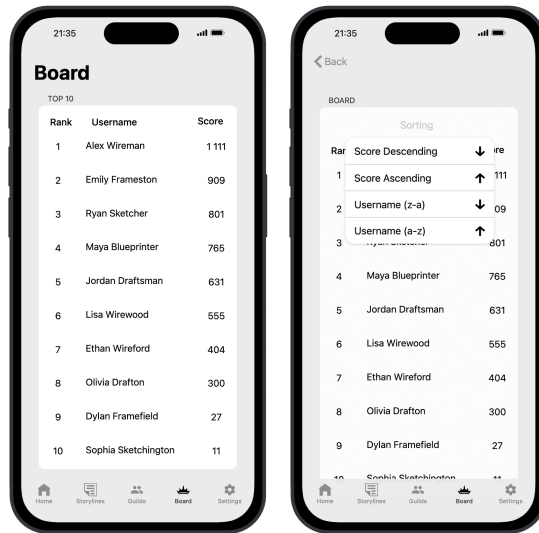


Figure 5.12: Wireframes of the Board Tab

### 5.3.3.6 Settings

The *Settings* tab adopts a design familiar to Apple users from the *System Preferences*, enhancing it with a user information section at the top of the settings list. This section displays the user's avatar, username, and email. Below this section, individual settings items allow users to perform actions such as changing their password, modifying the language, logging out, and more. For a detailed view of the wireframes for these sections, see Figure 5.13.

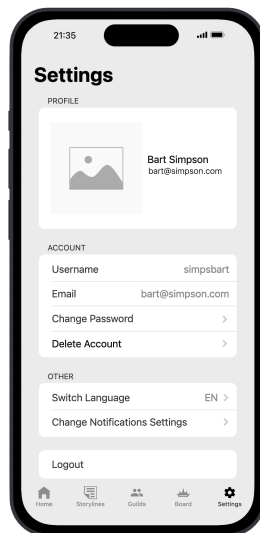


Figure 5.13: Wireframe of the Settings Tab

### 5.3.4 Miscellaneous

The application's design is made to ensure seamless adaptation across various screen sizes and device types, including iPads. Designed with flexibility and responsiveness in mind, it maintains optimal functionality regardless of the device employed. This universal accessibility enhances user experience, accommodating diverse user preferences.

Moreover, the selection of the name *Memorizify* was the result of a comprehensive analysis of the application's functionality. With a focus on memory enhancement and cognitive exercises rather than creative tasks, as discussed in Chapter 2, the chosen name embodies the essence of the application succinctly. Its contemporary and trendy appeal aligns with prevailing app naming conventions, facilitating easy recall among users.

## 5.4 Summary of Design

This chapter has followed standard Software Engineering processes. After analyzing various options, appropriate architecture and design patterns were selected and elaborated upon. Subsequently, a database schema was designed, followed by the prototyping of the user interface using wireframes accompanied by detailed descriptions. These efforts collectively fulfill the requirements of the thesis assignment.

---

## Implementation

The chapter about implementation details the entire process of developing the Memorizify application, based on findings from the previous design chapter. It begins with a description of the initial setup, followed by an exploration of the core components used in the project. Later in the chapter, an example workflow is thoroughly explained, demonstrating practical application of the concepts discussed. The chapter concludes with a noteworthy example of a particularly interesting part of the implementation and briefly mentions the implementation of other modules.

### 6.1 Intial Setup

Setting up a new project and its infrastructure correctly is a fundamental step for a successful project. This section provides a comprehensive overview of the necessary steps to properly set up a new Xcode project, establish a Firebase backend and integrate it with the application, configure *Jira*<sup>12</sup>, and set up *Bitbucket*<sup>13</sup>. Finally, the configuration of AppStore Connect and its TestFlight is demonstrated.

#### 6.1.1 Jira and BitBucket Setup

Jira and Bitbucket are platforms developed by Atlassian<sup>14</sup>. Jira is an agile project management tool that assists in planning, tracking, and releasing software. It offers a centralized platform for managing both simple and complex projects, enhancing team alignment and communication throughout the development lifecycle. [116]

Bitbucket is a Git-based code hosting and collaboration platform. It integrates with Jira to support the software lifecycle from planning to deployment, featuring tools like automated testing and secure deployment[117]. Utilizing these tools will benefit Memorizify by maintaining historical records and facilitating easier integration for future development by other developers.

---

<sup>12</sup>Jira is a project management tool for tracking and organizing tasks.

<sup>13</sup>Bitbucket is a version control and collaboration platform for software teams.

<sup>14</sup>Full company name is Atlassian Corporation Plc.

### 6.1.1.1 Jira Setup

Setting up Jira begins with creating a new Atlassian account, which provides universal access to all Atlassian platforms, including Jira, Bitbucket, and Confluence, the latter being used for documentation. After account creation, users encounter a questionnaire that probes their intentions with Jira, such as the type of project they plan to work on—be it a software project, management project, or other. This can be used for a quick setup.

However, to manually initiate a new project in Jira, one must choose from various project templates, each designed to serve specific purposes. Here are the three primary project types, each with its own template:

- **Kanban:** Ideal for projects that manage and visualize their work at various stages of the process. The *Kanban* template allows for continuous delivery and helps to manage workflow with greater flexibility.
- **Scrum:** Best suited for projects that plan their work in sprints, strictly following the agile development principles. The *Scrum* template helps organize tasks into manageable units with defined timelines, encouraging iterative progress, regular reflection, and adjustment.
- **Bug Tracking:** Designed to help to capture, assign, and prioritize bugs or issues. This *Bug Tracking* template is suitable for maintaining for more complex but high-quality software by ensuring all issues are tracked and resolved systematically. [118]

For the Memorizify project, a Kanban board was selected as the most suitable choice due to its simplicity, intuitiveness, and the way it mirrors the project's workflow dynamics effectively.

Another important decision to be made is whether the project will be team-managed or company-managed. Team-managed projects offer less functionality but allow greater control for non-administrator Jira users. On the other hand, a company-managed project unlocks all Jira features, some of which can only be managed by Jira administrators. [17] For Memorizify project, company-managed project will be used to be able to use all Jira features. Overview of the comparison of team-managed vs. company-managed is shown in Figure 6.1.

To complete the setup process, the user is required to enter the project name, which for this instance is *Memorizify iOS*. This crucial step triggers the generation of a project key, which acts as a unique identifier for the project within the workspace. This key is particularly important as it facilitates the management and retrieval of project-specific data across various system components. Although the key can be customized initially to suit organizational naming conventions, it is generally advisable to retain the original key once set. Changing the key later in the project lifecycle could necessitate a comprehensive reindexing of the Jira project database, potentially leading to disruptions in external references and integration links. Such changes could introduce complexities in project management and data consistency, making the initial choice of key quite significant. This final step of the project setup process, crucial for ensuring smooth project tracking and management, is illustrated in Figure 6.2.

The image compares two project management options in Jira: Team-managed and Company-managed.

Team-managed	Company-managed
<p><b>Set up and maintained by your team.</b></p> <p>For teams who want to control their own working processes and practices in a self-contained space. Mix and match agile features to support your team as you grow in size and complexity.</p> <p><b>Simplified configuration</b></p> <p>Get up and running quickly, with simplified configuration.</p> <ul style="list-style-type: none"> <li>Anyone on your team can set up and maintain</li> <li>Settings do not impact other projects</li> <li>Easy setup for issue types and custom fields</li> <li>Simple configuration for multiple workflows</li> <li>Access level permissions</li> </ul> <p>Select a team-managed project</p>	<p><b>Set up and maintained by your Jira admins.</b></p> <p>For teams who want to work with other teams across many projects in a standard way. Encourage and promote organizational best practices and processes through a shared configuration.</p> <p><b>Expert configuration</b></p> <p>Benefit from complete control with expert configuration, customization and flexibility.</p> <ul style="list-style-type: none"> <li>Set up and maintained by your Jira admins</li> <li>Standardized configuration shared across projects</li> <li>Complete control over issue types and custom fields</li> <li>Customizable workflows, statuses and issue transitions</li> <li>Detailed permission schemes</li> </ul> <p>Select a company-managed project</p>

Figure 6.1: Jira Team-Managed vs. Company-Managed Projects [17]

The screenshot shows the 'Add project details' screen in Jira. It includes the following elements:

- Section Header:** Add project details
- Text:** Explore what's possible when you collaborate with your team. Edit project details anytime in project settings. Required fields are marked with an asterisk \*
- Name:** A text input field containing 'Memorizify iOS'.
- Key:** A text input field containing 'MI'.
- Checkbox:** An unchecked checkbox labeled 'Share settings with an existing project'.
- Template Selection:** A section titled 'Template' with a 'Change template' link. It shows a 'Kanban' template with the Jira logo and the description: 'Visualize and advance your project forward using issues on a powerful board.'
- Type Selection:** A section titled 'Type' with a 'Change type' link. It shows a 'Company-managed' type with the description: 'Work with other teams across many projects in a standard way.'
- Navigation:** 'Cancel' and 'Next' buttons at the bottom right.

Figure 6.2: Jira Setup Overview

Finally, the Jira project is created, and the project home screen, which is the Kanban board, is displayed as shown in Figure 6.3. The Kanban board features swimlanes for organizing issues into categories. These swimlanes reflect the issue lifecycle states based on the specific setup.

For the Memorizify project, a board with four swimlanes will be used. These are *Backlog*, *Selected for Development*, *In Progress*, and *Done* which also reflect allowed states of issue within the project. The Backlog state holds issues

## 6. IMPLEMENTATION

---

that are ideas for potential changes to the application that may eventually be implemented. Selected for Development contains issues that are planned to be implemented soon. The In Progress state includes issues currently under implementation or review. Finally, the Done state consists of issues that have been resolved and closed.

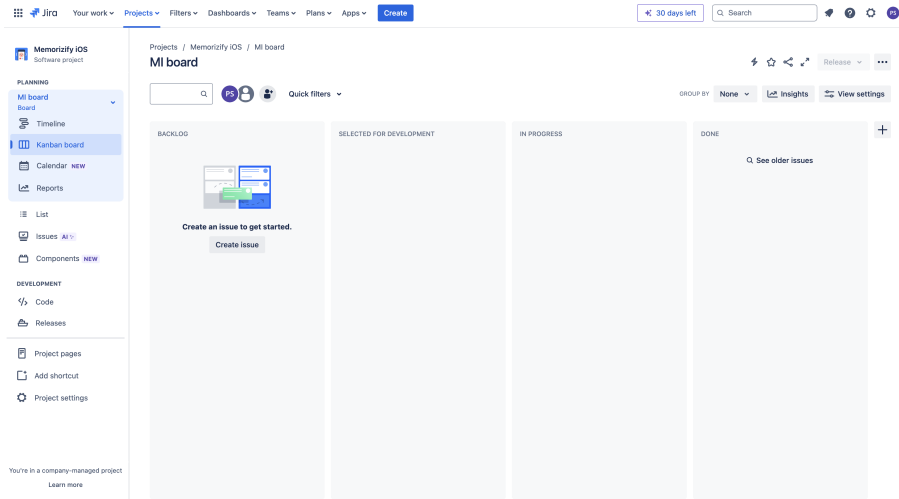


Figure 6.3: Jira Kanban Board After Project Creation

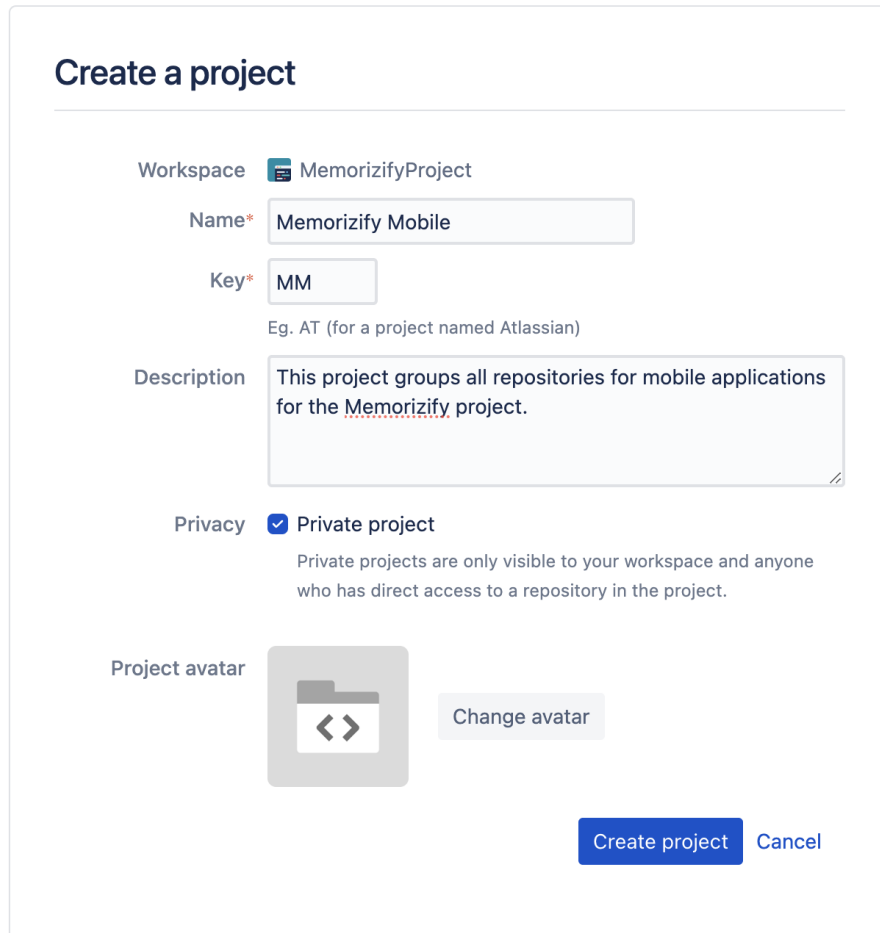
From the Jira home screen, it is possible to create new issues of various types (such as *Task*, *Bug*, *Epic*, and *Story*) using the blue *Create* button located on the top bar of the page. When creating a new issue, it is recommended to not only fill in its name but also provide a detailed description to clarify the purpose of the issue and the criteria for its completion, known as the *Definition of Done (DoD)*. Additional information for the issue can be specified, such as its label. For Memorizify, labels will categorize features according to the MoSCoW framework. Once created, the issue appears in a swimlane. For detailed view of the issue creation, see Figure 6.4.

In summary, Jira will be utilized as the central platform for issue tracking and management within the Memorizify project. This system will play a pivotal role in enabling an effective and clear definition of the project's implementation goals. By organizing and maintaining detailed records of all project activities and issues, Jira not only enhances the visibility and accountability of the development process but also greatly improves project coordination. Furthermore, it facilitates the thorough tracking of the project's history, preserving a comprehensive log of all decisions and changes. This historical record is invaluable for audit purposes and for understanding the evolution of the project over time. Additionally, should the need arise, the detailed and organized nature of Jira will allow new developers to seamlessly integrate into the app development process. This capability ensures that the project can adapt to expanding development needs, making it easier to onboard new team members who can quickly understand and contribute to ongoing tasks.





Once a workspace is established, a new project can be initiated using the blue **Create** button located on the top bar. Similar to a Jira project, a project in Bitbucket has its name and is identified by its key. This process is detailed in Figure 6.5. It's important to note that repositories are grouped within projects within the workspace. Each repository must belong to a project, which enhances the organization of repositories.



The screenshot shows the 'Create a project' form in Bitbucket. The form is titled 'Create a project' and is set within the 'MemorizifyProject' workspace. The 'Name\*' field contains 'Memorizify Mobile' and the 'Key\*' field contains 'MM'. A note below the key field states 'Eg. AT (for a project named Atlassian)'. The 'Description' field contains the text: 'This project groups all repositories for mobile applications for the Memorizify project.' The 'Privacy' section has the 'Private project' checkbox checked, with a note: 'Private projects are only visible to your workspace and anyone who has direct access to a repository in the project.' The 'Project avatar' section shows a folder icon with a code symbol and a 'Change avatar' button. At the bottom right, there are two buttons: 'Create project' (blue) and 'Cancel' (grey).

Figure 6.5: BitBucket Project Creation

Upon creating a new project, a repository for the Memorizify iOS application can be initialized. This process also involves utilizing the *Create* button, where Bitbucket prompts input for the repository name and key (optionally), along with minor repository settings such as specifying the inclusion of a gitignore file and/or a readme file. The setup page is depicted in Figure 6.6.

Upon creation, the main repository page is displayed, as shown in Figure 6.7, offering instructions for initiating a project within the repository.

In summary, a new Bitbucket repository has been established for the Memorizify iOS application, facilitating a code versioning system for tracking his-

**Create a new repository** [Import repository](#)

Workspace MemorizifyProject

Project\* Memorizify Mobile

Repository name\*

Access level  Private repository  
 Uncheck to make this repository public. Public repositories typically contain open-source code and can be viewed by anyone.

Include a README?

Default branch name

Include .gitignore?

[> Advanced settings](#)

[Create repository](#) [Cancel](#)

Figure 6.6: BitBucket Repository Creation

Bitbucket Your work Pull requests Repositories Projects People More [Create](#)

Memorizify iOS

Source Pipelines Deployments Jira Issues Security Downloads Repository settings

**Let's put some bits in your bucket**

HTTPS `git clone https://peemjka@bitbucket.org/memorizifyproject/memorizify`

**Get started quickly**  
 Creating a README or a .gitignore is a quick and easy way to get something into your repository.  
[Create a README](#) [Create a .gitignore](#)

**Get your local Git repository on Bitbucket**

Step 1: Switch to your repository's directory

```
1 cd /path/to/your/repo
```

Step 2: Connect your existing repository to Bitbucket

```
1 git remote add origin https://peemjka@bitbucket.org/memorizifyproject/memorizify-ios.git
2 git push -u origin main
```

Need more information? [Learn more](#)

Figure 6.7: BitBucket Initial Page

tory and potential future collaborative development by teams. This repository will store an Xcode project, the setup of which will be detailed in one of the upcoming sections. The subsequent section will demonstrate the integration process between Jira and Bitbucket, unlocking the full potential of these two tools.

### 6.1.1.3 Jira and BitBucket Integration

The integration between Jira and Bitbucket provides a collaborative environment for software development. By linking these two powerful tools, it is possible to streamline workflow and enhance productivity. One key feature of this integration is the ability to effortlessly link issues in Jira to branches, pull requests, and commits in Bitbucket, providing clear traceability between code changes and project tasks. Additionally, automatic updates in Jira reflect changes made in Bitbucket, ensuring that all developers stay informed about the progress. This integration also enables smart commits, allowing developers to update Jira issues directly from their commit messages, further enhancing efficiency and reducing the need for manual updates. Overall, the integration between Jira and Bitbucket empowers teams to collaborate more effectively, improve transparency, and accelerate the software development process. [119]

To begin the integration, one must ensure that this feature is enabled. The setting for this feature can be found in Jira's left menu bar, under *Project Settings — Features — Code*. When this feature is enabled, the integration itself is completed by navigating to the *Code* menu item on the main page of Jira. This option allows users to find and link the repository in Bitbucket to the Jira project, as shown in Figure 6.8.

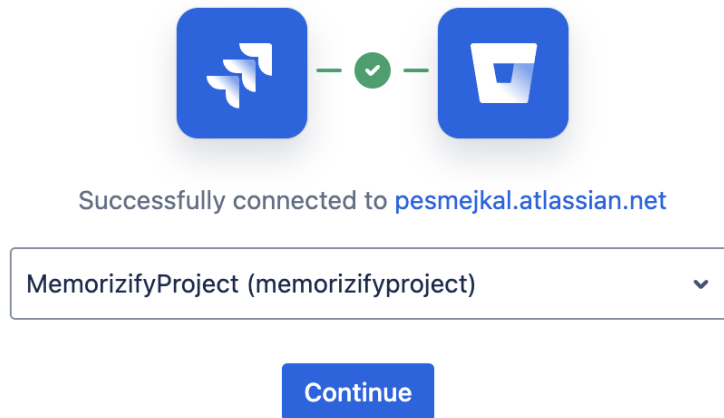


Figure 6.8: Jira-BitBucket Integration

Once this connection is established between the development tools, commits, branches, and pull requests in the repository can be linked directly to specific Jira issues by including the Jira project key in the commit message or within the names of the branches or pull requests.

Establishing such a link means that the project key acts as a direct connector to the issue, enabling to view details of Jira issues right from within the Bitbucket repository. This integration facilitates a streamlined workflow where updates to code and corresponding tracking in Jira are seamlessly interconnected, enhancing project transparency.

### 6.1.2 Xcode Project Setup

Once the Jira and Bitbucket tools are set up and linked, it's time to create an Xcode project. When creating an Xcode project, the first step is to select the correct platform, which for the Memorizify app is iOS. Then, there are several templates from which a project can be initialized, such as *Game*, *Document App*, *Augmented Reality App*, *Safari Extension*, and more. For the purpose of Memorizify, the basic *App* template will be used, as it is the most suitable option for a general iOS application.

Once the platform and template are chosen, the project information form must be filled in. The form, depicted in Figure 6.9. Description of the fields follows.

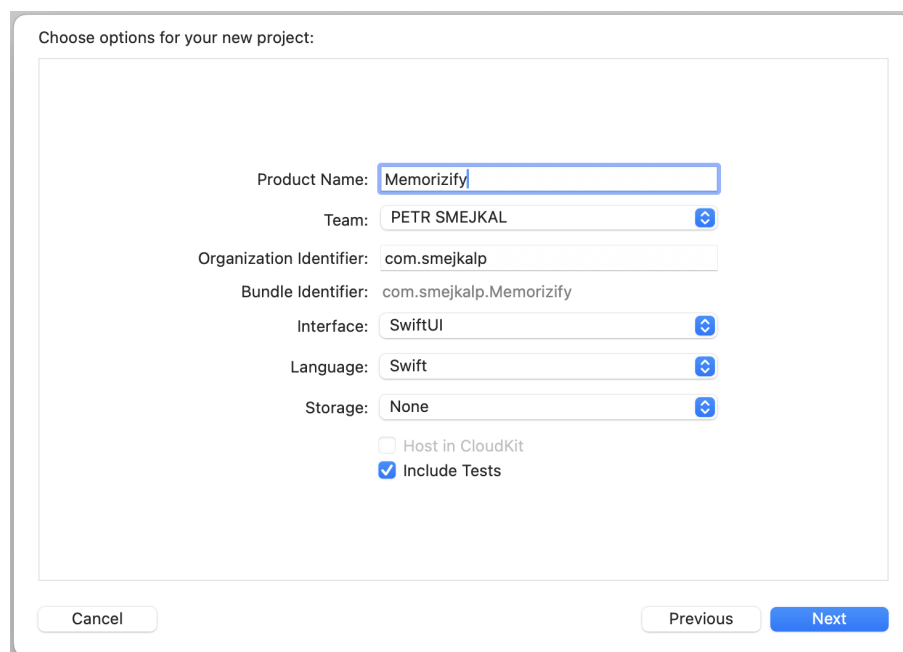


Figure 6.9: Xcode Project Creation

- **Product Name:** The full name of the iOS application.
- **Team:** The app must belong to a team in order to be deployable to App-Store Connect. The team is selected from a dropdown menu.
- **Organization Identifier:** The organization identifier is a unique string identifying the organization that the app is being developed under.
- **Bundle Identifier:** Bundle Identifier is a basic project property which is a unique identifier of the application. It usually consists of the format: `OrganizationIdentifier(dot)ProductName`.

- **Interface:** This dropdown menu offers to choose frameworks for UI building. The options are SwiftUI and UIKit.
- **Language:** Select the language for the project. Xcode 15 offers only Swift. Previously, *Objective-C* was also an option.
- **Store:** Optionally, select which kind of storage framework will be used. The options are *SwiftData* and *Core Data*, with the former being a newer, more modern solution.
- **Host in CloudKit:** If previously set up, it is possible to opt in to host the app on the *CloudKit* Apple service by checking the checkbox.
- **Include Tests:** If checked, Xcode will prepare targets for unit tests and UI tests. The targets can be added manually later.

After clicking on the *Next* button, the destination for the project files must be chosen and confirmed, thereby creating the new project. Subsequently, the project needs to be initialized with a Git repository and pushed to Bitbucket. This setup can be accomplished using a terminal window and a command sequence, as detailed in Figure 6.10. It's important to note that in order to communicate with the Bitbucket server, a private-public key pair must be set up between the local machine and Bitbucket servers.

```
MacBook-Pro: smejkalp$ cd Memorizify/  
MacBook-Pro: smejkalp$ git init  
MacBook-Pro: smejkalp$ git add .  
MacBook-Pro: smejkalp$ git commit -m "Initial commit"  
MacBook-Pro: smejkalp$ git remote add origin <repository-url>  
MacBook-Pro: smejkalp$ git push -u origin main
```

Figure 6.10: Xcode Project Git Setup

This completed the Xcode setup, preparing the project for development. At this point, it is possible to create issues in Jira, generate branches from those issues in Bitbucket, and pull them to this local repository. Once changes are made, they can be merged into the main branch using a pull request, establishing a complete workflow used for development of Memorizify. The workflow ensures efficiency, excellent history trackability, and fine organization. This structured approach not only streamlines the development process but also enhances the ability to monitor changes and progress over time.

### 6.1.3 Firebase Setup

This section will describe the process of integrating Firebase backend services with an Xcode project. The setup procedure may vary depending on the platform for which the Firebase services are being used; this text focuses on the iOS setup. The following sections detail the steps necessary to complete the setup.

#### 6.1.3.1 Create New Firebase Project

A new Firebase project can be created on the *Firebase Console* website, which serves as a central hub for managing Firebase services. To initiate a new project, one must first log in with a Google account (or create one if necessary). Subsequently, a new Firebase project can be set up with the following information:

- **Project Name:** The name of the Firebase project. Based on this name, a unique project identifier is generated, which is then used in various communications, such as email verification and password reset emails sent to users.
- **Google Analytics (Optional):** Enabling Google Analytics is an optional step when creating a new Firebase project. This service provides detailed statistics about the applications within the project.
- **Region:** Selecting a region from a list of available options can impact the latency of Firebase services depending on the geographical location of the user base. This option is only available when setting up the first project.

#### 6.1.3.2 Register iOS App

Once the project is created, it is possible to register a new iOS application within the project. This involves the following steps:

- **App Name (Optional):** The name of the iOS application. This name helps organize applications within the project.
- **AppStore ID (Optional):** The ID of the application on the App Store.
- **Bundle ID:** As previously mentioned, the Bundle ID is a unique identifier for the Xcode project. Firebase uses the Bundle ID to recognize the newly registered iOS app.

Based on the information provided during the Firebase backend setup, a `GoogleService-Info.plist` file will be generated. This file serves as the sole configuration file for Firebase services within the application and must be incorporated into the Xcode project. After adding it to the Xcode project, the *Firebase SDK* must be installed using SwiftPM. To add the Firebase SDK, navigate to *File — Add Package Dependencies...* in Xcode. This opens a SwiftPM window, where you can search for and add the Firebase SDK to the project, as depicted in Figure 6.11.

Once the Firebase SDK has been incorporated into the Xcode project, it is possible to configure the Firebase services using the configuration file upon

## 6. IMPLEMENTATION

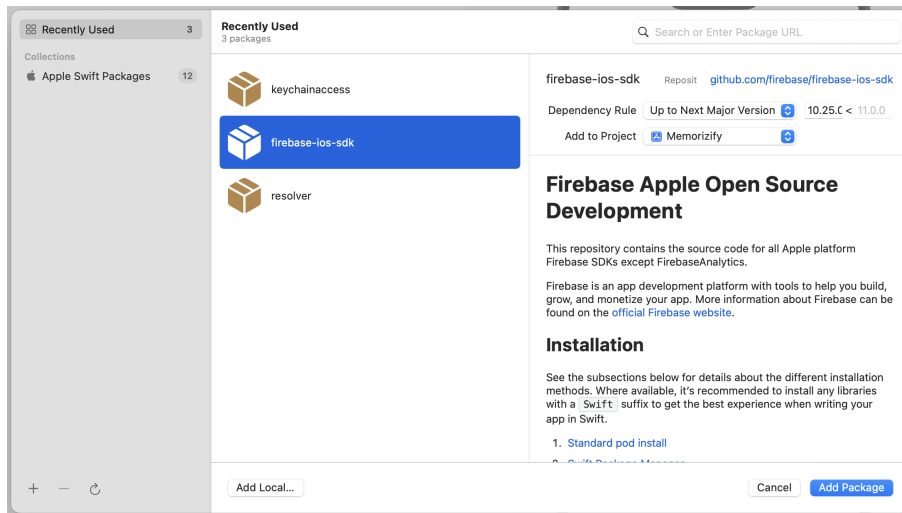


Figure 6.11: Adding Firebase SDK Dependency to Xcode

application startup. Since new Xcode projects no longer include `AppDelegate` class by default, this class must be manually added to handle application events such as application launch, termination, deep links, push notifications, and others. To handle `AppDelegate` events, the class must first be created. This process is illustrated in Figure 6.12, where the `AppDelegate` contains only one method, which is invoked upon the successful launch of the app.

```
class AppDelegate: NSObject, UIApplicationDelegate {
    func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [
            UIApplication.LaunchOptionsKey: Any
        ]? = nil
    ) -> Bool {
        // code will later go here
        return true
    }
}
```

Figure 6.12: AppDelegate Class

When the `AppDelegate` class is created, it must be linked to the `@main` annotated `SwiftUI App` struct, which serves as the entry point of the application. This linkage is demonstrated in Figure 6.13.



```

@main
struct MemorizifyApp: App {
    @UIApplicationDelegateAdaptor(AppDelegate.self)
    var appDelegate

    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}

```

Figure 6.13: AppDelegate Connection to App Struct

After these steps, the `AppDelegate` can be modified to incorporate the Firebase SDK. Its method, which is called on `didFinishLaunchingWithOptions`, should invoke `FirebaseApp.configure()`. This command will locate the Firebase `.plist` configuration file if it is in the root project directory. If the file is located elsewhere, its path must be specified in the `configure()` function. Completing these steps will finalize the Firebase setup for the Memorizify application, enabling the selection and configuration of suitable services in the Firebase Console, which can then be fully utilized in the application code.

#### 6.1.4 AppStore Connect Setup

Setting up AppStore Connect is essential for deploying apps to both TestFlight and the App Store. This section will provide a detailed description of configuring AppStore Connect for TestFlight deployment, which simplifies the distribution of iOS apps to beta testers. It is also important to note that access to AppStore Connect requires a subscription to the Apple Developer Program, which is an annually paid service.

After logging into AppStore Connect on their website, it is necessary to log in with the same *Apple Developer Account* in Xcode. Initially, the application's Bundle ID must be registered in Xcode's *Identifiers* section. Details such as the application type and capabilities need to be provided. This setup is illustrated in Figure 6.14. Once complete, the application can be registered in AppStore Connect.

**Certificates, Identifiers & Profiles**

[All Identifiers](#)

**Register an App ID** Back Continue

Platform  
iOS, iPadOS, macOS, tvOS, watchOS, visionOS

App ID Prefix  
4CSWQDTY94 (Team ID)

Description  
Productivity-Enhancing Application

Bundle ID  Explicit  Wildcard  
com.smeikalp.Memorizify

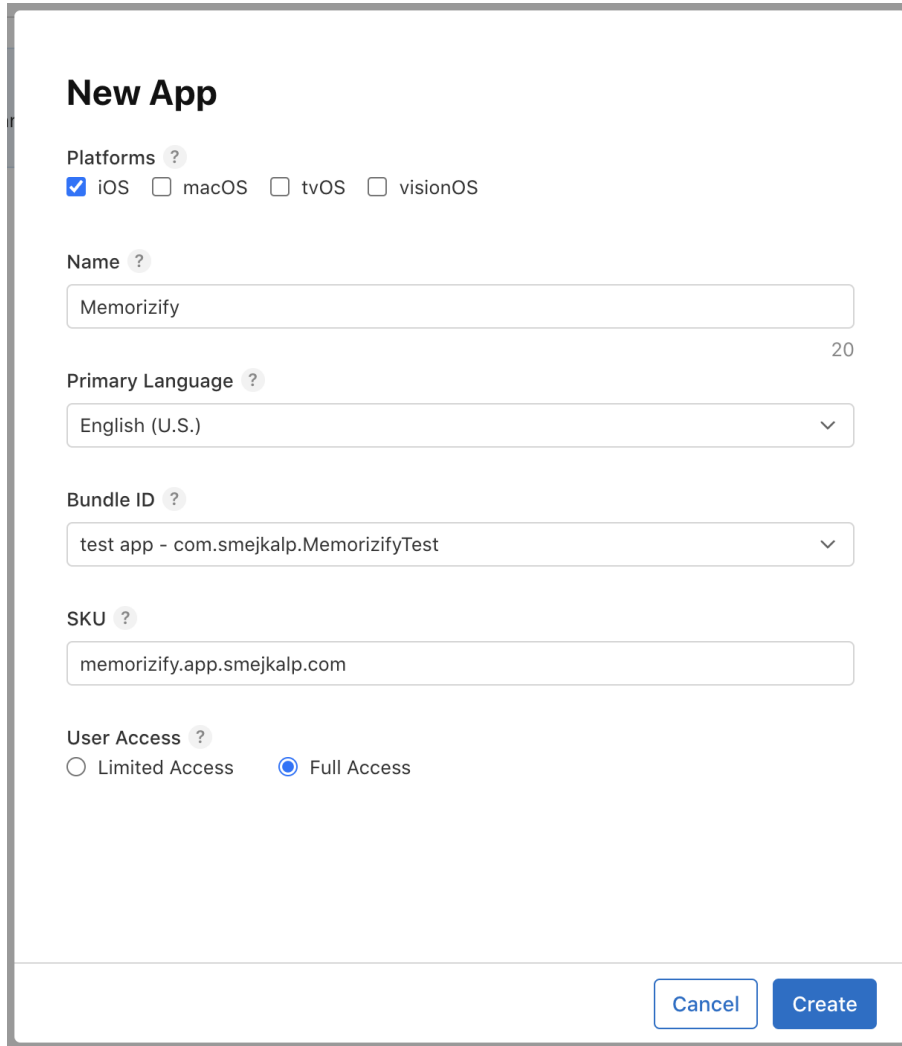
You cannot use special characters such as @, &, \*, \*  
We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (\*).

Figure 6.14: Bundle Identifier Registration

## 6. IMPLEMENTATION

---

The app can be registered in AppStore Connect by navigating to the *New App* button in the *Apps* section. This action opens a form to enter details about the application being registered. The required information includes the app's name, preferred language, Bundle ID, and SKU, which serves as a unique identifier for the app within AppStore Connect. Additionally, the platform for which the app is developed and its availability must be specified, as depicted in Figure 6.15.



The screenshot shows the 'New App' registration form in AppStore Connect. The form is titled 'New App' and contains the following fields and options:

- Platforms ?**: A group of radio buttons for selecting the target platform. The 'iOS' option is selected, while 'macOS', 'tvOS', and 'visionOS' are unselected.
- Name ?**: A text input field containing the name 'Memorizify'. A character count of '20' is displayed to the right of the field.
- Primary Language ?**: A dropdown menu currently showing 'English (U.S.)'.
- Bundle ID ?**: A dropdown menu currently showing 'test app - com.smejkalp.MemorizifyTest'.
- SKU ?**: A text input field containing the value 'memorizify.app.smejkalp.com'.
- User Access ?**: A group of radio buttons for selecting the user access level. The 'Full Access' option is selected, while 'Limited Access' is unselected.

At the bottom right of the form, there are two buttons: 'Cancel' and 'Create'.

Figure 6.15: AppStore Connect App Registration

With this setup complete, the development toolchain is fully established, and development of the Memorizify application can begin. Tester groups and individual testers can be added to the app through AppStore Connect, which offers extensive management features for greater control over tester groups. Builds will then be deployed to AppStore Connect straightforwardly

from the Xcode IDE. Testers will subsequently be notified via email that a new build is available for testing.

## 6.2 Core Components

The section on core components outlines the application of the `NSLog` framework, explains the usage of dependency injection, and discusses the implementation of string catalogs for localization. It concludes by delving into the implementation of navigation within the app.

### 6.2.1 NSLogging

To enable event logging within the application, a logging component was implemented using Apple's `NSLog` framework. It was determined that direct usage of `NSLog` methods would suffice for basic logging tasks. However, for logging Firebase events, a `static struct` named `MemorizifyLogger` was introduced. This struct provides functions specifically designed for logging the *fetch* and *update* actions related to Firebase documents. Figure 6.16 displays the signatures of these logging functions.

```
struct MemorizifyLogger {
    static func logDocumentsUpdate(
        file: String,
        line: Int,
        documentPath: String,
        data: [String: Any],
        description: String? = nil
    ) { /* logging of documents update */ }

    static func logDocumentsFetch(
        file: String,
        line: Int,
        documentPath: String,
        description: String? = nil
    ) { /* logging of documents fetch */ }

    static func logDocumentsDelete(
        file: String,
        line: Int,
        documentPath: String,
        description: String? = nil
    ) { /* logging of documents delete */ }
}
```

Figure 6.16: Skeleton of `MemorizifyLogger`

Together with the `#line` and `#file` constructs, which record the line number and file name where they are invoked, this structure creates a powerful framework for logging.

### 6.2.2 Dependency Injection

*Dependency injection (DI)* is a software design pattern that enhances modularity and maintainability by managing dependencies between classes through *Inversion of Control (IoC)*. Rather than classes creating dependencies internally, DI facilitates the external provision of dependencies, which reduces hard-coded dependencies and promotes component decoupling. This approach allows easy runtime changes and simplifies testing, as components can receive different implementations of their dependencies, such as service interfaces, without needing code modifications. By decreasing coupling and increasing component reusability, DI contributes significantly to improving the scalability and maintainability of software systems. [120]

In the Memorizify project, dependency injection framework called Resolver will be used. Resolver is a lightweight dependency injection framework for Swift 5.x on iOS, supporting various injection methods like constructor, property, and service locator. It streamlines dependency management by enabling simple registration and resolution of services, and is known for its speed and ease of use, encapsulated in a single file. Although now deprecated and replaced by Factory, Resolver is celebrated for its efficient handling of automatic type inference, custom containers, and cyclic dependencies, making it highly effective in production environments. [121]

The *Resolver* framework package can be added in the same way as the Firebase SDK, as detailed in Section 6.1.3. Dependencies should be registered at app launch in the `AppDelegate`, similar to the configuration of Firebase. Subsequently, it can be utilized to inject dependencies throughout the Memorizify application. The usage of Resolver will be showcased later in this chapter.

### 6.2.3 String Catalog

Since Memorizify is designed to support multiple languages, a *string catalog* will be employed to manage localizations. A string catalog in Xcode centralizes all localizable strings, managing translations and pluralizations for various languages, and simplifies updates and maintenance by automatically tracking changes within the project. Adding string catalogs to the project through Xcode enhances the ease and accuracy of application localization. A populated string catalog is illustrated in Figure 6.17.

To add a new string catalog, a new file can simply be created in Xcode and String Catalog file type should be selected. This action initiates the automatic detection of all localizable strings within the application's code base upon building, marking them for translation. However, not all strings will be identified automatically, as Xcode searches for the `LocalizableStringResource` type, not `String`. Common UI elements such as `Text` or `Button` that use `LocalizableStringResource` in their constructors will automatically be added to the catalog. To manually add a string of the `String` type to the catalog, use the initializer with the `localized` label, as in: `String(localized: "String to be localized")`. This ensures the string is included in the catalog.

Key	Default Localization (en)	Czech (cs)	Comment	State
Failed to load the guild data. Please try again.	Failed to load the guild data. Please try again.	Chyba při načítání cechu. Prosim, zkuste to znovu.		⊙
Done	Done	Hotovo		⊙
Your email is not verified. Do you wish to resend the email verification link to your email?	Your email is not verified. Do you wish to resend the email verification link to your email?	Váš email není potvrzen. Přejete si zaslat nový potvrzovací odkaz na Váš email?		⊙
From Trash to Treasure	From Trash to Treasure	Z odpadu k pokladu		⊙
Tilly's Contributions	Tilly's Contributions	Tillyiny příspěvky		⊙
Loading Board Failed	Loading Board Failed	Chyba při načítání tabulky		⊙
Sign Up	Sign Up	Registrovat		⊙
An error occurred when sending verification email. Please try again.	An error occurred when sending verification email. Please try again.	Při zasílání ověření nastala chyba. Prosim, zkuste to znovu.		⊙
Guild Setup	Guild Setup	Nastavení cechu	Comment	⊙
An error occurred when removing the user. Please try again.	An error occurred when removing the user. Please try again.	Při odebrání žlena nastala chyba. Prosim, zkuste to znovu.		⊙

Figure 6.17: Xcode String Catalog

This approach ensures Memorizify is ready for internationalization, simplifying updates and reducing complexity in supporting multiple languages. The use of a string catalog enhances adaptability and user experience by making content accessible and relevant across various languages.

### 6.2.4 Navigation Routing

Last for the *Core Components* section, a routing system is defined and explained in detail. There are several navigation options in SwiftUI applications, but for a project like Memorizify, the most suitable is the `NavigationStack`, introduced in iOS 16.0. This option was chosen for its reliability and ease of setup and use, in contrast to other historically used approaches that often required a steep learning curve or other SwiftUI options with limited features. [122]

To set up the routing, a `Router` class will manage the state of the view hierarchy. This class maintains the state in a navigation path, as illustrated in Figure 6.18. It is designated as an `ObservableObject`, enabling views that use the `path` variable, marked `@Published`, to automatically respond to changes in the routing state.

```
import SwiftUI

final class Router: ObservableObject {
    // Navigation path for the application
    @Published var path = NavigationPath()
}
```

Figure 6.18: Router Class Skeleton

This approach offers precise control over which views are displayed and allows for responses to events such as deeplinks, which may trigger the addition

or removal of a view to or from the `path` variable, which behaves like any other array.

With the `Router` class established, it is necessary to define the routes available for navigation within the app's component. These routes are specified in an `enum` that conforms to the `Hashable` protocol. It is important to note that Swift enums can have associated values, such as a `String`, as depicted in Figure 6.19. This feature is beneficial for passing data to views being instantiated.

```
// Defines available routes (screens)
enum ComponentRoute: Hashable {
    case firstScreen
    case secondScreen(String)
}
```

Figure 6.19: ComponentRoute Enumeration

To utilize the state for routing, the `Router` must be initialized at the root level of the application and marked with `@EnvironmentObject`. It can then be passed to a view using the `.environmentObject()` modifier. This setup is detailed in Figure 6.20.

```
import SwiftUI

@main
struct MemorizifyApp: App {
    // Initialize Router class as env. object
    @EnvironmentObject var router = Router()

    var body: some Scene {
        WindowGroup {
            ContentView()
            // Pass Router to view
            .environmentObject(router)
        }
    }
}
```

Figure 6.20: Router Class Initialization and Passing

Additionally, the root view's `body` must contain a `NavigationStack`, incorporating the `path` property. With this configuration, it is possible to respond to changes in the `Router`'s `path` variable using the `.navigationDestination()` view modifier applied to the root view of the hierarchy. To modify the path (to present or dismiss a view), the `path` can be altered from within any view

in the hierarchy by calling `.append()` or `.remove()` on the `path`. This functionality is illustrated in Figure 6.21.

```
import SwiftUI

struct ContentView: View {
    // "Catching" Router passed from parent Scene
    @EnvironmentObject var router = Router()

    var body: some View {
        // Using the Router's path for this root view
        NavigationStack(path: router.path) {
            VStack {
                Button("Present View") {
                    // Button tap presents FirstView() screen
                    router.path.append(ComponentRoute.firstScreen)
                }
            }
            .padding()
            // Reacting to path changes made from
            // this or any child view
            .navigationDestination(
                for: ComponentRoute.self
            ) { route in
                switch route {
                case .firstScreen:
                    FirstView()
                    // Passing Router to child view
                    .environmentObject(router)
                case let .secondScreen(name):
                    // Using "name" to initialize view
                    SecondView(name)
                }
            }
        }
    }
}
```

Figure 6.21: NavigationStack Usage Example

In summary, this setup establishes a robust and flexible navigation system for the Memorizify application, utilizing `NavigationStack` to efficiently manage view hierarchies and state changes. Integrating the `Router` class at the core of the application's structure allows dynamic handling of user interactions. The use of `NavigationStack`, in conjunction with `Router` and defined routes, streamlines development and ensures scalability and maintainability of the navigation system.

## 6.3 Typical Workflow

In this section, the typical workflow within the Memorizify app development process is detailed, starting from the creation of a Jira issue. It covers the subsequent steps of branch creation, followed by the implementation of changes. The process continues with the reviewing and merging of changes in the codebase. Finally, it concludes with the deployment of the new build to Testflight.

### 6.3.1 Issue and Branch Creation

Before the implementation begins, it is necessary to select the task to be addressed in Jira. Each feature task in Jira is labeled according to the MoSCoW method described in Section 3.3.1.3, facilitating clearer prioritization. Upon selection of the task, the *Create Branch* functionality is available in the task detail view. This leads to a new page that enables the creation of a new branch in a specified repository. In the example of implementing the feature, the branch is created in the *Memorizify iOS* repository and is categorized as a *feature* branch; BitBucket automatically prefixes the branch name with *feature/*. Once the branch is created, it can be pulled to the local repository and checked out, preparing the environment for development.

### 6.3.2 Module Implementation

Overall, the implementation of score addition will follow the example Feature detailed in Figure 5.3, as discussed in the previous chapter on design. For the feature currently being implemented, the process begins at the core of the architecture, within the domain layer, as described in Section 5.1.1 on Clean Architecture.

Initially, it is necessary to add models that represent the entities relevant to this issue. Consequently, a `User` structure will be introduced. A user in the Memorizify domain has the following properties: `username`, `email`, and `score`, where the latter is used to track the user's progress. The new `User` model is illustrated in Figure 6.22.

```
// Represents a user in the system
struct User: Codable {
    // The username of the user.
    let username: String

    // The email address of the user
    let email: String

    // The score of the user
    let score: Int
}
```

Figure 6.22: User Model



After creating the user model, it is necessary to define a use case for the described functionality. In this instance, the requirement is to create a use case that can increase the user's score; therefore, a new `IncreaseUserScoreUseCase` protocol is defined. Memorizify adheres to a convention whereby use cases only have one `execute()` method. Since this method involves communication with the Firebase backend, it must be marked with `async`, and it is also expected to possibly throw an error (hence it is marked with `throws`), as illustrated in Figure 6.23.

```
protocol IncreaseUserScoreUseCase {
    func execute(_ scoreIncrease: Int) async throws
}
```

Figure 6.23: IncreaseScoreUseCase Protocol

Upon defining the protocol for the use case, it is necessary to implement the business logic that the use case will execute. In this instance, the logic is straightforward: it simply needs to update the `score` property of the current user. This update, involving a backend call, will be executed using a repository, specifically the `UserRepository`. Consequently, it is now necessary to add a new method for increasing score to the repository's interface, as illustrated in Figure 6.24.

```
protocol UserRepository {
    ...
    // Add the new method for increasing the score
    func increaseScore(by score: Int) async throws
    ...
}
```

Figure 6.24: Update of UserRepository Protocol

With the addition of the new repository method, the focus shifts towards its implementation. Transitioning from the domain layer, the implementation is placed in `UserRepositoryImpl` located within the data layer. This placement is consistent with the principles of Clean Architecture, which advocates for a separation of concerns to enhance maintainability and scalability.

The implementation consists of making Firebase backend calls to update the document pertaining to the current user. Specifically, these calls facilitate the incrementation of the user's score, an essential feature for user engagement and tracking progress. This process of updating and managing user data through Firebase is detailed in Figure 6.25, which provides a code snippet to understand the sequence of operations.

```
import Firebase

struct UserRepositoryImpl: UserRepository {
    ...
    // Add implementation of the new method
    func increaseScore(by score: Int) async throws {
        // Create reference to Firestore
        let db = Firestore.firestore()

        // Get current Firebase User
        let firUser = getFirUser()

        // Create reference to users collection
        let usersCollectionRef = db.collection(
            Constants.FIREBASE_COLLECTION_USERS
        )

        // Fetch the document for current user
        let query = usersCollectionRef.whereField(
            "uid", isEqualTo: firUser.uid
        )

        let querySnapshot = try await query.getDocuments()

        // There must only be one user with that uid
        guard let userDoc = querySnapshot.documents.first else {
            throw UserError.duplicateUser
        }

        // Decode the user data and increase score
        var user = try userDoc.data(as: User.self)
        user.score += score

        // Encode the User and update remote document
        let userData = try Firestore.Encoder().encode(user)
        try await userDoc.setData(userData)
    }
    ...
}
```

Figure 6.25: Update of UserRepository Implementation

Once the new repository method is implemented, it can be invoked in the implementation of the new use case. This is illustrated in Figure 6.26. This completes the first part of the implementation, resulting in defined models and use cases that support the described functionality.

```
struct IncreaseScoreUseCaseImpl: IncreaseScoreUseCase {
    private let userRepository: UserRepository

    init(userRepository: UserRepository) {
        self.userRepository = userRepository
    }

    func execute(_ scoreIncrease: Int) async throws {
        try await userRepository.increaseScore(by: scoreIncrease)
    }
}
```

Figure 6.26: Update of UserRepository Implementation

Now, the repository and use case need to be registered with the Resolver framework to enable the injection of the use case into a view model. The registration process is detailed in Figure 6.27. It is also worth noting that the Resolver's registration functions are called in `AppDelegate` upon the app's launch.

```
extension Resolver {
    static func registerRepositories() {
        // Register the repository
        register {
            UserRepositoryImpl(
                authenticationRepository: resolve()
            ) as UserRepository
        }
    }

    static func registerUseCases() {
        // Register the use case
        register {
            IncreaseScoreUseCaseImpl(
                userRepository: resolve()
            ) as IncreaseScoreUseCase
        }
    }
}
```

Figure 6.27: UserRepository and IncreaseScoreUseCase Registration

After registering the new repository and use case, it is possible to proceed to the presentation layer to create a new view and its corresponding view model. This new view will also need to be added to the view hierarchy as described in Section 6.2.4. The implementation of the new view model is shown in Figure 6.28.

```
import Resolver
import SwiftUI

final class UserScoreViewModel: ObservableObject {

    @Published var state = State()
    // Use case injection
    @Injected
    private var increaseScoreUseCase: IncreaseScoreUseCase

    // Encapsulated view state
    struct State {
        var alert: AlertData?
        var isLoading = false
        var scoreIncrease = 5
    }

    func increaseScore() {
        Task {
            // Loading state handling
            defer { state.isLoading = false }
            state.isLoading = true

            do {
                // Call the use case and increase score
                try await increaseScoreUseCase.execute(scoreIncrease)
            } catch {
                // Show alert in case of an error
                state.alert = AlertData(
                    title: "Error Increasing Score"
                )
            }
        }
    }
}
```

Figure 6.28: UserScoreViewModel Implementation

This view model can then be used to manage the state of a view, such as in the example `UserScoreView`, detailed in Figure 6.29. With the implementation of the view, the entire process of implementation of increasing a user's

```

import SwiftUI

struct UserScoreView: View {
    @ObservedObject var viewModel: UserScoreViewModel

    var body: some View {
        VStack {
            Text("Increase Score By")
            TextField(
                "Score",
                value: viewModel.state.scoreIncrease
            )
            Button("Increase Score") {
                viewModel.increaseScore()
            }
        }
        .padding()
    }
}

```

Figure 6.29: UserScoreView Implementation

score is completed. The view provided is merely an example, but it effectively illustrates the overall workflow. In the following sections, changes will be committed, reviewed, and merged according to the process that has been applied across all modules of the Memorizify application.

### 6.3.3 Committing, Reviewing, Merging and Deployment

After implementing the changes, it is necessary to update the version and build number. These updates must be committed and pushed to the BitBucket repository to ensure that they are securely stored in version control. *Commit messages* should be clear and descriptive, helping to understand and assess the changes effectively. *Pushing* these commits to BitBucket updates the remote repository, thereby backing up the changes and making them accessible from other devices.

Once the implementation is complete and the pushed changes are available on the server, a pull request (PR) can be initiated using the BitBucket web UI. This interface allows to name the PR, provide a comprehensive description—Memorizify requires a list of all changes—and select the target merge branch, typically the `develop` branch. In this stage, it is also possible to add reviewers and attach commentaries, attachments, and more. After the PR details are entered, it is formally created.

The PR then undergoes a *review* process. If there are any flaws identified, comments are added to the PR, and the issues must be resolved. Once the PR is deemed flawless, it can be *merged* using a *squash merge* to maintain a tidy `develop` branch. After merging, a *git tag* corresponding to the version is added

## 6. IMPLEMENTATION

to the repository. A link to this tag is also included as a comment in the merged PR, as illustrated in Figure 6.30. Memorizify consistently follows this process for all application changes, ensuring robust trackability and maintaining a clean history.

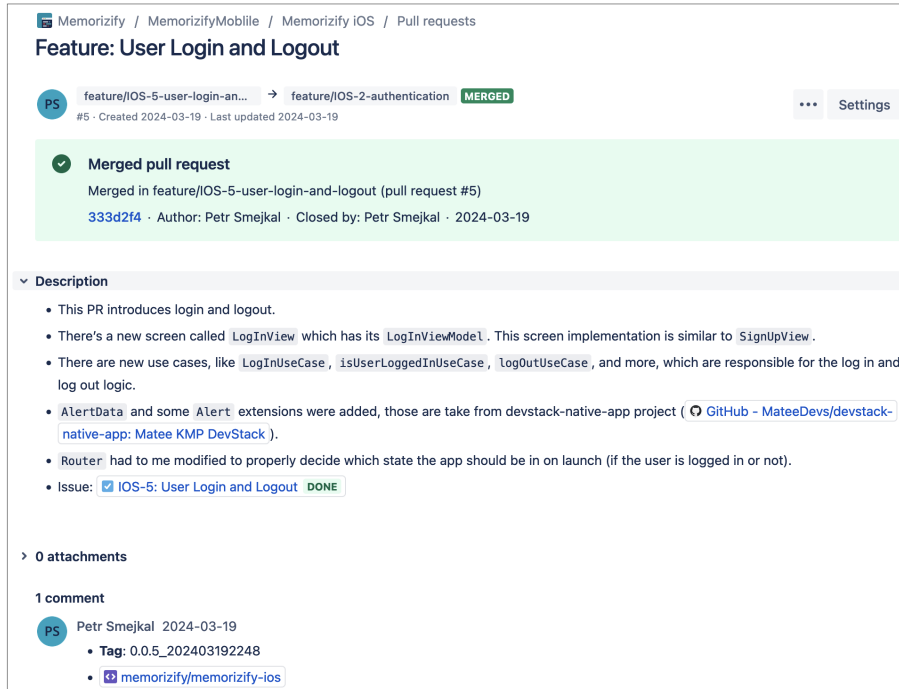


Figure 6.30: Merged BitBucket Pull Request

To finalize the workflow, the build of the new version is deployed to TestFlight from Xcode, an essential step for testing applications on iOS devices before their final release. The deployment process begins by creating a deployable build, which necessitates archiving the current state of the application. This is accomplished through the *Product* menu in Xcode, where selecting the *Archive* option initiates the compilation and packaging of the application into a single, distributable entity known as an archive.

Archiving is more than just compressing the files; it involves compiling the application's code, linking required libraries, and embedding assets in a way that conforms to Apple's requirements for distribution. This process ensures that the build is stable and encapsulates all necessary configurations and resources. Once the build is successfully archived, Xcode automatically opens the *Archive Manager*, a tool that lists all the archived versions of applications. From the *Archive Manager*, it is possible to select the appropriate build and upload it directly to TestFlight. This step is crucial as it allows for the beta testing of the application in a real-world environment, which is invaluable for identifying and addressing potential issues before the public release. The process of uploading the build of the Memorizify application to TestFlight is illustrated in Figure 6.31.

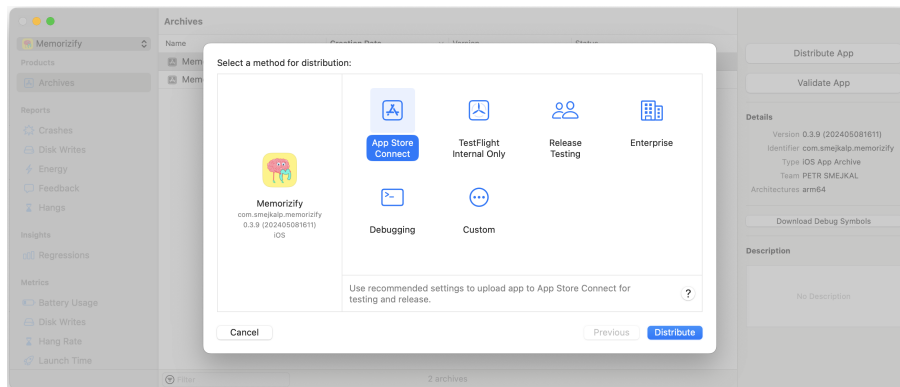


Figure 6.31: Testflight Distribution

Upon completion of the distribution upload, the build is processed, while its status can be monitored in AppStore Connect. When the processing is finished, testers are notified via email and can download the new build to their devices to begin testing.

This completes the workflow, which has been employed for all modules implemented in the Memorizify app, achieving excellent history trackability and a clean process. This workflow is based on the best practices of Software Engineering.

## 6.4 Storylines Module

The *Storyline* module has specific implementation characteristics that are worth noting and are described in this section. The structure of the **Storyline** model is derived from the domain model introduced in Section 3.2.2. This means that a **Storyline** has a **StorylineKind**. Moreover, each **StorylineKind** has **StorylineData** consisting of multiple **StorylinePage**. Consequently, when the **Storyline** is uploaded to the backend, only key (id) of the **StorylineKind** is sent. Conversely, when it is received, the **Storyline** must be locally reconstructed starting from the **StorylineKind** entity. This is an effective way to handle the **Storyline** model.

This behavior is facilitated by using a custom encoder and decoder (ideally via a Swift *extension*) for the **StorylineKind** model. The key is to avoid encoding the **StorylineKind**, **StorylineData**, and **StorylinePage** entities recursively but rather save the kind identifier as a **String**. Conversely, when a **Storyline** is decoded, it constructs the **StorylineKind**, **StorylineData**, and all **StorylinePage** entities based on the single **String** key. An outline of the implementation of the custom encoder and decoder for the **Storyline** entity is shown in Figure 6.32.

```
struct Storyline: Codable {
    ...
    // Custom encoding
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(
            keyedBy: CodingKeys.self
        )
        // Just encode the rawValue (identifier)
        try container.encode(kind.rawValue, forKey: .kind)
        ...
    }

    // Custom decoding
    init(from decoder: Decoder) throws {
        let container = try decoder.container(
            keyedBy: CodingKeys.self
        )
        // Get the key identifier
        let kindRawValue = try container.decode(
            StorylineKind.RawValue.self,
            forKey: .kind
        )
        // Attempt to initialize the given StorylineKind
        guard let kind = StorylineKind(
            rawValue: kindRawValue
        ) else {
            throw DecodingError.dataCorruptedError(
                forKey: .kind,
                in: container,
                debugDescription: "Invalid StorylineKind value"
            )
        }
        self.kind = kind
    }
    ...
}
```

Figure 6.32: Storyline Custom Encoding and Decoding

## 6.5 Other Modules

The workflow described earlier in this chapter is employed across all modules in the Memorizify application, including key modules such as **Authentication**, **Home**, **Storylines**, **Board**, **Guilds**, **Settings**, and others. This is further supported by Figure 6.33 and Figure 6.34.



## 6.6. Summary of Implementation

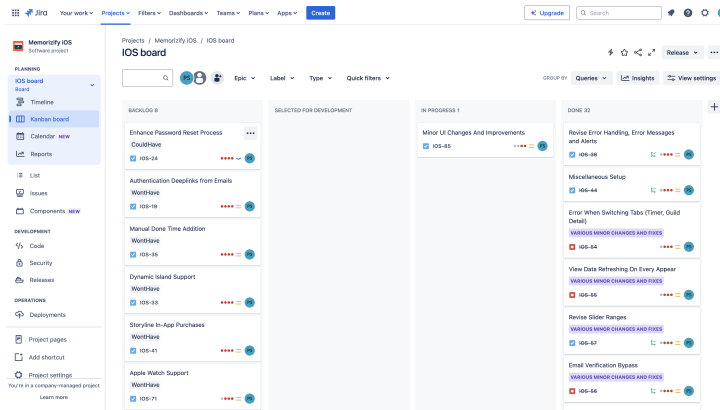


Figure 6.33: Memorizify Jira Board Overview

Summary	Created	Activity	Reviewers
Other: 3 Storylines With Assets <a href="#">IOS-81-3-storylines-with-as...</a> → develop Petr Smejkal · #52, updated 2024-04-28	7 days ago	1	No reviewers
Other: Developer Documentation <a href="#">IOS-53-developer-document...</a> → develop Petr Smejkal · #51, updated 2024-04-28	11 days ago	1	No reviewers
Other: UI Testing <a href="#">IOS-26-ui-testing</a> → develop Petr Smejkal · #50, updated 2024-04-27	11 days ago	1	No reviewers
Other: Unit Testing <a href="#">IOS-25-unit-testing</a> → develop Petr Smejkal · #59, updated 2024-04-27	11 days ago	1	No reviewers
Feature: Account Deletion <a href="#">feature/IOS-83-account-dete...</a> → develop Petr Smejkal · #58, updated 2024-04-27	11 days ago	1	No reviewers
Other: Privacy Policy <a href="#">IOS-82-privacy-policy</a> → develop Petr Smejkal · #57, updated 2024-04-27	11 days ago	1	No reviewers
Other: Add NSLogging For Networking <a href="#">IOS-34-add-nslogging-for-n...</a> → develop Petr Smejkal · #56, updated 2024-04-26	12 days ago	1	No reviewers
Other: Update User Data Only Username <a href="#">IOS-77-update-user-data-on...</a> → develop Petr Smejkal · #55, updated 2024-04-26	12 days ago	1	No reviewers
Epic: Various Minor Changes and Fixes <a href="#">IOS-42-various-minor-chang...</a> → develop Petr Smejkal · #54, updated 2024-04-26	13 days ago	No activity	No reviewers
Bugfix: Guild Data Updates <a href="#">bugfix/IOS-79-guild-data-up...</a> → <a href="#">IOS-42-various-minor-chang...</a> Petr Smejkal · #53, updated 2024-04-26	13 days ago	1	No reviewers
Bugfix: Email, Nickname And Password Validation <a href="#">bugfix/IOS-80-email-nickna...</a> → <a href="#">IOS-42-various-minor-chang...</a> Petr Smejkal · #52, updated 2024-04-26	13 days ago	1	No reviewers
Other: Revise Background Assets <a href="#">IOS-76-revise-background-4...</a> → <a href="#">IOS-42-various-minor-chang...</a> Petr Smejkal · #51, updated 2024-04-25	13 days ago	1	No reviewers
Other: No Connection View State <a href="#">IOS-74-no-connection-view...</a> → <a href="#">IOS-42-various-minor-chang...</a> Petr Smejkal · #50, updated 2024-04-25	13 days ago	1	No reviewers

Figure 6.34: Memorizify BitBucket PRs Overview

## 6.6 Summary of Implementation

This chapter begins by discussing the setup of the development environment, then moves on to describe the implementation and usage of fundamental components, followed by an example showcasing a typical workflow employed during the development of all Memorizify modules. The chapter also includes a detailed description of a particularly interesting part of the code and concludes with a brief section on other modules.

All requirements categorized as *Must Have* and *Should Have* have been satisfied, and most of the *Could Have* requirements were also implemented. The initial goal was to create a prototype of a productivity-enhancing application; however, the outcome is a production-ready mobile application, thereby exceeding the objectives set for this thesis. Screenshots of the application’s interface are included in Appendix D.

## 6. IMPLEMENTATION

---

In total, over 13,000 lines of code were written in Swift files during the development of the Memorizify application. Additionally, more than 90 Jira issues were created, over 60 PRs were merged, among other achievements. Overall, the result is a well-implemented mobile application, with the adoption of best practices and standard processes of Software Engineering throughout the development.

---

## Testing and Documentation

Testing and documentation phases are both integral components of the thesis assignment, detailed in this chapter. Following the implementation discussed in the previous chapter, these phases are conducted to ensure the robustness and usability of the developed software. Testing is aimed at verifying the functionality and performance of the application through various methodologies, while documentation provides clear and useful guides for future developers and users. Together, these steps are essential for refining and detailing the work completed, underscoring their importance in the overall software development lifecycle as prescribed by this thesis.

### 7.1 Testing

Both testing during the main implementation phase and subsequent refinements to the final product are important elements of software development. This section offers a comprehensive overview of all testing activities conducted throughout this thesis. It starts with *unit testing*, aimed at identifying early bugs during the development process. This is followed by *UI testing*, which addresses any issues arising from changes in the user interface—changes anticipated and adjusted for based on insights from *usability testing*. The discussion concludes with *performance testing*, which evaluates the app's behavior under various network conditions and assesses its launch performance.

#### 7.1.1 Unit Testing

*Unit testing* is a fundamental aspect of software development that enables developers to catch logical errors early in the development process. This approach involves decomposing application functionality into testable units and writing tests to assess these units under various input conditions. Typically, the unit size corresponds to a method, meaning that a unit test validates the correct behavior of a method. [123]

##### 7.1.1.1 Characteristics of Unit Testing

Unit testing is a fundamental testing method that can be beneficial in identifying bugs early in the development cycle, providing developers with sufficient

time to fix these bugs before the product is released. Since unit tests isolate a small portion of code functionality, they are reliable as they focus on very basic logic. However, this is also a limitation of unit tests—they cannot detect more complex systematic issues. In such cases, a different type of test, such as integration testing, is required. [123]

Unit testing typically involves three key phases: *Arrange*, *Act*, and *Assert*.

- In the *Arrange* phase, the testing environment is set up, and the necessary objects are initialized to prepare for the test.
- The *Act* phase involves executing the specific functionality being tested, usually by calling a method or a function with the prepared objects.
- Finally, the *Assert* phase checks the outcomes of the action against expected results to verify that the code behaves as intended. [123]

This structured approach helps ensure that each unit test is clear and maintains a consistent methodology for verifying software functionality. [123]

In addition to the three phases of unit testing that define how a unit test should behave, there are several methods to evaluate the quality of unit tests. One such method is A TRIP, an acronym standing for *Automatic, Thorough, Repeatable, Independent, and Professional*. These criteria help ensure that each test is robust, reliable, and effective in maintaining high software quality. [124]

- *Automatic* tests are executed without manual intervention, facilitating continuous integration processes.
- *Thorough* testing means that all conceivable scenarios, including edge cases, are considered and tested.
- *Repeatable* tests ensure that running the tests repeatedly in the same environment produces the same results, demonstrating reliability and stability.
- *Independent* tests verify that each test can run alone and in any order, not affecting the outcomes of other tests.
- Finally, *Professional* tests are well-documented, clearly written, and maintained to uphold standards over time, making them useful not just for current development but for future modifications as well. [124]

These principles help maintain the integrity and utility of the testing process, contributing to the overall robustness of the software product. [124]

### 7.1.1.2 Mocking

*Mocking* in unit testing involves substituting real dependencies with mock objects to isolate software system components. These mock objects simulate the behavior of actual components and concentrate on verifying interactions rather than states, thereby enhancing testability and design modularity. Often, unit testing cannot be conducted with real components, making mocks essential for feasible testing. [125]

```
struct OnboardingRepositoryImpl: OnboardingRepository {
    private let keychainProvider: KeychainProvider

    init(keychainProvider: KeychainProvider) {
        self.keychainProvider = keychainProvider
    }

    // Saves the information that the user
    // has seen the onboarding.
    func saveHasUserSeenOnboarding() throws {
        try keychainProvider.add(
            .hasUserSeenOnboarding, value: "true"
        )
    }

    // Loads the information about whether
    // the user has seen the onboarding.
    func loadHasUserSeenOnboarding() throws -> String {
        return try keychainProvider.read(.hasUserSeenOnboarding)
    }
}
```

Figure 7.1: Implementation of `OnboardingRepository`

On the other hand, mocking closely binds tests to specific implementations, which can complicate test maintenance when behaviors change. Despite these challenges, mocking continues to be an important aspect of unit testing, supported by various frameworks that ensure that components functioning independently behave as expected. [125]

### 7.1.1.3 Unit Testing Example

This section presents a unit testing example that employs the methods and concepts previously introduced. The tests are conducted using the *XCTest* framework, a suite of testing tools included with the iOS SDK. For simplicity, the example focuses on a straightforward repository named `OnboardingRepository`, which manages *Keychain* data stored in secure storage. The implementation of the repository is illustrated in Figure 7.1.

From Figure 7.1, it is evident that the `OnboardingRepositoryImpl` strictly adheres to the `OnboardingRepository` protocol and maintains a single dependency, the `KeychainProvider`. To ensure rigorous and accurate unit testing, it is essential to isolate the behavior of the repository. This isolation is crucial for verifying that the repository interacts correctly with its dependencies without unintended side effects. Fortunately, the extensive use of dependency injection throughout the Memorizify app greatly facilitates this process. For instance, by introducing a `KeychainProviderMock`, which also conforms to the `KeychainProvider` protocol, as depicted in Figure 7.2.

```
struct KeychainProviderMock: KeychainProvider {  
  
    private var data: [String: String] = [:]  
  
    // Adds value to the secure storage  
    func add(_ key: KeychainKey, value: String) throws {  
        data[key.rawValue] = value  
    }  
  
    // Adds loads value at key from the secure storage  
    func read(_ key: KeychainKey) throws -> String {  
        guard let value = data[key.rawValue] else {  
            throw KeychainError.valueForKeyNotFound  
        }  
        return value  
    }  
  
    // Removes value at the key  
    func remove(_ key: KeychainKey) throws {  
        data[key.rawValue] = nil  
    }  
  
    // Removes all values in Keychain  
    func removeAll() throws {  
        data.removeAll()  
    }  
  
    // Removes all values except those at keys in 'values'  
    func removeAll(except values: [KeychainKey]) throws {  
        let keysToRemove = data.keys.filter {  
            !values.contains(  
                KeychainKey(rawValue: $0)!  
            )  
        }  
        for key in keysToRemove {  
            data[key] = nil  
        }  
    }  
}
```

Figure 7.2: Mock of KeychainProvider

```

import XCTest
@testable import Memorizify

final class OnboardingRepositoryTests: XCTestCase {
    var repository: OnboardingRepositoryImpl!
    var mockKeychainProvider: KeychainProviderMock!

    // Sets up necessary objects before each test runs
    override func setUpWithError() throws {
        mockKeychainProvider = KeychainProviderMock()
        repository = OnboardingRepositoryImpl(
            keychainProvider: mockKeychainProvider
        )
    }

    // Tests saving the onboarding status to the keychain.
    func testSaveHasUserSeenOnboarding() {
        // Verifies no exceptions are thrown when saving
        XCTAssertNoThrow(
            try repository.saveHasUserSeenOnboarding()
        )

        do {
            // Attempts to read the saved value
            let value = try mockKeychainProvider.read(
                .hasUserSeenOnboarding
            )
            XCTAssertEqual(value, "true") // Asserts the value
        } catch { XCTFail("Unexpected error: \(error)") }
    }

    // Other tests such as loading value will follow here
}

```

Figure 7.3: Unit Testing of OnboardingRepository

Once the dependency is mocked, unit testing can commence. The tests are conducted using the previously mentioned Xcode XCTest framework within a `MemorizifyTests` target, a separate target in the Xcode project. This separation is standard practice in iOS development, ensuring a clear distinction between the app's codebase and its tests. As depicted in Figure 7.3, to unit test the repository, the repository instance and its dependency mocks are first prepared within the `setUpWithError` function, followed by the unit tests themselves, which conclude this section. Tests such as the one showcased cover the key functionality of the Memorizify application.

### 7.1.2 User Interface Testing

*UI testing* focuses on verifying the user interface of an application by simulating interactions with UI elements through tests. This type of testing is essential for ensuring that the UI functions correctly and to identify any regressions that may occur from changes in UI-related code. Apple supports this testing approach with the XCTest framework, which utilizes accessibility data to interact with on-screen elements. Similar to unit testing, it is possible to create UI test cases by adding methods that interact with UI components. These methods include code that manipulates UI elements and uses assertions to confirm that the outcomes match expected result. [126] An example of code of a UI test is illustrated in Figure 7.4, while a video demonstrating a UI test suite that tests various elements in the authentication section of Memorizify is included in the electronic attachments detailed in Appendix G.

One powerful feature of modern UI testing frameworks including XCTest, is the ability to record interactions within a simulator and convert these actions into executable test code. This capability allows developers to interact with the application's UI as a user would, with the framework capturing each click, scroll, or input. After the recording session, the framework translates these interactions into a set of instructions that can be run as a UI test. This method not only speeds up the test creation process but also ensures that the tests accurately reflect real user behaviors, making it an invaluable tool for enhancing test coverage and reliability. [126]

The Memorizify app ensures that all primary components of the application are at least partially covered by UI tests, which enhances the overall robustness of the application.

### 7.1.3 Usability Testing

*Usability testing* is a method that assesses how easily real users can interact with an app by having them complete specific tasks while being observed. Either in person or remotely, this form of testing aims to identify confusing areas and pain points within the user journey, thereby pinpointing opportunities for enhancing the overall user experience. The primary focus of usability testing is to measure the product's practical functionality, particularly in terms of how effectively and efficiently users can achieve predefined goals. This testing is essential for understanding user interaction patterns and improving interface design to meet user needs more effectively. [127]

Usability testing offers several advantages, including the ability to test products with real users before deployment. This preemptive feedback can enhance usability and satisfaction by allowing developers to observe real user behaviors and preferences, thereby supporting a user-centered design approach. However, usability testing also has drawbacks. It can be time-consuming and costly, especially if it requires specialized facilities. Furthermore, the results can be subjective and depend on the participant selection, potentially leading to data that does not accurately represent the entire target user base and may not fully capture broader user needs or experiences. [127]



```
import XCTest

final class AuthenticationUITests: XCTestCase {

    override func setUpWithError() throws {
        continueAfterFailure = false
    }

    func testResetPassword() {
        let app = XCUIApplication()
        app.launch() // Launch the app

        // Tap the "Log In" button
        app.buttons["Log In"].tap()

        let scrollViewsQuery = app.scrollViews
        // Find a "Reset Password" button in the scroll view
        scrollViewsQuery
            .otherElements
            .buttons["Reset it now!"].tap()

        // Find the text field to fill in email
        // and type in the email
        scrollViewsQuery
            .otherElements
            .containing(
                .staticText,
                identifier: "Please, fill in your email"
            )
            .children(matching: .textField)
            .element.typeText("pesmejkal@post.cz")

        // Tap on "Reset Password" and then tap "Back" button
        app.buttons["Reset Password"].tap()
        app.buttons["Back"].tap()
    }
}
```

Figure 7.4: UI Testing of Reset Password Feature

### 7.1.3.1 Usability Testing Scenarios

*Usability testing scenarios* effectively translate user goals into actionable tasks, allowing testers to observe real users navigating an app to complete specific activities. These tasks are designed to reflect actual user behavior, helping identify usability issues and measure user efficiency and satisfaction. The aim is to create engaging and realistic scenarios without overly directing the user, thereby preserving the integrity of their natural interactions. Testing scenarios often originate from the application's use cases. [128]

Creating effective scenarios is crucial for usability testing. The scenarios should guide users without dictating every step, allowing them to explore the interface independently. For example, instructing a user to "find a product within a specific budget" rather than "click here, then there" helps collect data on true user behavior rather than their ability to follow instructions. This approach ensures that usability tests genuinely reflect an interface's ease of use and guide improvements in intuitive design. [128]

The Memorizify application underwent usability testing with seven testers across multiple sessions. The group included four males and three females, ranging in age from 19 to 53. Among the testers, five were university students, one had completed higher education, and another held a university degree.

Each testing scenario was structured, starting with specific prerequisites to effectively set the stage. For example, if the scenario involved changing the app language to English, the initial setting would be a different language. The preparatory phase also ensured that each scenario provided sufficient context to align the testers with its objectives. Then, the task of the scenario was outlined, and the tester was given free rein to complete it. Furthermore, each scenario outlined the expected steps that testers might take to successfully achieve the scenario's goals. These expected actions were not disclosed to the testers but were used to compare their actual actions with the expected ones. Detailed descriptions of these scenarios are available in Appendix E. These details follow the format specified in Figure 7.5.

**Identifier:** Identifier of the scenario.

**Name:** Name of the scenario.

**Prerequisites:** Prerequisites that must be fulfilled before starting the test scenario.

**Context:** Context provided for a better understanding of the situation.

**Success:** Definition of what constitutes success in the scenario.

**Expected Steps:** List of steps that the user is expected to perform in order to achieve the scenario's goal.

Figure 7.5: Usability Test Scenario Format

### 7.1.3.2 Changes Made Based On Feedback

Based on the outcomes of the usability testing, several modifications were implemented to enhance the user experience and interface design of the application. Notably, button placements were adjusted to the bottom of the screen, rather than above forms, to improve accessibility and ease of reach for users. This change is depicted in Figure 7.6. Moreover, flat pull-down indicators

were added to the bottom sheets throughout the application to make dismissing the sheet easier and more intuitive. Additionally, scroll views were incorporated into forms to better accommodate smaller devices; this adjustment ensures that when a keyboard appears during text entry, the text fields remain visible and accessible, preventing them from being obscured. Lastly, various other UI details were refined to enhance overall aesthetic appeal and functionality.

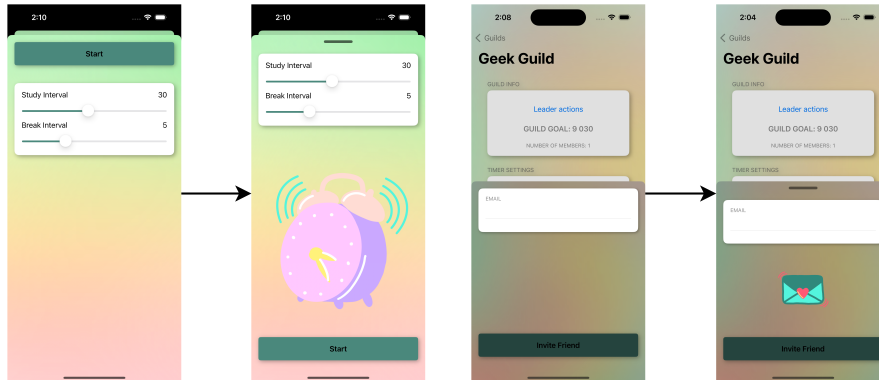


Figure 7.6: Implemented Usability Testing Changes

### 7.1.4 Performance Testing

Xcode, in combination with XCTest, offers powerful features for testing various metrics including CPU, GPU, and memory performance. These metrics can be assessed under different conditions, while the app is running, as well as under simulated conditions such as an overheated device, various GPU states, and more [129]. For the Memorizify app, launch performance and network performance were tested. It's worth noting that this text does not use any standardized measurements for this kind of testing, therefore, the results are more subjective in nature.

#### 7.1.4.1 Launch Performance Testing

Testing *launch performance* in Xcode is important for optimizing the initial responsiveness of the application. XCTest provides several metrics to measure app launch performance [130]. For Memorizify, launch tests were conducted, specifically, the duration (`XCTApplicationLaunchMetric`), CPU utilization (`XCTCPUMetric`), memory usage (`XCTMemoryMetric`), and storage usage (`XCTStorageMetric`) were measured. These tests were performed 20 times across various devices. The results are detailed in Table 7.1.

As mentioned in the introduction of this section, those tests do not follow any specification for evaluation of the results, hence the results are kind of subjective. Anyways, the application feels fast during launch, which is supported by the average launch duration of 1.142 seconds. The CPU utilization is also very low, with only 0.043 seconds of total CPU time during launch.

## 7. TESTING AND DOCUMENTATION

LAUNCH METRIC [unit]	AVERAGE VALUE
Duration [seconds]	1.142s
CPU Utilization [cycles, instructions retired, seconds]	98436.276 kC, 126709.696 kI, 0.043s
Memory Usage [kB peak, kB]	25994.445 kB, 9.830 kB
Storage Usage [kB]	0.000 kB

Table 7.1: Results for the Launch Performance Testing

The memory usage is also low, with only 9.8 MB on average, which is very few since the least memory capacity on an iOS that supports iOS 17 is 4GB. There's no storage usage recorded during launch, which makes sense, because the application manipulates storage only later (not during launch). Overall, the app subjectively feels very responsive and swift during launch and it should satisfy all users' needs in this regard.

### 7.1.5 Network Performance Testing

Another type of performance testing conducted for Memorizify is *network testing*. When a real device is connected, Xcode offers options to throttle the network based on defined presets, such as EDGE, 3G, 4G, among others (see Figure 7.7). This testing is useful for evaluating how the application behaves under various network conditions.

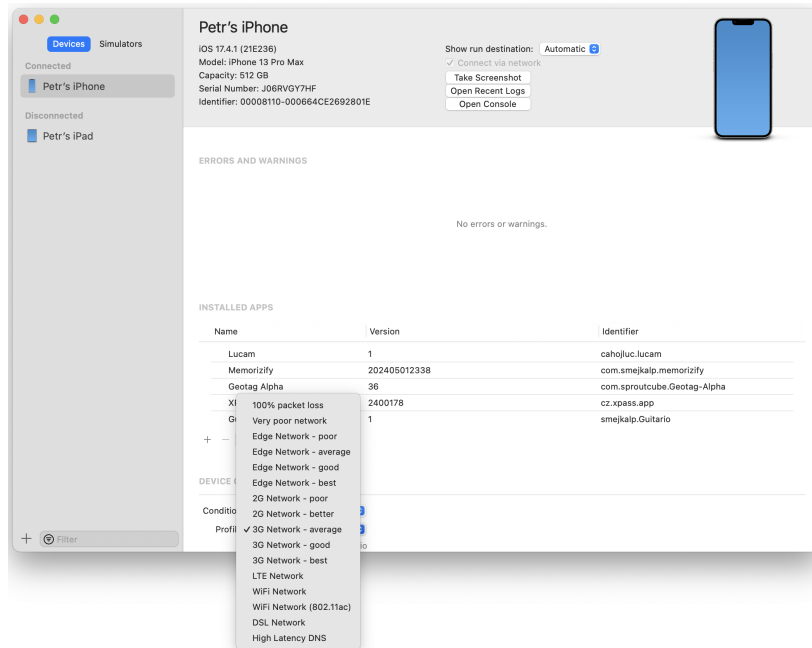


Figure 7.7: Xcode Network Performance Presets

For Memorizify, presets testing several network conditions were used. Specifically, these were *100% packet loss*, *Edge Network - average*, *3G Network - av-*

*erage*, and *LTE Network*. Memorizify is designed to handle different network states, always keeping the user informed about the current situation during network requests.

For the 100% packet loss preset, a No Internet Connection sign is displayed over the view. If the user attempts to pull down to refresh, the connection state is checked and updated accordingly in the view. For the Edge Network - average, the application takes a significant amount of time to load; however, the views usually load eventually, and if they don't, an error message is displayed. During the loading time, a view with pulsing placeholders is shown. Lastly, for both 3G Network - average and LTE Network, the application behaves normally, with no significant delays in loading, with the LTE Network being slightly faster. To view all network states in Memorizify, see Figure 7.8. Overall, the app demonstrates excellent handling of network states and loading processes, contributing to a great user experience.

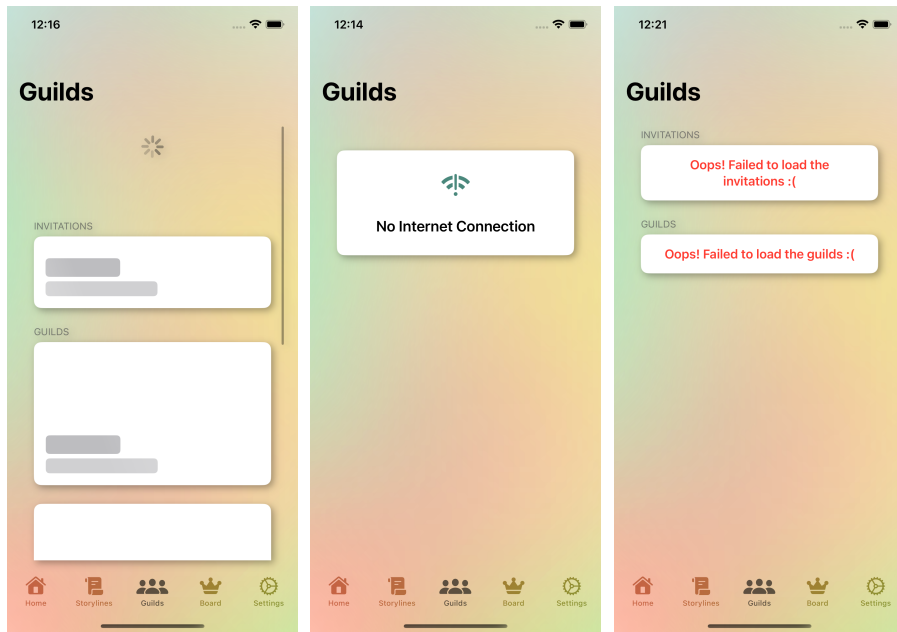


Figure 7.8: Memorizify Network Conditions Handling

### 7.1.6 Summary of Testing

The testing phase of this chapter involved various methods, starting with unit testing conducted during implementation. Unit tests were employed to identify bugs as early as possible in the development process. As the UI of the application was implemented, UI tests were introduced to ensure functionality during iterative changes. Additionally, Memorizify underwent usability testing with seven testers, whose valuable feedback contributed to subsequent usability enhancements. Performance testing was also conducted, assessing both launch performance and the application's ability to handle various network conditions. Overall, the application was thoroughly tested according to standard processes in Software Engineering, thus fulfilling one of the thesis' objectives.

## 7.2 Documentation

*Documentation* is an important yet often underemphasized part of the software development lifecycle. The documentation will be particularly useful in the future when the application undergoes further development, allowing other developers to easily join the project and understand the logic of the codebase. The following sections describe the specific tools and strategies used to document Memorizify.

### 7.2.1 Code Commentary Format

There is a code commentary format recommended by Apple that facilitates the use of other Xcode tools. This format, which is similar to those used in other languages, allows for comment annotations for classes, functions, and variables, as depicted in Figure 7.9. While additional inline comments can be added anywhere in the code, they will not be utilized by Xcode documentation tools. [131] This format is consistently used throughout Memorizify.

```
1
2 /// Description
3 class MyClass {
4
5     /// Description
6     let someNumber: Int
7
8     /// Description
9     var someText: String
10
11    /// Description
12    /// - Parameters:
13    ///   - someNumber: someNumber description
14    ///   - someText: someText description
15    init(someNumber: Int, someText: String) {
16        self.someNumber = someNumber
17        self.someText = someText
18    }
19
20    /// Description
21    /// - Parameters:
22    ///   - argNumber: argNumber description
23    ///   - argText: argText description
24    ///   - argDate: argDate description
25    func myMethod(argNumber: Int, argText: String, argDate: Date) {
26        // implementation goes here
27    }
28 }
```

Figure 7.9: Xcode Code Comment Format

### 7.2.2 DocC

*DocC* is a documentation compiler specifically designed for Swift frameworks and packages, transforming Markdown-based text into comprehensive API documentation and interactive tutorials. It enables developers to preview their

work in real-time. Using a custom version of Markdown known as documentation markup, DocC incorporates features such as cross-symbol linking and code listings, allowing developers to integrate detailed documentation directly into their source code. Additionally, it supports the creation of a documentation archive from comments within a package, which can be exported from Xcode and either browsed on a Mac or hosted as a website. [132]

Memorizify utilizes this tool throughout the entire codebase, documenting the code thoroughly. This allows developers to quickly view documentation on all classes, methods, and properties, as shown in Figure 7.10. The documentation is also available as a DocC Archive, which is included in the electronic attachments, described in Appendix G.

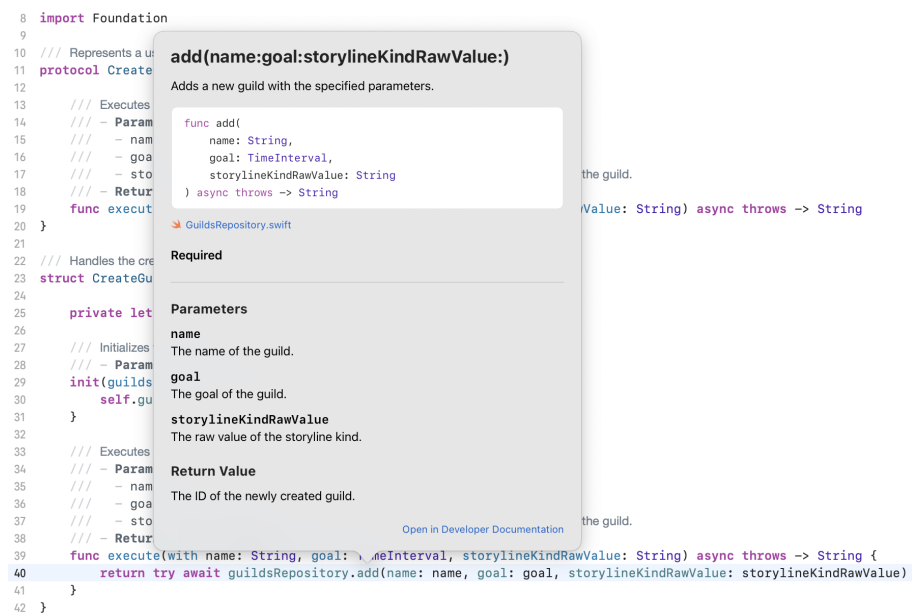


Figure 7.10: Xcode DocC Referencing

### 7.2.3 Summary of Documentation

As outlined in the thesis assignment, one of the objectives is comprehensive documentation. The documentation for Memorizify was meticulously carried out using Apple's DocC documentation tool, resulting in an interactive archive that is included as part of the electronic attachments. With this, the thesis successfully achieves its stated documentation objective.





---

## Results and Future Development

The results of the thesis and future development are outlined in this final chapter, which completes the fulfillment of all the thesis goals.

### 8.1 Results

This section provides a retrospective evaluation of the decisions made during the analysis and design phases, which can only be effectively assessed after implementation. It then evaluates the implementation and its fulfillment of the non-functional and functional requirements specified during the domain analysis.

#### 8.1.1 Analysis, Design, Implementation and Testing

An in-depth analysis of learning approaches and cognitive processes conducted in this thesis provided substantial insights into designing an interactive productivity enhancing iOS application. It identified and detailed the learning methods that are most effective and appropriate for such tools, ensuring that the application's design is both scientifically grounded and practically relevant. Additionally, domain analysis and evaluation of similar solutions revealed pitfalls and flaws in existing applications, guiding the avoidance of these issues while highlighting user-preferred features. Based on these comprehensive findings, the necessary requirements and use cases were specified.

Subsequently, in the technology analysis, *Clean Architecture* was chosen as the choice for the architecture together with the *MVVM* pattern. This choice proved to be very beneficial, as it provided a robust framework for maintaining separation of concerns, enhancing the modularity and scalability of the application. This architectural decision facilitated easier testing and modifications, allowing for more efficient iteration and improvements throughout the development process. However, the only downside of this choice was the increased amount of boilerplate code required. While this architectural approach improved modularity and maintainability, it also introduced more complexity in initial setup and increased the overall codebase.

The testing phase played a important role in refining the application. Thorough unit, UI, performance, and usability tests ensured seamless functionality. Usability testing was particularly valuable, providing feedback from actual

users, which informed subsequent modifications. Additionally, network performance testing was beneficial, allowing the app's behavior to be fine-tuned across different network conditions.

### 8.1.2 Requirements Fulfillment

All non-functional requirements were met, except for the *WatchOS* support, which was categorized as a *Won't Have* priority requirement. All functional requirements with *Must Have* and *Should Have* priorities were fulfilled. Regarding the *Could Have* functional requirements, 7 out of 9 were successfully implemented. The *Won't Have* priority requirements were not implemented at all.

### 8.1.3 Summary of the Results

In summary, all main and partial goals of the thesis were achieved by conducting thorough analyses of learning approaches and cognitive processes, exploring the domain of interactive productivity-enhancing applications, specifying requirements and use cases, and evaluating the tools used. The application was successfully implemented, meeting all *Must Have* and *Should Have* requirements, and most of the *Could Have* requirements. Additionally, the application underwent comprehensive testing which not only enhanced the application's stability and performance but also elevated it from a prototype to a production-ready product, surpassing the original thesis requirements. Finally, the codebase was meticulously documented using the DocC framework.

## 8.2 Future Development

This section offers a comprehensive outline of potential directions for future development of the application, detailing various enhancements and expansions that could enhance its functionality.

### 8.2.1 Usability Testing Feedback

The first area for potential development emerges from feedback gathered during usability testing. Users frequently expressed a desire for the application to have the capability to block notifications from selected apps, enhancing focus and reducing interruptions. Additionally, there was a common request for the ability to revisit previously unlocked storyline pages, suggesting a need for greater navigational freedom within the app. Lastly, although minor, some users suggested changes in the placement and design of certain UI elements to improve the overall user interface aesthetics and functionality. These enhancements could significantly improve user satisfaction and app usability.

### 8.2.2 Remaining Requirements

Future development could also focus on implementing features from the *Won't Have* and currently not implemented *Could Have* requirements. Adding a statistics view would allow users to monitor their learning progress, enhancing motivation. Introducing WatchOS support could increase accessibility and user

interaction, while in-app purchases for new storylines would provide a revenue stream and keep content fresh. These enhancements among other currently not implemented requirements would effectively expand the app's functionality.

### 8.2.3 Other Possible Enhancements

For further expansion, the application is partially prepared for broader infrastructure development due to its existing integration with *Firebase*. Developing a companion web platform would enhance accessibility, allowing users to engage with the application across multiple devices seamlessly. Additionally, creating a macOS version could leverage the continuity features of the Apple ecosystem, providing a more integrated experience for users who utilize multiple Apple devices. This expansion is facilitated by *Firebase*, which supports cross-platform synchronization and management.

### 8.2.4 Summary of Future Development

In summary, thanks to the robust design of the application and the numerous development ideas outlined in this section, the app is well-prepared for immediate further development and expansion. The foundational architecture, built with flexibility and scalability in mind, ensures that the application can easily incorporate additional features and thanks to *Firebase* extend across different platforms.



---

## Conclusion

The goal of this thesis was to develop an interactive productivity-enhancing mobile application for the iOS platform, adhering to the standard processes of Software Engineering. This involved analyzing the domain of productivity-enhancing mobile applications, specifying requirements and use cases, designing the application, implementing it, testing, and documenting the process. Finally, the application's results were discussed, and potential avenues for future development were outlined.

The second chapter, after stating the thesis' goals in the first chapter, provided a thorough analysis of learning approaches and cognitive processes. Additionally, a research study supporting the main application concepts was conducted in cooperation with the author's classmates.

The third chapter delved into the domain of productivity-enhancing applications, including an evaluation of current solutions and a discussion of their strengths and weaknesses.

The fourth chapter evaluated tools such as *SwiftUI* and the *Firebase* backend services to explore their capabilities, which was followed by the specification of requirements and use cases discussed in the fifth chapter. This chapter also introduced both the architectural and user interface design.

Building on insights from previous chapters, the sixth chapter detailed implementation of the application. All requirements in categorized as *Must Have* and *Should Have* were implemented, along with most *Could Have* requirements, using the tools specified in the thesis assignment.

The application underwent extensive and comprehensive testing described in the seventh chapter, which included unit tests, UI tests, usability tests, and selected performance tests. Subsequently, it was meticulously documented as part of the same chapter using the *DocC* framework.

The final chapter reviewed the thesis results and proposed future development directions for the application. Altogether, the thesis goals were met by fulfilling all main and partial goals outlined in the assignment.



---

## Bibliography

- [1] Active Learning. *Center for Educational Innovation [online]*, April 2024, [ref. 2024-04-29]. Available from: <https://cei.umn.edu/teaching-resources/active-learning>
- [2] Cloke, H. The Forgetting Curve: Why we forget and how to remember [online]. 2024, [ref. 2024-04-30]. Available from: <https://www.growthengineering.co.uk/forgetting-curve/>
- [3] What you need to know about The Curve of Forgetting [online]. 2018, [ref. 2024-04-30]. Available from: <https://medium.com/@WeAreHowDoI/what-you-need-to-know-about-the-curve-of-forgetting-dff1bb3e6d26>
- [4] Prashanti, E.; Salian, K.; et al. Cooperative learning through jigsaw classroom technique for designing cast partial dentures - a comparative study. *MedEdPublish*, volume 6, 06 2017, doi:10.15694/mep.2017.000088.
- [5] Toda, A. M.; Klock, A. C. T.; et al. Analysing gamification elements in educational environments using an existing Gamification taxonomy. *Smart Learning Environments*, volume 6, Dec 2019: p. 16, ISSN 2196-7091, doi:10.1186/s40561-019-0106-1. Available from: <https://doi.org/10.1186/s40561-019-0106-1>
- [6] Apple. Forest: Focus for Productivity [online]. 2024, [ref. 2024-05-02]. Available from: <https://apps.apple.com/us/app/forest-focus-for-productivity/id866450515>
- [7] Apple. Focus To-Do: Focus TimerTasks [online]. 2024, [ref. 2024-05-02]. Available from: <https://apps.apple.com/us/app/focus-to-do-focus-timer-tasks/>
- [8] Apple. Study Bunny: Focus Timer [online]. 2024, [ref. 2024-05-02]. Available from: <https://apps.apple.com/us/app/study-bunny-focus-timer/id1478345385>
- [9] Capka, D. Lesson 4 - UML - Domain Model [online]. 2021, [ref. 2024-05-03]. Available from: <https://www.ictdemy.com/software-design/uml/uml-domain-model>

## BIBLIOGRAPHY

---

- [10] Yeung, J. Use Case Diagram notations guide [online]. 2018, [ref. 2024-05-03]. Available from: <https://circle.visual-paradigm.com/docs/uml-and-sysml/use-case-diagram/use-case-diagram-notations-guide/>
- [11] Apple. Weak References [online]. 2024, [ref. 2024-05-04]. Available from: <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/automaticreferencecounting#Weak-References>
- [12] Apple. Unowned References [online]. 2024, [ref. 2024-05-04]. Available from: <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/automaticreferencecounting#Unowned-References>
- [13] iOS Architecture [online]. 2022, [ref. 2024-05-05]. Available from: <https://redfoxsec.com/blog/ios-architecture/>
- [14] Martin, R. C. The Clean Architecture [online]. 2012, [ref. 2024-05-05]. Available from: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- [15] Pasquier, B. How to implement MVVM pattern in Swift from scratch [online]. 2018, [ref. 2024-05-05]. Available from: <https://benoitpasquier.com/ios-swift-mvvm-pattern/>
- [16] Entity-Relationship Diagram Symbols and Notation [online]. 2024, [ref. 2024-05-05]. Available from: <https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning>
- [17] Atlassian. What are team-managed and company-managed projects? [online]. 2024, [ref. 2024-05-07]. Available from: <https://support.atlassian.com/jira-software-cloud/docs/what-are-team-managed-and-company-managed-projects/>
- [18] Schunk, D. H. *Learning theories an educational perspective*. Pearson Education, Inc, 2012, 72–73 pp.
- [19] Baulo, J.; Nabua, E. Behaviourism: Its implication to education. 12 2019.
- [20] Yilmaz, K. The Cognitive Perspective on Learning: Its Theoretical Underpinnings and Implications for Classroom Practices. *The Clearing House*, volume 84, 08 2011: pp. 204–212, doi:10.1080/00098655.2011.568989.
- [21] Gunduz, N.; Hursen, C. Constructivism in Teaching and Learning; Content Analysis Evaluation. *Procedia - Social and Behavioral Sciences*, volume 191, 2015: pp. 526–533, ISSN 1877-0428, doi: <https://doi.org/10.1016/j.sbspro.2015.04.640>, the Proceedings of 6th World Conference on educational Sciences. Available from: <https://www.sciencedirect.com/science/article/pii/S1877042815029079>



- 
- [22] Fred Paas, A. R.; Sweller, J. Cognitive Load Theory and Instructional Design: Recent Developments. *Educational Psychologist*, volume 38, no. 1, 2003: pp. 1–4, doi:10.1207/S15326985EP3801\_1, [https://doi.org/10.1207/S15326985EP3801\\_1](https://doi.org/10.1207/S15326985EP3801_1). Available from: [https://doi.org/10.1207/S15326985EP3801\\_1](https://doi.org/10.1207/S15326985EP3801_1)
- [23] Sweller, J.; Ayres, P.; et al. *Cognitive Load Theory*. Springer, 2011.
- [24] Clark, R.; Nguyen, F.; et al. Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load. *Performance Improvement*, volume 45, 10 2006, doi:10.1002/pfi.4930450920.
- [25] Wang, Y.-T.; Lin, K.-Y. Understanding Continuance Usage of Mobile Learning Applications: The Moderating Role of Habit. *Frontiers in Psychology*, volume 12, 2021, ISSN 1664-1078, doi:10.3389/fpsyg.2021.736051. Available from: <https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2021.736051>
- [26] Cepeda, N. J.; Pashler, H.; et al. Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological bulletin*, volume 132 3, 2006: pp. 80–354. Available from: <https://api.semanticscholar.org/CorpusID:18831615>
- [27] Karpicke, J. D.; Blunt, J. R. Retrieval Practice Produces More Learning than Elaborative Studying with Concept Mapping. *Science*, volume 331, no. 6018, 2011: pp. 772–775, doi:10.1126/science.1199327. Available from: <https://www.science.org/doi/abs/10.1126/science.1199327>
- [28] Terwogt, M. M.; Hoeksma, J. B. Colors and Emotions: Preferences and Combinations. *The Journal of General Psychology*, volume 122, no. 1, 1995: pp. 5–17, doi:10.1080/00221309.1995.9921217, PMID: 7714504. Available from: <https://doi.org/10.1080/00221309.1995.9921217>
- [29] Elliot, A. J. Color and psychological functioning: a review of theoretical and empirical work. *Frontiers in Psychology*, volume 6, 2015: p. 368.
- [30] Tarpy, R.; Mayer, R. *Foundations of Learning and Memory*. Scott, Foresman, 1978, ISBN 9780673150745.
- [31] Gauthier, L. How Learning Works: 7 Research-Based Principles for Smart Teaching. *Journal of the Scholarship of Teaching and Learning*, volume 14, 02 2013: p. 126, doi:10.14434/josotl.v14i1.4219.
- [32] Sarrica, T. Active Learning vs Traditional Lecture. Which Impacts Students More? [online]. 2018, [ref. 2024-04-30]. Available from: <https://teaching-learning.hastac.hcommons.org/2018/11/30/active-learning-vs-traditional-lecture-which-impacts-students-more/>
- [33] Theresa Pesavento, D. M. C. S. S. W., Jonathan Klein. Teaching with Technology [online]. 2015, [ref. 2024-04-30]. Available from: <https://wisc.pb.unizin.org/teachingwithtech/>

## BIBLIOGRAPHY

---

- [34] Darder, A. *Reinventing Paulo Freire: A Pedagogy of Love*. Routledge, 2017, ISBN 9781315560779. Available from: <https://doi.org/10.4324/9781315560779>
- [35] Dewey, J. *The Philosophy of Education: Democracy Education in USA, Moral Principles in Education, Health and Sex in Higher Education, The Child and the Curriculum*. DigiCat, 2023.
- [36] Montessori, M. *The Montessori Method*. Dover Publications Inc., 2002, ISBN 0486421627.
- [37] Barnes, D. R. *Active Learning*. Leeds University TVEI Support Project, 1989, ISBN 9781872364001.
- [38] Marcela de Oliveira e Silva Lemos, D. W., Kevin Mudavadi. Evidence Based Teaching: Active Learning [online]. 2024, [ref. 2024-04-30]. Available from: <https://citl.indiana.edu/teaching-resources/evidence-based/active-learning.html>
- [39] Valverde, C.; Allen, K.; et al. Active Learning Interventions in a Predominantly Black, Urban College Increase Positive Attitudes toward Class Participation. *The International Journal of Science, Mathematics and Technology Learning*, volume 30, January 2022, doi:10.18848/2327-7971/CGP/v30i01/17-30.
- [40] Tientongdee, S. Development of problem-solving skill by using active learning for student teachers in Introductory Physics. *Journal of Physics: Conference Series*, volume 1144, December 2018, doi:10.1088/1742-6596/1144/1/012002. Available from: <https://dx.doi.org/10.1088/1742-6596/1144/1/012002>
- [41] Millenbah, K.; Millspaugh, J. Using Experiential Learning in Wildlife Courses to Improve Retention, Problem Solving, and Decision-Making. *Wildlife Society Bulletin*, volume 31, 03 2003: pp. 127–137, doi:10.2307/3784366.
- [42] of Redlands, U. Active Learning [online]. 2024, [ref. 2024-04-30]. Available from: <https://sites.redlands.edu/information-technology-services/its-organization/instructional-technology/course-design-assistance/pedagogy/active-learning/>
- [43] Stranford, S. A.; Owen, J. A.; et al. Active Learning and Technology Approaches for Teaching Immunology to Undergraduate Students. *Frontiers in Public Health*, volume 8, 2020, ISSN 2296-2565, doi:10.3389/fpubh.2020.00114. Available from: <https://www.frontiersin.org/journals/public-health/articles/10.3389/fpubh.2020.00114>
- [44] Chinge, E.; Yii, T.; et al. Use of Technology in Active Learning Teaching Practices to Enhance Lecturers' Self-Efficacy in Technical University Environment. *International Journal of Engineering and Technology*, volume 9, 07 2020: pp. 436–443.

- 
- [45] Reddy, S.; Labutov, I.; et al. Unbounded Human Learning: Optimal Scheduling for Spaced Repetition. 2016, doi:10.1145/2939672.2939850. Available from: <https://doi.org/10.1145/2939672.2939850>
- [46] Wittman, D. J. The Forgetting Curve (California State University Stanislaus) [online]. 2024, [ref. 2024-04-30]. Available from: [https://www.csustan.edu/sites/default/files/groups/Writing%20Program/forgetting\\_curve.pdf](https://www.csustan.edu/sites/default/files/groups/Writing%20Program/forgetting_curve.pdf)
- [47] Schimanke, F.; Mertens, R.; et al. What to learn next? Content selection support in mobile game-based learning. 10 2013.
- [48] Kang, S. Spaced Repetition Promotes Efficient and Effective Learning: Policy Implications for Instruction. *Policy Insights from the Behavioral and Brain Sciences*, volume 3, January 2016, doi:10.1177/2372732215624708.
- [49] Baturay, M. H.; Yildirim, S.; et al. Effects of Web-Based Spaced Repetition on Vocabulary Retention of Foreign Language Learners. *Eurasian Journal of Educational Research (EJER)*, volume 8, December 2009.
- [50] Yuan, X. Evidence of the Spacing Effect and Influences on Perceptions of Learning and Science Curricula. 2022. Available from: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8759977/>
- [51] Paul Kelley, T. W. Making long-term memories in minutes: a spaced learning pattern from memory research in education. *Frontiers in human neuroscience*, 2013. Available from: <https://doi.org/10.3389/fnhum.2013.00589>
- [52] Vole, M. Improve Your Learning With Spaced Repetition [online]. 2022, [ref. 2024-04-30]. Available from: <https://insights.gostudent.org/en/spaced-repetition>
- [53] Keder, B. D. Computer-assisted language learning using spaced repetition [online]. 2009, [ref. 2024-04-30]. Available from: <https://is.muni.cz/th/uwa3f/diplomka.pdf>
- [54] for Teaching Innovation, C. Collaborative Learning [online]. 2024, [ref. 2024-04-30]. Available from: <https://teaching.cornell.edu/teaching-resources/active-collaborative-learning/collaborative-learning>
- [55] Yang, X. A Historical Review of Collaborative Learning and Cooperative Learning. *TechTrends*, 2023. Available from: <https://doi.org/10.1007/s11528-022-00823-9>
- [56] Cindy Hmelo-Silver, C. C. A. O., Clark Chinn. *The International Handbook of Collaborative Learning*. Routledge, 2013, ISBN 9780203837290, 1–2 pp.
- [57] University, C. Why use collaborative learning? [online]. 2024, [ref. 2024-04-30]. Available from: <https://teaching.cornell.edu/teaching-resources/active-collaborative-learning/collaborative-learning>

## BIBLIOGRAPHY

---

- [58] Qiannan Zhang, J. L., Sheng Lin; Jin, Y. A game perspective on collaborative learning among students in higher education. *Cogent Education*, volume 9, no. 1, 2022, doi:10.1080/2331186X.2022.2115617. Available from: <https://doi.org/10.1080/2331186X.2022.2115617>
- [59] Rutherford, S. *Collaborative learning: Theory, strategies and educational benefits*. January 2014.
- [60] Drew, C. Collaborative Learning: Pros Cons [online]. 2023, [ref. 2024-05-01]. Available from: <https://helpfulprofessor.com/collaborative-learning/>
- [61] Vali, I. The Impact of Technology on Collaborative Learning. *European Proceedings of Educational Sciences*, volume 5, 2023, doi:10.1080/2331186X.2022.2115617. Available from: <https://doi.org/10.15405/epes.23045.13>
- [62] Kapp, K. M. *The gamification of learning and instruction: Game-based methods and strategies for training and education*. Pfeiffer, 2012, ISBN 9781118096345, 7–12, 9–15 pp.
- [63] A Brief History Of Gamification In Education [online]. 2023, [ref. 2024-05-01]. Available from: <https://www.teachthought.com/education/a-brief-history-of-gamification-in-education/>
- [64] Toda, A. M.; Klock, A. C. T.; et al. Analysing gamification elements in educational environments using an existing Gamification taxonomy. *Smart Learning Environments*, volume 6, Dec 2019: p. 16, ISSN 2196-7091, doi:10.1186/s40561-019-0106-1. Available from: <https://doi.org/10.1186/s40561-019-0106-1>
- [65] Deterding, S.; Sicart, M.; et al. Gamification. using game-design elements in non-gaming contexts. New York, NY, USA: Association for Computing Machinery, 2011, ISBN 9781450302685, p. 2425–2428, doi:10.1145/1979742.1979575. Available from: <https://doi.org/10.1145/1979742.1979575>
- [66] Yildirim, I.; Sen, S. The effects of gamification on students' academic achievement: a meta-analysis study. *Interactive Learning Environments*, volume 29, no. 8, 2021: pp. 1301–1318, doi:10.1080/10494820.2019.1636089. Available from: <https://doi.org/10.1080/10494820.2019.1636089>
- [67] Mehrnoosh Khoshnoodifar, M. T., Asieh Ashouri. Effectiveness of Gamification in Enhancing Learning and Attitudes: A Study of Statistics Education for Health School Students. *Journal of advances in medical education professionalism*, volume 11, 2023: p. 230–239, doi:<https://doi.org/10.30476/JAMP.2023.98953.1817>. Available from: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10611935/>
- [68] Mirzaie Feiz Abadi, B.; Khalili Samani, N.; et al. Pros and Cons of Tomorrow's Learning: A Review of Literature of Gamification in Education Context. *Medical Education Bulletin*, volume 3, no. 4, 2022: pp. 543–554,

---

ISSN 2783-1809, doi:10.22034/meb.2022.350941.1063. Available from:  
[https://www.medicaleducation-bulletin.ir/article\\_153199.html](https://www.medicaleducation-bulletin.ir/article_153199.html)

- [69] Kayimbasioglu, D.; Oktekin, B.; et al. Integration of Gamification Technology in Education. *Procedia Computer Science*, volume 102, 2016: pp. 668–676, ISSN 1877-0509, doi:<https://doi.org/10.1016/j.procs.2016.09.460>, 12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016, 29-30 August 2016, Vienna, Austria. Available from: <https://www.sciencedirect.com/science/article/pii/S1877050916326400>
- [70] Cirillo, F. *The Pomodoro Technique*. Creative Commons, 2009, ISBN 9781445219943. Available from: <https://books.google.cz/books?id=ThkbQwAACAAJ>
- [71] R. Dizon, D. E., H. Ermitanio. The Effects of Pomodoro Technique on Academic-Related Tasks, Procrastination Behavior, and Academic Motivation Among College Students in a Mixed Online Learning Environment. *Globus Journal of Progressive Education*, 2021, doi:10.46360.
- [72] Sarkar, S.; Parnin, C. Characterizing and Predicting Mental Fatigue During Programming Tasks. In *Proceedings of the 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion '17)*, IEEE Press, 2017, pp. 32–37.
- [73] Schoolfield, N. Pomodoro Technique Pros And Cons [online]. 2021, [ref. 2024-05-01]. Available from: <https://beextrayoga.com/pomodoro-technique-pros-and-cons/>
- [74] Weinstein, Y.; Madan, C. R.; et al. Teaching the science of learning. *Cognitive Research: Principles and Implications*, volume 3, no. 1, Jan 2018: p. 2.
- [75] Rohrer, D. Interleaving helps students distinguish among similar concepts. *Educational Psychology Review*, volume 24, no. 3, 2012: pp. 355–367.
- [76] Vygotsky, L. S. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978, ISBN 9780674576285. Available from: <http://www.jstor.org/stable/j.ctvjf9vz4>
- [77] Garcia, T.; Pintrich, P. R. The Effects of Autonomy on Motivation and Performance in the College Classroom. *Contemporary Educational Psychology*, volume 21, no. 4, 1996: pp. 477–486, ISSN 0361-476X, doi: <https://doi.org/10.1006/ceps.1996.0032>.
- [78] Gupta, S.; Bostrom, R. Technology-Mediated Learning: A Comprehensive Theoretical Model. *J. AIS*, volume 10, 09 2009, doi:10.17705/1jais.00207.
- [79] Hwang, G.-J.; Wu, P.-H. Applications, impacts and trends of mobile technology-enhanced learning: A review of 2008-2012 publications in selected SSCI journals. *Int. J. of Mobile Learning and Organisation*, volume 8, 01 2014: pp. 83–95, doi:10.1504/IJMLO.2014.062346.

## BIBLIOGRAPHY

---

- [80] Liu, R.; Wang, L.; et al. Effects of an immersive virtual reality-based classroom on students' learning performance in science lessons. *British Journal of Educational Technology*, volume 51, 11 2020: pp. 2034–2049, doi:10.1111/bjet.13028.
- [81] Pimmer, C.; Linxen, S.; et al. Mobile learning in resource-constrained environments: a case study of medical education. *Med Teach*, volume 35, no. 5, Nov 2012: pp. e1157–65.
- [82] Henderson, A. App Review: Is Forest The Best Productivity App? [online]. 2021, [ref. 2024-05-02]. Available from: <https://aniahenderson.com/app-review-is-forest-the-best-productivity-app/>
- [83] Gallucci, N. Forest is a useful app that helps you go phone-free by inspiring you to plant trees [online]. 2019, [ref. 2024-05-02]. Available from: <https://mashable.com/article/forest-app-productivity-focus-review>
- [84] Song, V. How a Pomodoro timer app helped me regain my focus [online]. 2022, [ref. 2024-05-04]. Available from: <https://www.theverge.com/23466074/pomodoro-timers-focus-productivity-app>
- [85] SuperElement. Focus To-Do - Pomodoro Technique Tasks - Features [online]. 2024, [ref. 2024-05-04]. Available from: <https://www.focustodo.cn/#features>
- [86] Zelia. Case Study: Study Bunny App Redesign — What my first UI/UX project has taught me [online]. 2024, [ref. 2024-05-04]. Available from: <https://bootcamp.uxdesign.cc/what-my-first-ui-ux-case-study-has-taught-me-study-bunny-app-redesign-496ec1b91258>
- [87] Silang, J. P. Case Study: Study Bunny App Redesign — What my first UI/UX project has taught me [online]. 2024, [ref. 2024-05-04]. Available from: <https://superbyte.site/tutorial>
- [88] ISO/IEC 19505-2:2012. Information technology – Object Management Group Unified Modeling Language (OMG UML), 2012, available from ISO, ISO/IEC 19505-2:2012.
- [89] Wiegers, K. E.; Beatty, J. *Software Requirements 3*. USA: Microsoft Press, 2013, ISBN 0735679665, 9–14 pp.
- [90] Mannion, M.; Keepence, B. SMART requirements. *SIGSOFT Softw. Eng. Notes*, volume 20, no. 2, apr 1995: p. 42–47, ISSN 0163-5948, doi:10.1145/224155.224157. Available from: <https://doi.org/10.1145/224155.224157>
- [91] Al-Qutaish, R. Quality Models in Software Engineering Literature: An Analytical and Comparative Study. *Journal of American Science*, volume 6, 11 2010.

- 
- [92] Kravchenko, T.; Bogdanova, T.; et al. Ranking Requirements Using MoSCoW Methodology in Practice. In *Cybernetics Perspectives in Systems*, edited by R. Silhavy, Springer International Publishing, 2022, ISBN 978-3-031-09073-8, pp. 188–199.
- [93] Gomaa, H. *Software Modeling and Design*. Cambridge University Press, 2012, ISBN 9780511779183, 71–76 pp.
- [94] Google. Firebase Documentation [online]. 2024, [ref. 2024-05-05]. Available from: <https://firebase.google.com/docs>
- [95] Google. Firebase Authentication [online]. 2024, [ref. 2024-05-05]. Available from: <https://firebase.google.com/docs/auth>
- [96] Google. Cloud Firestore [online]. 2024, [ref. 2024-05-05]. Available from: <https://firebase.google.com/docs/firestore>
- [97] Apple. A Swift Tour [online]. 2024, [ref. 2024-05-05]. Available from: <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/guidedtour>
- [98] Apple. Swift [online]. 2024, [ref. 2024-05-05]. Available from: <https://developer.apple.com/swift/>
- [99] Apple. Automatic Reference Counting [online]. 2024, [ref. 2024-05-04]. Available from: <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/automaticreferencecounting1>
- [100] Hoffman, J. *Mastering Swift 5*. Packt Publishing Limited, 2019, ISBN 9781789139860, 131–140 pp.
- [101] Apple. *The Swift Programming Language*. Apple Inc., 2022, ISBN 9781789139860, 159–167 pp.
- [102] Apple. SwiftUI [online]. 2024, [ref. 2024-05-04]. Available from: <https://developer.apple.com/xcode/swiftui/>
- [103] Kannan, S. How SwiftUI View Works [online]. 2023, [ref. 2024-05-05]. Available from: <https://medium.com/@sarathiskannan/how-swiftui-view-works-7095f717d1c2>
- [104] Trogrlic, I. Best mobile app development tech stacks to use in 2024 [online]. 2024, [ref. 2024-05-05]. Available from: <https://decode.agency/article/best-mobile-app-development-tech-stack/>
- [105] Apple. Package Manager [online]. 2024, [ref. 2024-05-05]. Available from: <https://www.swift.org/documentation/package-manager/>
- [106] Apple. Testflight [online]. 2024, [ref. 2024-05-05]. Available from: <https://developer.apple.com/testflight/>
- [107] Apple. AppStore: Built for growth and scale [online]. 2024, [ref. 2024-05-05]. Available from: <https://developer.apple.com/app-store/features/>

## BIBLIOGRAPHY

---

- [108] Jeff Gilbert, C. S. Architecting iOS Apps with VIPER [online]. 2014, [ref. 2024-05-05]. Available from: <https://www.objc.io/issues/13-architecture/viper/>
- [109] Allies, P. Clean Architecture: iOS App [online]. 2022, [ref. 2024-05-05]. Available from: <https://nanosoft.co.za/blog/post/clean-architecture-ios>
- [110] Thomas, C. Breaking a Monolith: Using Cocoa Frameworks in iOS [online]. 2019, [ref. 2024-05-05]. Available from: <https://medium.com/john-lewis-software-engineering/breaking-a-monolith-using-cocoa-frameworks-in-ios-5f66a046c2aa>
- [111] Martin, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. USA: Prentice Hall Press, first edition, 2017, ISBN 0134494164, 176 pp.
- [112] Zeba. MVVM in iOS Swift [online]. 2023, [ref. 2024-05-05]. Available from: <https://medium.com/@zebayasmeen76/mvvm-in-ios-swift-6afb150458fd>
- [113] Nielsen, J. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1994, ISBN 9780080520292.
- [114] Alicar, M. What is wireframing? [online]. 2024, [ref. 2024-05-06]. Available from: <https://www.experienceux.co.uk/faqs/what-is-wireframing/>
- [115] ProtoIo. Prototyping for all. [online]. 2024, [ref. 2024-05-06]. Available from: <https://proto.io>
- [116] Atlassian. Welcome to Jira [online]. 2024, [ref. 2024-05-07]. Available from: <https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software>
- [117] Atlassian. A brief overview of Bitbucket [online]. 2024, [ref. 2024-05-07]. Available from: <https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket>
- [118] Atlassian. Jira for teams [online]. 2024, [ref. 2024-05-07]. Available from: <https://www.atlassian.com/software/jira/guides/getting-started/who-uses-jira#for-agile-teams>
- [119] Atlassian. Jira Bitbucket Integration [online]. 2024, [ref. 2024-05-07]. Available from: <https://www.atlassian.com/software/jira/bitbucket-integration>
- [120] Microsoft. .NET dependency injection [online]. 2024, [ref. 2024-05-07]. Available from: <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>
- [121] Long, M. Resolver [online]. 2024, [ref. 2024-05-07]. Available from: <https://github.com/hmlongco/Resolver>



- 
- [122] Apple. Navigation Stack [online]. 2024, [ref. 2024-05-07]. Available from: <https://developer.apple.com/documentation/swiftui/navigationstack>
- [123] van Elsloo, T. Unit test basics [online]. 2022, [ref. 2024-05-07]. Available from: <https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics>
- [124] Langr, J.; Hunt, A.; et al. *Pragmatic Unit Testing in Java 8 with JUnit*. Pragmatic Bookshelf, first edition, 2015, ISBN 1941222595, 78-82 pp.
- [125] Fowler, M. Mocking in Unit Tests [online]. 2023, [ref. 2024-05-07]. Available from: <https://microsoft.github.io/code-with-engineering-playbook/automated-testing/unit-testing/mocking/>
- [126] Riyam, R. UI Testing in Swift [online]. 2021, [ref. 2024-05-08]. Available from: <https://semaphoreci.com/blog/ui-testing-swift>
- [127] Hotjar. Usability testing: your 101 introduction [online]. 2023, [ref. 2024-05-08]. Available from: <https://www.hotjar.com/usability-testing/>
- [128] McCloskey, M. Turn User Goals into Task Scenarios for Usability Testing [online]. 2014, [ref. 2024-05-08]. Available from: <https://www.nngroup.com/articles/task-scenarios-usability-testing/>
- [129] Apple. Performance Tests [online]. 2024, [ref. 2024-05-08]. Available from: [https://developer.apple.com/documentation/xctest/performance\\_tests](https://developer.apple.com/documentation/xctest/performance_tests)
- [130] Apple. Reducing your app’s launch time [online]. 2024, [ref. 2024-05-08]. Available from: <https://developer.apple.com/documentation/xcode/reducing-your-app-s-launch-time>
- [131] Apple. Writing symbol documentation in your source files [online]. 2024, [ref. 2024-05-08]. Available from: <https://developer.apple.com/documentation/xcode/writing-symbol-documentation-in-your-source-files>
- [132] Apple. DocC [online]. 2024, [ref. 2024-05-08]. Available from: <https://www.swift.org/documentation/docc/>



---

# Requirement Specification

## A.1 Non-Functional Requirements

### 1. Identifier: NR1

**Name:** iOS 17.0+ Compatibility

**Description:** The application is compatible with the iOS platform version 17.0 or later.

**FURPS:** Usability

**MoSCoW:** Must Have

### 2. Identifier: NR2

**Name:** iPadOS 17.0+ Compatibility

**Description:** The application is compatible with the iPadOS platform version 17.0 or later.

**FURPS:** Usability

**MoSCoW:** Could Have

### 3. Identifier: NR3

**Name:** User-Friendly Interface

**Description:** The application is designed with ease of use in mind, ensuring a user-friendly experience.

**FURPS:** Usability

**MoSCoW:** Should Have

4. **Identifier:** NR4

**Name:** Application Extensibility

**Description:** The application is developed with extensibility in mind, allowing for easy expansion and customization.

**FURPS:** Supportability

**MoSCoW:** Must Have

5. **Identifier:** NR5

**Name:** WatchOS Compatibility

**Description:** The application is compatible with the WatchOS platform.

**FURPS:** Usability

**MoSCoW:** Won't Have

## A.2 Functional Requirements

1. **Identifier:** FR1

**Name:** Account Creation

**Description:** Users can create a new account to access the application.

**FURPS:** Functionality

**MoSCoW:** Must Have

2. **Identifier:** FR2

**Name:** Log In

**Description:** Users can log in to their existing accounts.

**FURPS:** Functionality

**MoSCoW:** Must Have

3. **Identifier:** FR3

**Name:** Log Out

**Description:** Users can log out of their accounts.

**FURPS:** Functionality

**MoSCoW:** Must Have

4. **Identifier:** FR4

**Name:** Email Verification

**Description:** User accounts require email verification before accessing the application.

**FURPS:** Functionality

**MoSCoW:** Should Have

5. **Identifier:** FR5

**Name:** Reset Password

**Description:** Users can reset their forgotten passwords.

**FURPS:** Functionality

**MoSCoW:** Should Have

6. **Identifier:** FR6

**Name:** Change Password

**Description:** Users can change their existing passwords.

**FURPS:** Functionality

**MoSCoW:** Could Have

7. **Identifier:** FR7

**Name:** Change Username

**Description:** Users can update their usernames.

**FURPS:** Functionality

**MoSCoW:** Could Have

8. **Identifier:** FR8

**Name:** Change Email

**Description:** Users can modify their registered email addresses.

**FURPS:** Functionality

**MoSCoW:** Could Have

9. **Identifier:** FR9

**Name:** Account Deletion

**Description:** Users can permanently delete their accounts and associated data.

**FURPS:** Functionality

**MoSCoW:** Must Have

10. **Identifier:** FR10

**Name:** English Localization

**Description:** Users can switch the application language to English.

**FURPS:** Functionality

**MoSCoW:** Must Have

11. **Identifier:** FR11

**Name:** Czech Localization

**Description:** Users can switch the application language to Czech.

**FURPS:** Functionality

**MoSCoW:** Could Have

12. **Identifier:** FR12

**Name:** Plain Timer

**Description:** Users can start a basic Pomodoro Timer and adjust study and break durations.

**FURPS:** Functionality

**MoSCoW:** Must Have

13. **Identifier:** FR13

**Name:** Create Storyline

**Description:** Users can create interactive storylines, integrating gamification with the Pomodoro technique.

**FURPS:** Functionality

**MoSCoW:** Must Have

14. **Identifier:** FR14

**Name:** Delete Storyline

**Description:** Users can delete their created storylines.

**FURPS:** Functionality

**MoSCoW:** Must Have

15. **Identifier:** FR15

**Name:** Update Storyline

**Description:** Users can modify timer settings and goals for their storylines.

**FURPS:** Functionality

**MoSCoW:** Could Have

16. **Identifier:** FR16

**Name:** Create Guild

**Description:** Users can create guilds for collaborative or competitive activities.

**FURPS:** Functionality

**MoSCoW:** Must Have

17. **Identifier:** FR17

**Name:** Update Guild

**Description:** Guild leaders can update the guild's goal.

**FURPS:** Functionality

**MoSCoW:** Could Have

18. **Identifier:** FR18

**Name:** Delete Guild

**Description:** Guild leaders can delete their guilds.

**FURPS:** Functionality

**MoSCoW:** Must Have

19. **Identifier:** FR19

**Name:** Invite Guild Member

**Description:** Guild members with sufficient permissions can invite new members.

**FURPS:** Functionality

**MoSCoW:** Must Have

20. **Identifier:** FR20

**Name:** Remove Guild Member

**Description:** Guild leaders can remove members from their guilds.

**FURPS:** Functionality

**MoSCoW:** Could Have

21. **Identifier:** FR21

**Name:** Board

**Description:** The application includes a board where users can compare their total scores.

**FURPS:** Functionality

**MoSCoW:** Must Have

22. **Identifier:** FR22

**Name:** Board Sorting

**Description:** Users can customize the sorting order of the board list.

**FURPS:** Functionality

**MoSCoW:** Could Have

23. **Identifier:** FR23

**Name:** Notifications

**Description:** The application sends push notifications to update users on Pomodoro timer changes.

**FURPS:** Functionality

**MoSCoW:** Could Have

24. **Identifier:** FR24

**Name:** Calendar Progress Tracker And Organizer

**Description:** Users can track their study progress with a calendar view and access charts and statistics.

**FURPS:** Functionality

**MoSCoW:** Won't Have

25. **Identifier:** FR25

**Name:** Authentication Process Deeplinks

**Description:** Incorporates deeplinks for a smoother authentication process.

**FURPS:** Functionality

**MoSCoW:** Won't Have

26. **Identifier:** FR26

**Name:** In-App Purchases

**Description:** Users can purchase additional storylines within the application.

**FURPS:** Functionality

**MoSCoW:** Won't Have

27. **Identifier:** FR27

**Name:** Dynamic Island Support

**Description:** Displays Pomodoro timer progress on the Dynamic Island feature.

**FURPS:** Functionality

**MoSCoW:** Won't Have



---

## Use Case Specification

### 1. Identifier: UC1

**Name:** Sign Up

**Description:** Initially, users are directed to a sign up form where they are prompted to input their username, email, password (adhering to security standards), and confirm their password. After completing the form, users click the *Sign Up* button. If the form is filled out correctly, users receive a notification confirming that an activation email has been sent along with additional instructions. If there are errors in the form, users are prompted to correct them.

**Requirements Covered:** FR1, FR4, FR25

### 2. Identifier: UC2

**Name:** Log In

**Description:** Upon clicking the *Login* button at the bottom of the screen, users are prompted to enter their email and password into the login form. If the combination of these details is valid, users gain access to the application's home screen. Otherwise, they are informed of the reason for the unsuccessful login attempt.

In the event of a forgotten password, users can utilize the *Reset Password* link at the bottom of the login form. They are prompted to enter their email, and an email with additional instructions for password reset is sent to them. If the account is not registered with the provided email, users are alerted accordingly.

**Requirements Covered:** FR2, FR5, FR25

### 3. Identifier: UC3

**Name:** Log Out

**Description:** Logging out is possible only for logged-in users. This option is located in the *Settings* tab. Upon selecting the *Log Out* option in the scroll menu, the application logs the users out and redirects them to the initial home screen.

**Requirements Covered:** FR2, FR3

### 4. Identifier: UC4

**Name:** Change Password

**Description:** To change the password, users need to navigate to the *Settings* tab after logging into the application. Clicking on the *Change Password* option opens a form where users must input their current password, new password, and confirm the new password. The new password must meet the same security standards as the original. Upon successfully filling out the form, users receive a notification confirming that their password has been changed. In case of any errors, users are alerted to the specific inconsistency.

**Requirements Covered:** FR2, FR6

### 5. Identifier: UC5

**Name:** Change Email

**Description:** To change the email, users must navigate to the *Settings* tab after logging into the application. Clicking on the *Email* option opens a form where users must input their current password, new email, and confirm the new email. Upon successfully filling out the form, users receive a notification confirming that their email has been changed and a verification link has been sent to the new email. In case of any errors, users are alerted to the specific inconsistency.

**Requirements Covered:** FR2, FR4, FR8, FR25

### 6. Identifier: UC6

**Name:** Change Username

**Description:** To change the username, users must navigate to the *Settings* tab after logging into the application. Clicking on the *Username* option opens a form where users must input their current password and new username. Upon successfully filling out the form, users receive a notification confirming that their username has been changed. In case of any errors, users are alerted to the specific inconsistency.

**Requirements Covered:** FR2, FR4, FR7

---

7. **Identifier:** UC7

**Name:** Delete Account

**Description:** To delete an account, users must navigate to the *Settings* tab after logging into the application. Clicking on the *Delete Account* option opens a form where users must enter their current password and confirm it. Upon successfully completing the form, users are presented with an alert asking if they are sure about this action. If they accept, another alert detailing what data will be deleted is shown. If the user confirms this second alert, the account and all associated data are deleted. If either alert is dismissed, the action is aborted. Upon successful account deletion, the application returns to the initial home screen.

**Requirements Covered:** FR2, FR9

8. **Identifier:** UC8

**Name:** Show My Storylines

**Description:** To view their storylines, users navigate to the *Home* tab after logging into the application. In this tab, all the user's storylines are listed in the *My Storylines* section.

**Requirements Covered:** FR2

9. **Identifier:** UC9

**Name:** Start Plain Timer

**Description:** To start a new plain Pomodoro timer, users navigate to the *Home* tab after logging into the application. Here, users can tap on the *Plain Timer* tile at the top of the screen. After tapping, a bottom sheet appears allowing the users to set parameters for the Pomodoro Timer (study and break interval lengths). When users are satisfied with their settings, they can tap on the *Start* button at the bottom of that sheet to start the timer. The timer will send a push notification at the end of the timer period if users have opted to allow notifications during the first start of the application.

**Requirements Covered:** FR2, FR12, FR23, FR27

### 10. Identifier: UC10

**Name:** Create New Storyline

**Description:** To create a new storyline, users navigate to the *Storyline* tab after logging into the application. In this tab, users select their desired storyline and tap on it. A quick presentation of the storyline is displayed, allowing the users to set up the new storyline with the *Setup Storyline* button at the bottom of that screen. After tapping on that button, users are presented with a setup screen, where they can set their goals for that storyline as well as study and break interval lengths (parameters for the Pomodoro timer). When users are satisfied with their storyline settings, they can tap the *Create Storyline* button at the bottom of the setup screen. After creating a new storyline, users are automatically redirected to the *Home* tab.

**Requirements Covered:** FR2, FR13

### 11. Identifier: UC11

**Name:** Delete Storyline

**Description:** To delete a storyline, users navigate to the *Home* tab after logging into the application. In this tab, users select one of their storylines listed under the *My Storylines* section. By long pressing that storyline, a context menu appears. Users select the *Delete* menu option to delete the storyline. Users are prompted with an alert asking them to confirm their action. If confirmed, the storyline is deleted; otherwise, the action is aborted.

**Requirements Covered:** FR2, FR14

### 12. Identifier: UC12

**Name:** Update Storyline

**Description:** To update a storyline, users navigate to the *Home* tab after logging into the application. In this tab, users select one of their storylines listed under the *My Storylines* section. By long pressing that storyline, a context menu appears. Users select the *Update* menu option to change the parameters of the storyline. A bottom sheet then appears, allowing users to change the parameters of the storyline. When users are done with their settings, they can tap the *Setup Storyline* button, which saves the new settings.

**Requirements Covered:** FR2, FR15

---

13. **Identifier:** UC13

**Name:** Show My Guilds or Guild Detail

**Description:** To view their guilds, users navigate to the *Guilds* tab after logging into the application. In this tab, all guilds to which the users belong are listed in the *Guilds* section. By tapping on a guild preview in the list, a detailed view of the guild is presented.

**Requirements Covered:** FR2

14. **Identifier:** UC14

**Name:** Create New Guild

**Description:** To create a new guild, users navigate to the *Guilds* tab after logging into the application. Users tap the *Create New Guild* tile at the bottom of the *Guilds* tab. A new bottom sheet appears allowing users to set parameters for the new guild. They are presented with a form asking them to fill in the name for the new guild, pick a storyline that the guild will be based on, and set the guild goal. Optionally, they can add the email(s) of their friends to whom they want invitations sent along with the guild creation. After setting up the guild, users can tap the *Create Guild* button at the bottom of the presented sheet. After tapping the button, a new guild is created and a preview of it is displayed in the list of guilds on the *Guilds* tab.

**Requirements Covered:** FR2, FR16

15. **Identifier:** UC15

**Name:** Update Guild

**Description:** To update a guild, users must navigate to the *Guilds* tab after logging into the application. They also need to be a guild leader or have elevated permissions to be able to make changes to the guild settings. At the *Guilds* tab, users open the detail of a guild by tapping on its preview tile. After that, there is a *Leader Actions* button at the top of the detail that users can tap. This button presents a context menu which offers guild administration options based on the users' guild permissions. One of those options is *Update Guild*. Tapping this option presents a bottom sheet which allows the users to change the goal of the guild, its name, and the storyline it has set. When users are satisfied with their new settings, they can tap the *Setup Guild* button at the bottom, effectively changing the settings of the guild.

**Requirements Covered:** FR2, FR17

### 16. Identifier: UC16

**Name:** Delete Guild

**Description:** To delete a guild, users must navigate to the *Guilds* tab after logging into the application. They also need to be a guild leader to be able to delete a guild. At the *Guilds* tab, users open the detail of a guild by tapping on its preview tile. After that, there is a *Leader Actions* button at the top of the detail that users can tap. This button presents a context menu which offers guild administration options based on the users' guild permissions. One of those options is *Delete Guild*. Tapping this option presents an alert which prompts the users to confirm their action. If confirmed, the guild is deleted. The action is aborted otherwise.

**Requirements Covered:** FR2, FR18

### 17. Identifier: UC17

**Name:** Invite Someone To a Guild

**Description:** To invite someone new to a guild, users must navigate to the *Guilds* tab after logging into the application. They also need to be a guild leader or have elevated permissions to be able to invite a new member. At the *Guilds* tab, users open the detail of a guild they want to invite someone to by tapping on its preview tile. After that, there is a *Leader Actions* button at the top of the detail that users can tap. This button presents a context menu which offers guild administration options based on the users' guild permissions. One of those options is *Invite Friend*. Tapping this option presents a bottom sheet where users need to fill in the email of the user they want to invite. After that, they can tap on the *Invite Friend* button at the bottom of the sheet. After that, an alert is presented to the users informing them about the status of their invitation—either it was sent to the user, the user was not found, or there was an error. In any case, the users are properly informed about what is happening.

**Requirements Covered:** FR2, FR19

---

18. **Identifier:** UC18

**Name:** Remove Someone From a Guild

**Description:** To remove someone from a guild, users must navigate to the *Guilds* tab after logging into the application. They also need to be a guild leader or have elevated permissions to be able to remove a guild member. At the *Guilds* tab, users open the detail of a guild they want to remove someone from by tapping on its preview tile. After that, they need to find the user they want to remove from the guild in the list of members under the *Members* section of the detail screen. Long pressing on the member's name will bring up a context menu, offering the item *Kick from the Guild*. Pressing this will present a confirmation alert prompt. If confirmed, the member is removed; otherwise, the action is aborted.

**Requirements Covered:** FR2, FR20

19. **Identifier:** UC19

**Name:** Show Board With Optional Sorting

**Description:** To show a board, users navigate to the *Board* tab after logging into the application. This tab will present a list of users, sorted by score from highest to lowest. Only the top 10 users will be displayed on this tab. If there are more than 10 users in total, a *Show More* button will appear at the bottom of the tab. Tapping this button will present a new view with all members. Optionally, if users prefer a different sorting of the board, by pressing the *Sorting* button at the top of the screen, the board sorting can be modified. This button will show a context menu with the following options: *Score Descending*, *Score Ascending*, *Username (a-z)*, and *Username (z-a)*. Tapping an option from the menu will change the order of the users in the board.

**Requirements Covered:** FR2, FR21, FR22

20. **Identifier:** UC20

**Name:** Change Language

**Description:** To change the language, users navigate to the *Settings* tab after logging into the application. At this tab, users tap on the *Change Language* item in the settings list. This option will present an alert explaining to users that the language can be set in *System Settings*. The alert provides two options—*Continue* and *Cancel*, where *Continue* opens *System Settings* allowing users to switch to the desired language there. The other available option, which is a *Cancel* button, aborts the action.

**Requirements Covered:** FR2, FR10, FR11

21. **Identifier:** UC21

**Name:** Change Notification Preferences

**Description:** To change to desired notification preferences, users navigate to the *Settings* tab after logging into the application. There, users tap on the *Change Notification Preferences* item in the settings list. This option will present an alert explaining to users that notification preferences can be changed in *System Settings*. The alert provides two options—*Continue* and *Cancel*, where *Continue* opens *System Settings* allowing users to change notification preferences there. The *Cancel* option aborts the action.

**Requirements Covered:** FR2, FR23

22. **Identifier:** UC22

**Name:** Show Statistics

**Description:** To show statistics, users navigate to the *Board* tab after logging into the application. At this tab, users tap the *Statistics* button at the top of the screen. This will present a new screen with a calendar and individual statistics of the users' studying which the user can browse.

**Requirements Covered:** FR2, FR24

23. **Identifier:** UC23

**Name:** Unlock a Storyline

**Description:** To unlock a storyline, users navigate to the *Storylines* tab after logging into the application. At this tab, users tap a locked storyline (grayed out). This will present a new screen stating information about the payment that is required to unlock the storyline. In case of successful payment, the storyline is unlocked.

**Requirements Covered:** FR2, FR26



# Wireframes

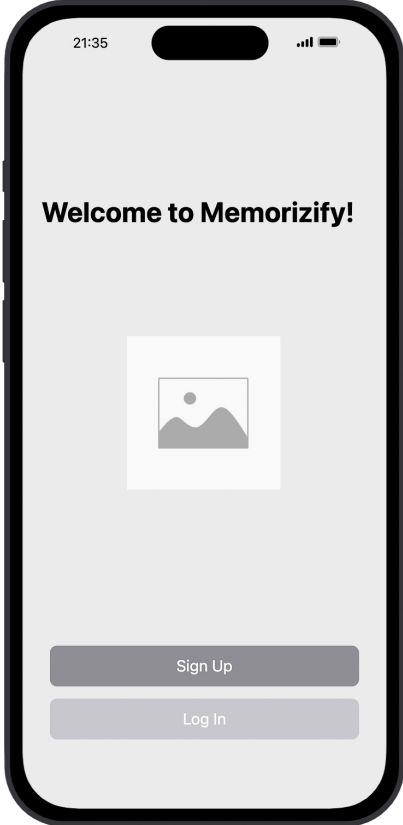


Figure C.1: Initial Home

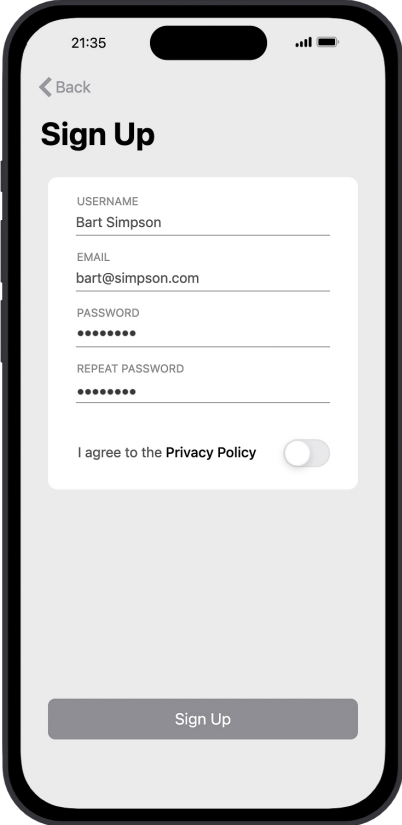


Figure C.2: Sign Up

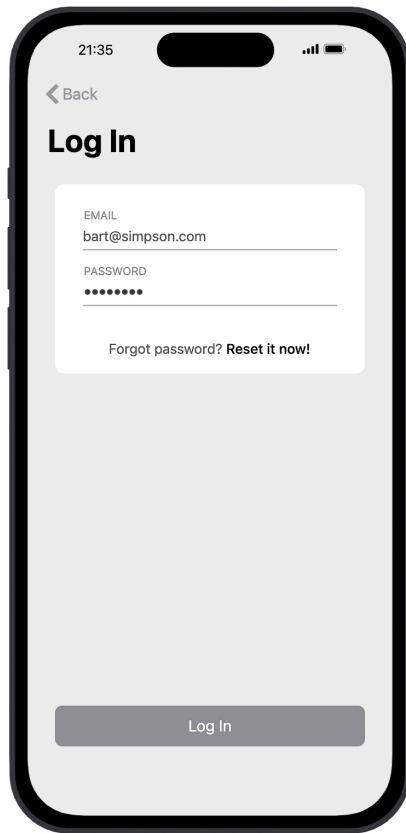


Figure C.3: Log In

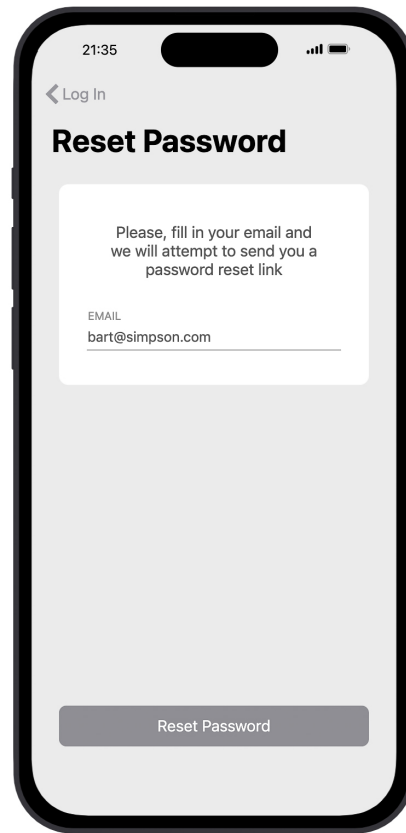


Figure C.4: Reset Password

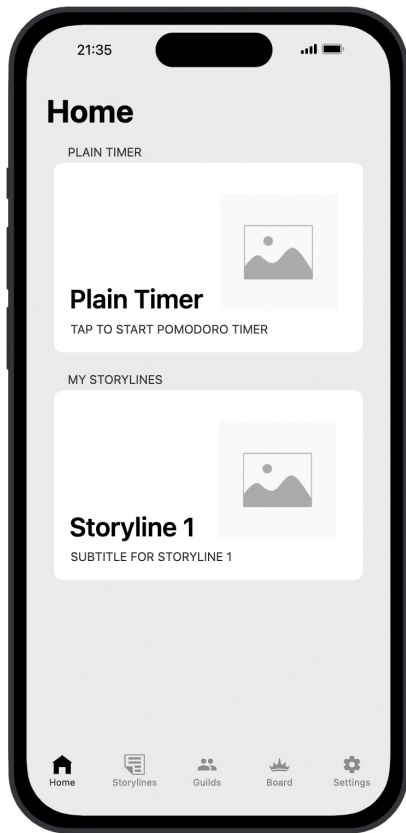


Figure C.5: Home Tab



Figure C.6: Plain Timer



Figure C.7: Storylines Tab



Figure C.8: Storyline Detail

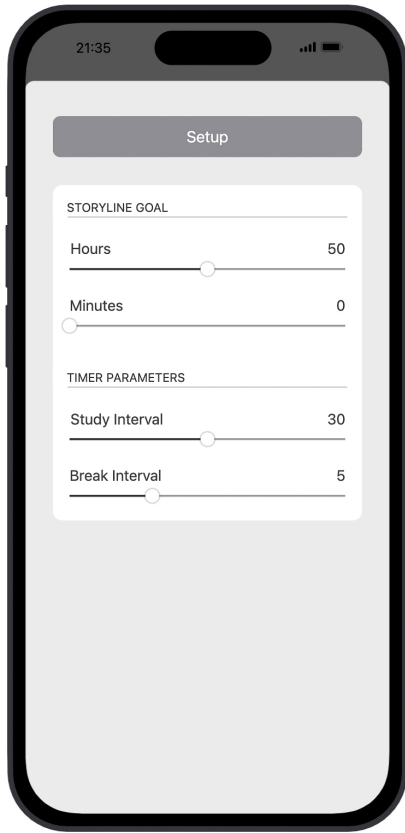


Figure C.9: Storyline Setup



Figure C.10: Storyline Timer

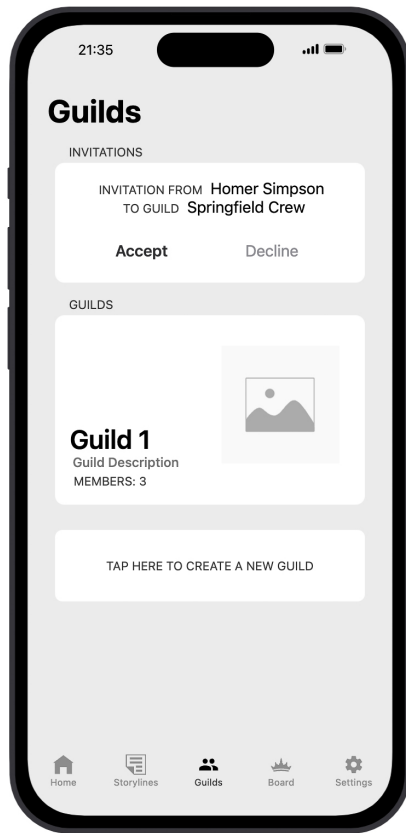


Figure C.11: Guilds Tab

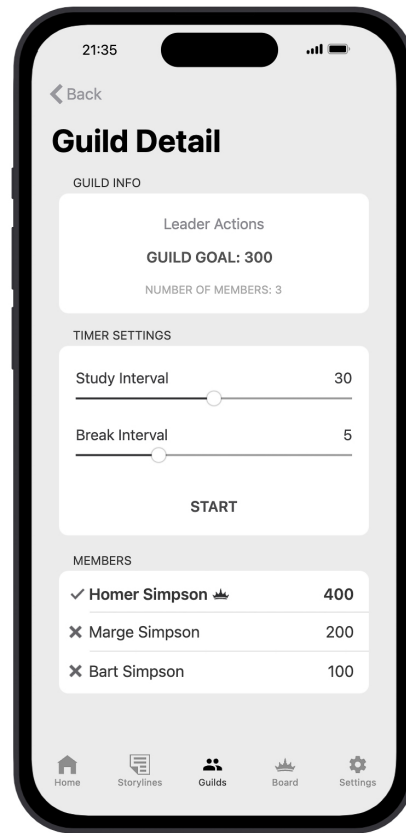


Figure C.12: Guild Detail

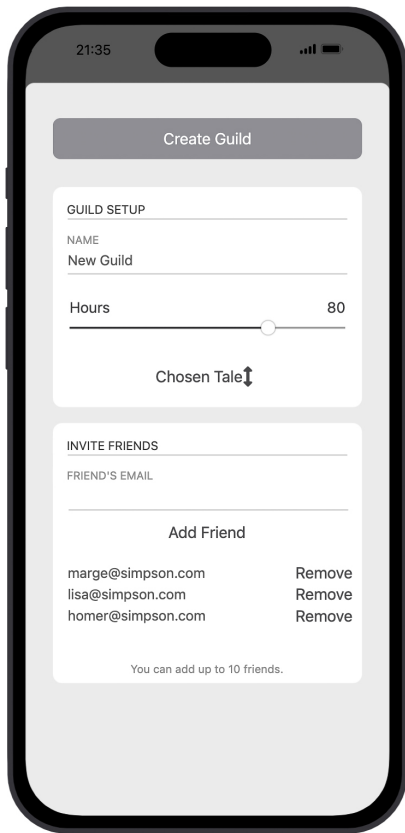


Figure C.13: Create Guild

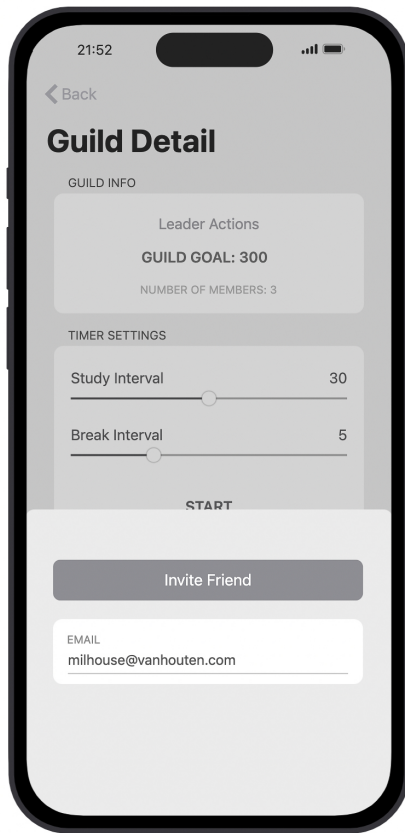


Figure C.14: Invite Friend

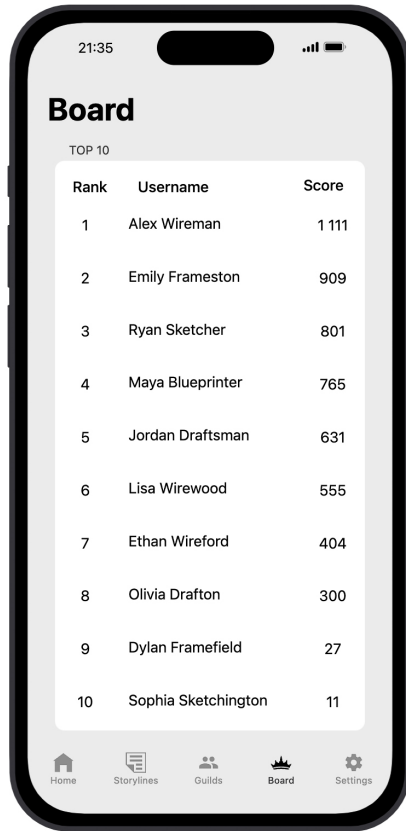


Figure C.15: Board Tab

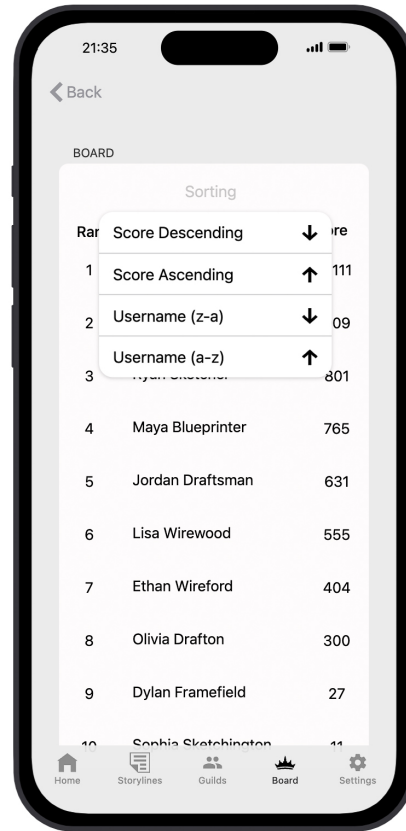


Figure C.16: Board Detail



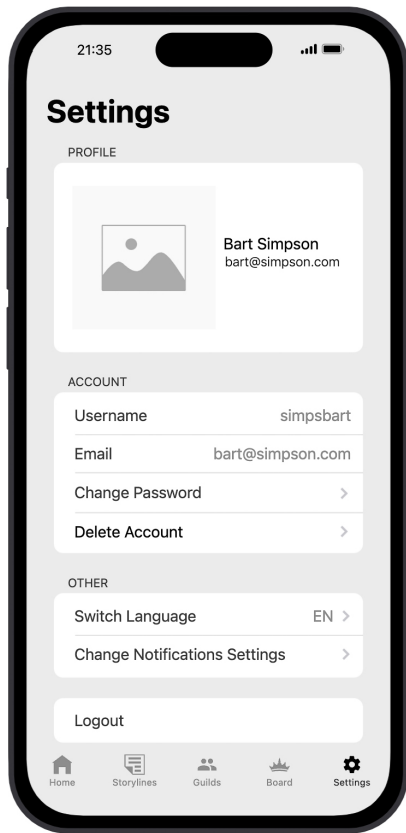


Figure C.17: Settings Tab

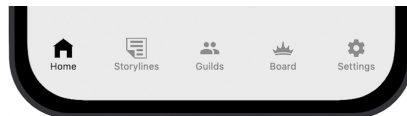


Figure C.18: Navigation Bar



## Final Application Interface Screenshots

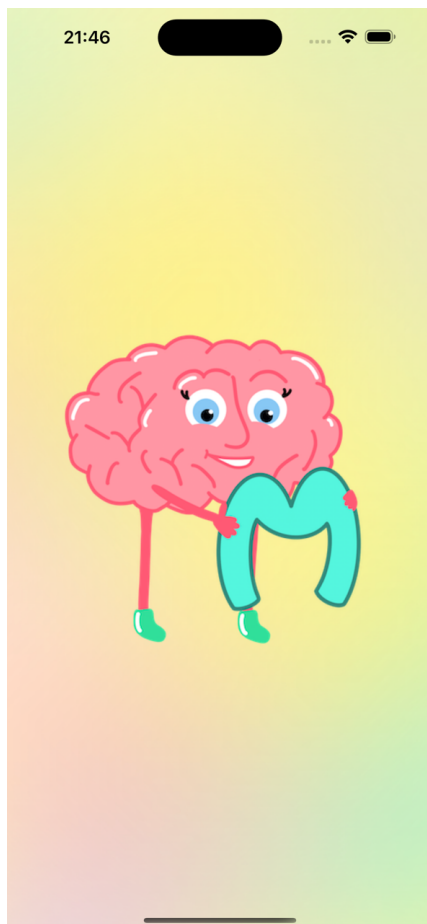


Figure D.1: Launch Screen

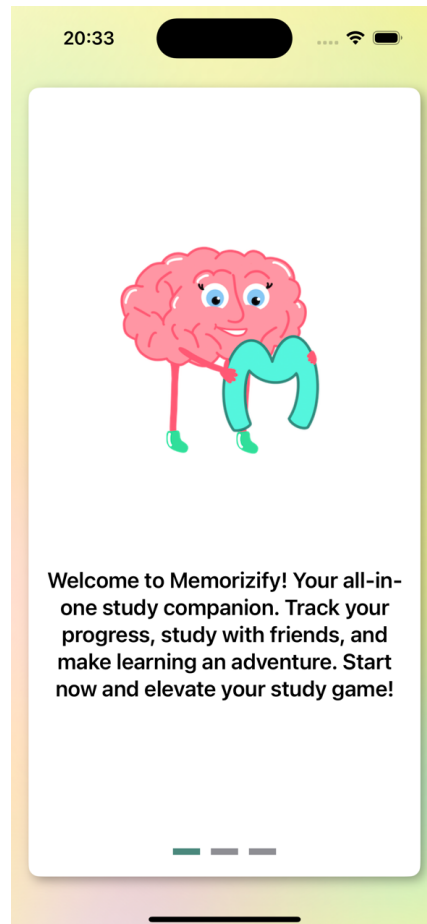


Figure D.2: Onboarding 1st Page

D. FINAL APPLICATION INTERFACE SCREENSHOTS

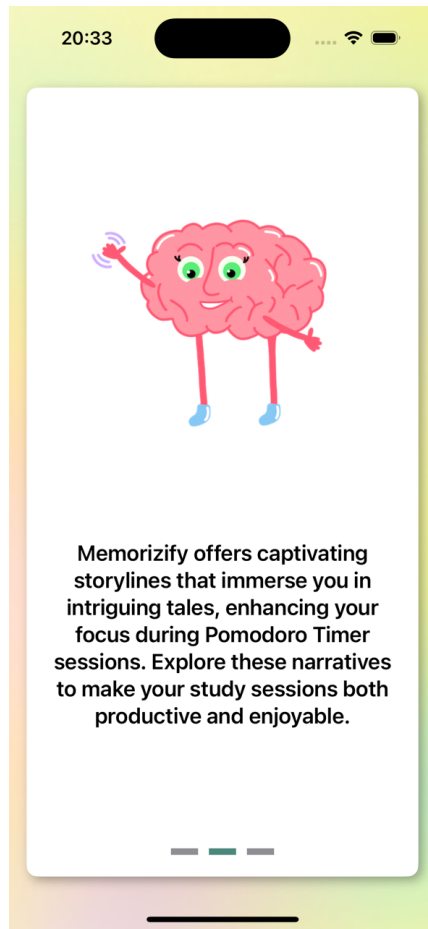


Figure D.3: Onboarding 2nd Page



Figure D.4: Onboarding 3rd Page

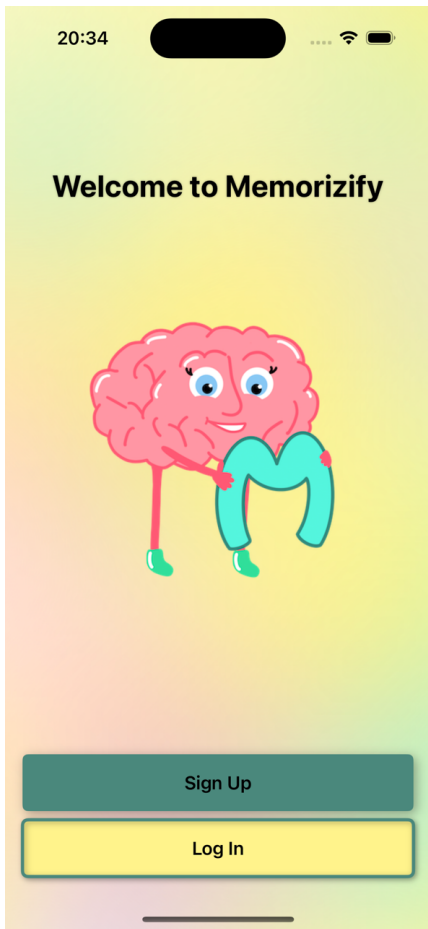


Figure D.5: Initial Home

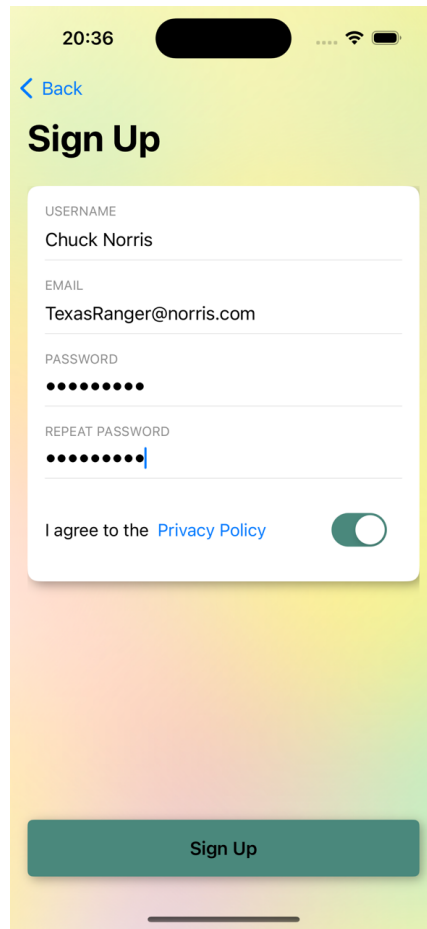


Figure D.6: Sign Up

## D. FINAL APPLICATION INTERFACE SCREENSHOTS

---

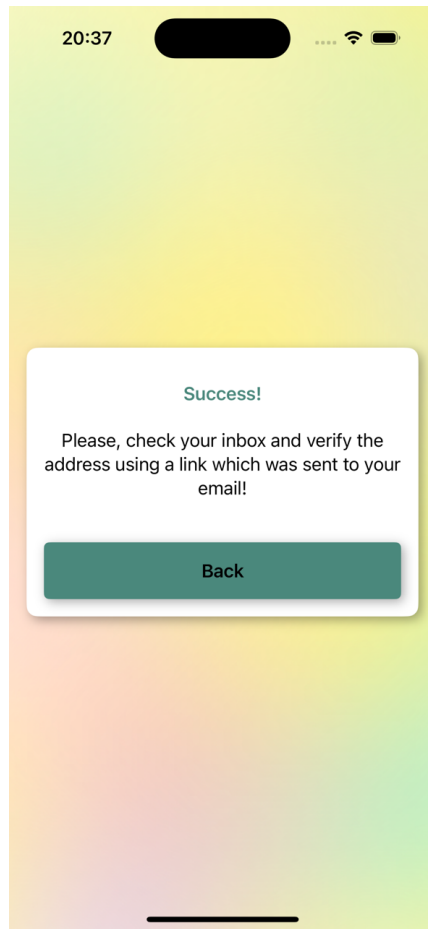


Figure D.7: Sign Up Done

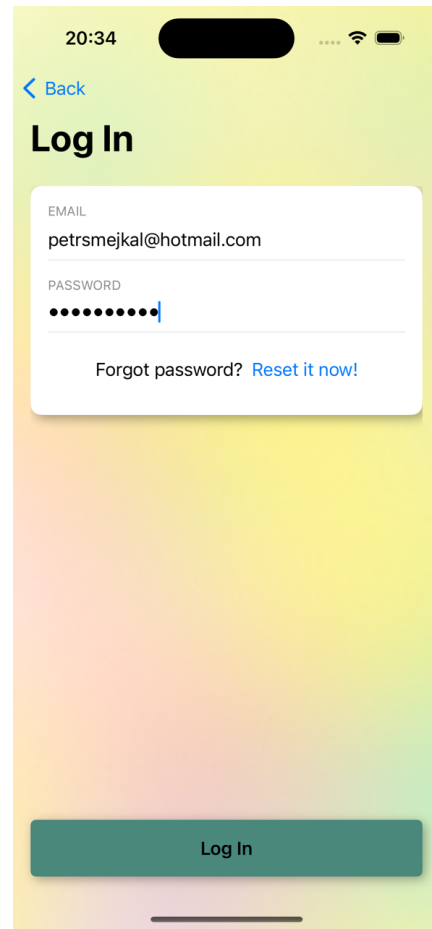


Figure D.8: Log In

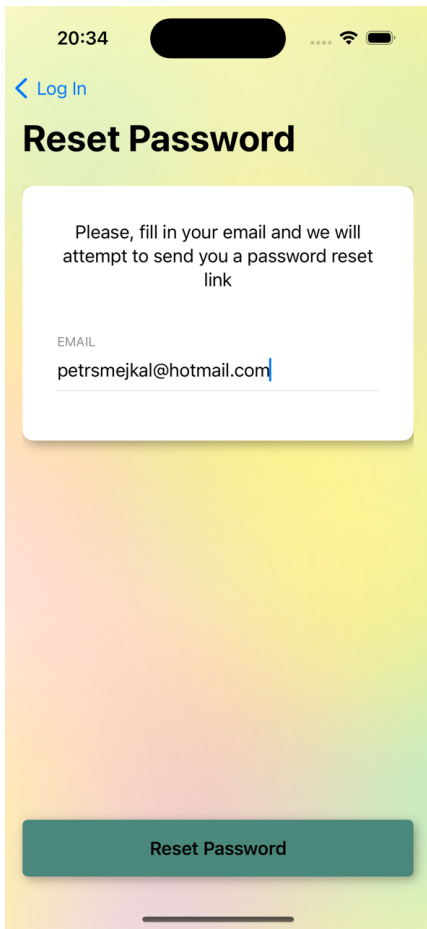


Figure D.9: Reset Password

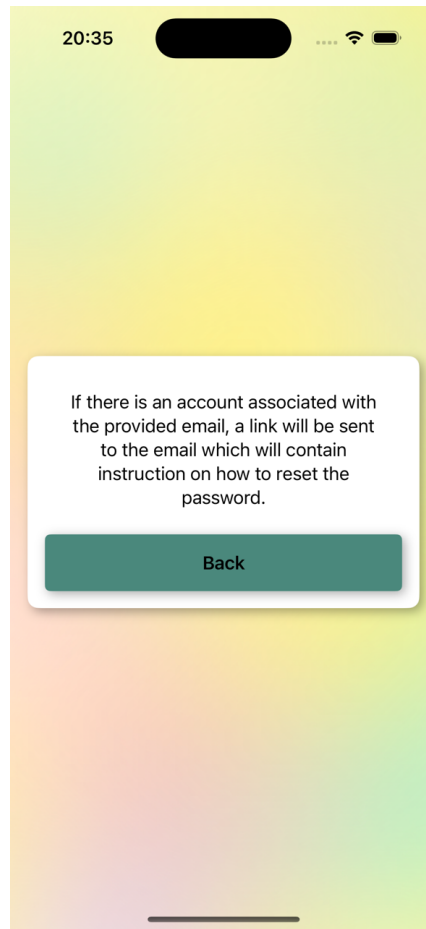


Figure D.10: Password Reset Done

D. FINAL APPLICATION INTERFACE SCREENSHOTS

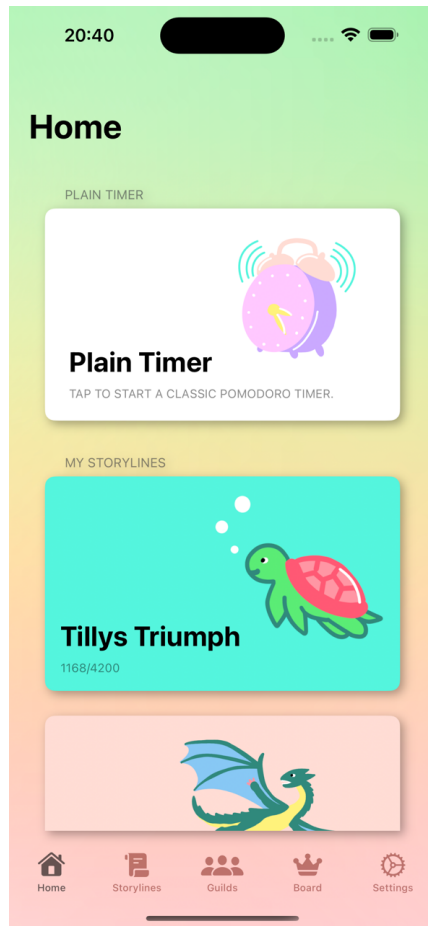


Figure D.11: Home Tab

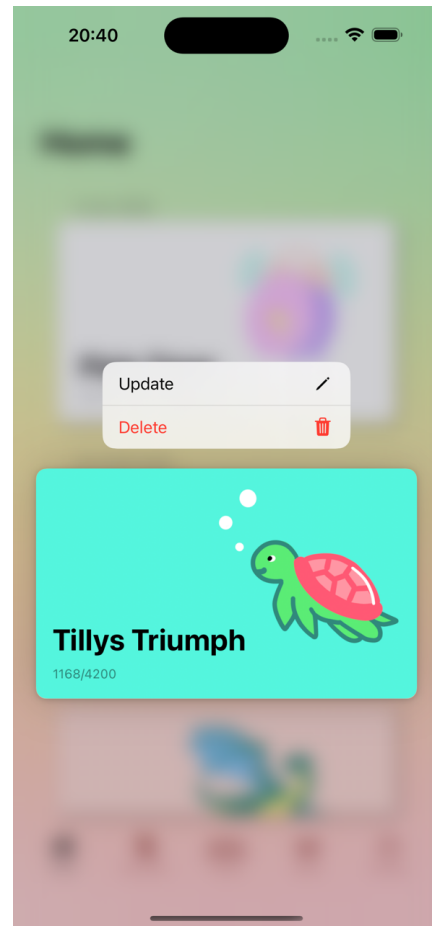


Figure D.12: Storyline Menu



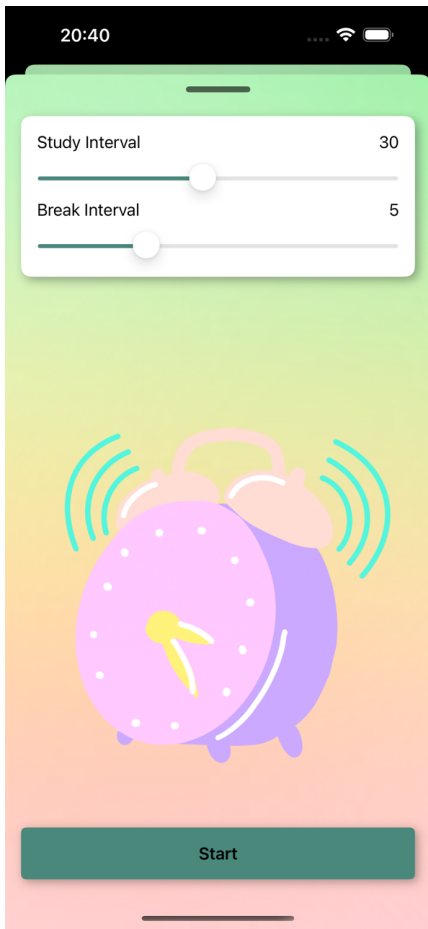


Figure D.13: Plain Timer Setup



Figure D.14: Plain Timer

## D. FINAL APPLICATION INTERFACE SCREENSHOTS

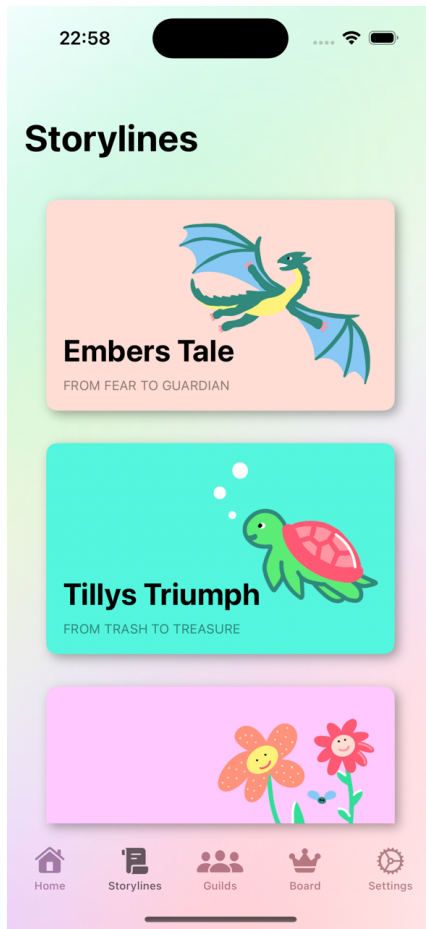


Figure D.15: Storylines Tab

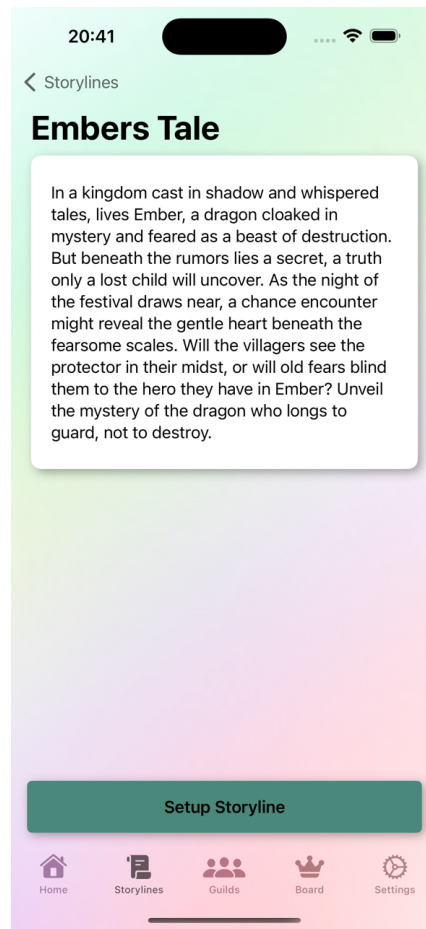


Figure D.16: Storyline Detail

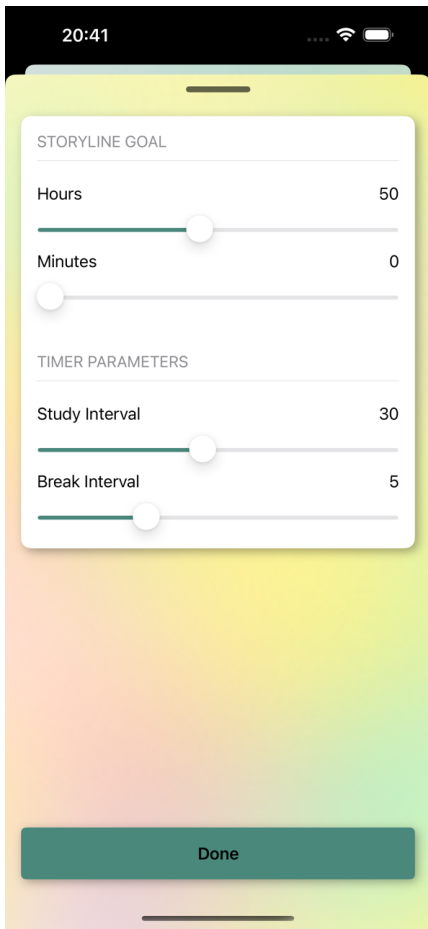


Figure D.17: Storyline Setup

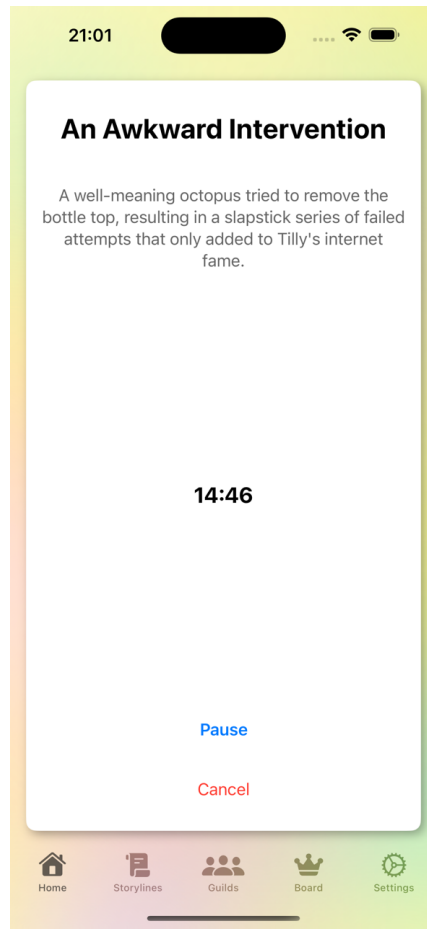


Figure D.18: Storyline Timer

D. FINAL APPLICATION INTERFACE SCREENSHOTS

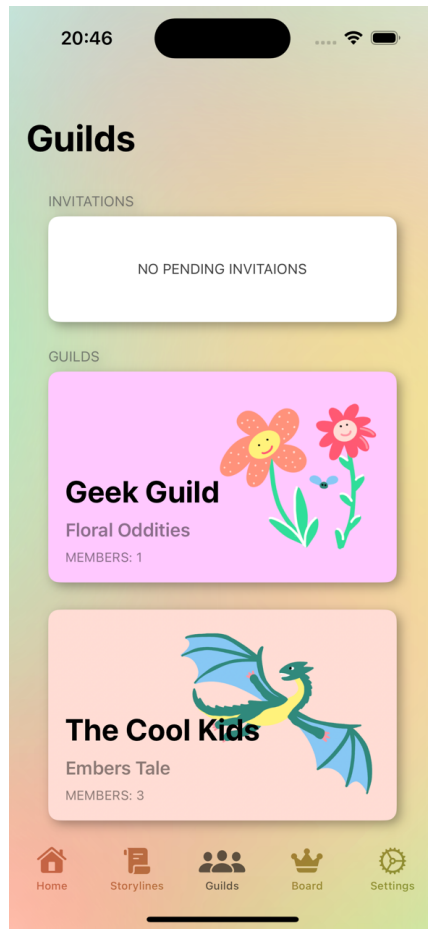


Figure D.19: Guilds Tab

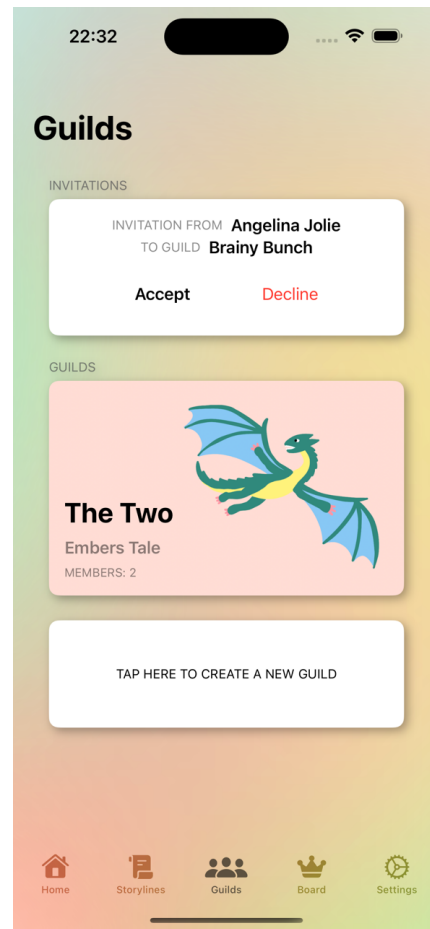


Figure D.20: Guild Invitation

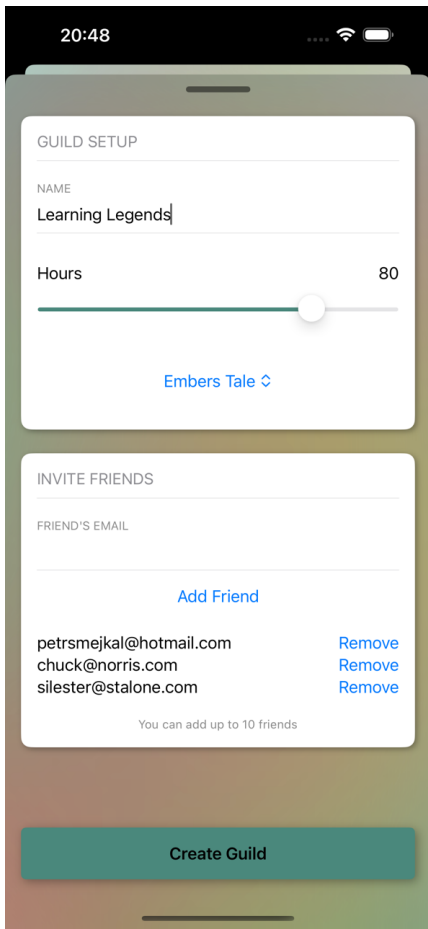


Figure D.21: Create Guild

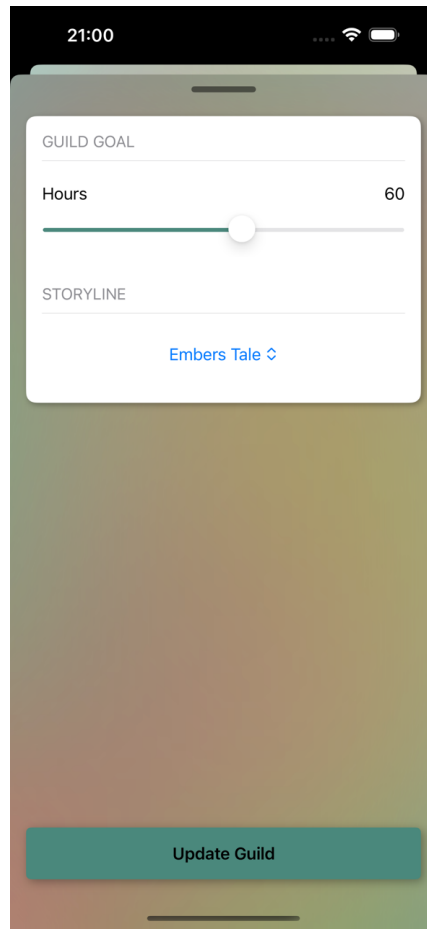


Figure D.22: Update Guild

## D. FINAL APPLICATION INTERFACE SCREENSHOTS

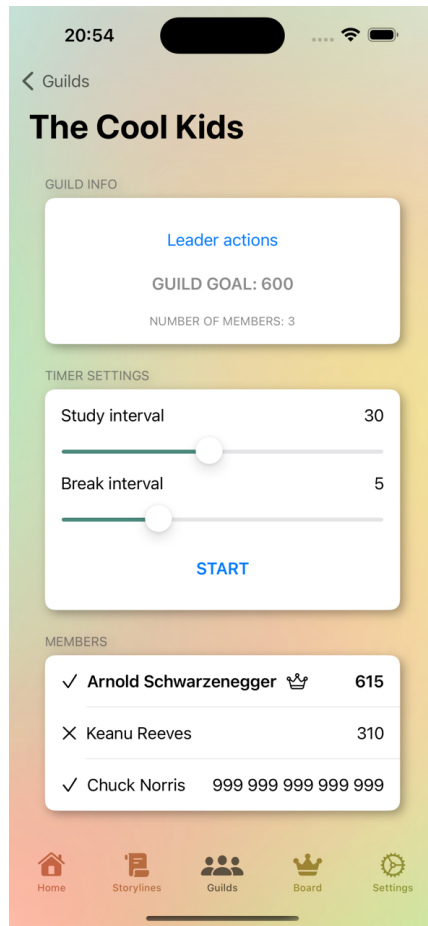


Figure D.23: Guild Detail

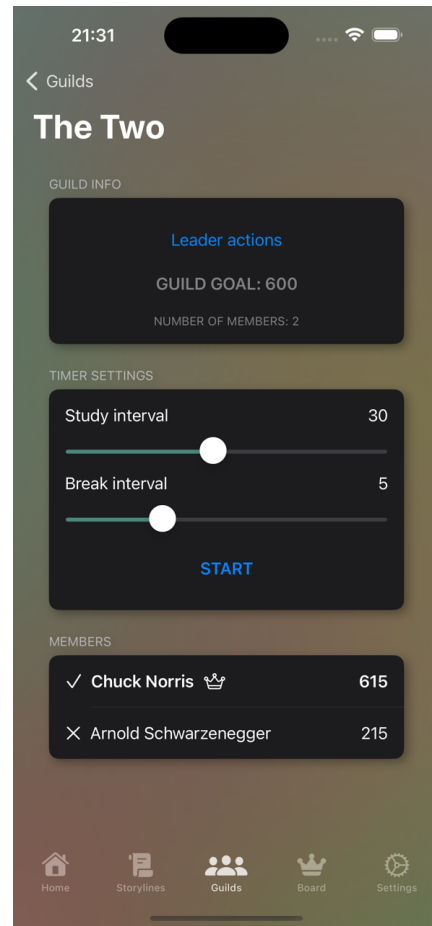


Figure D.24: Guild Detail (Dark)

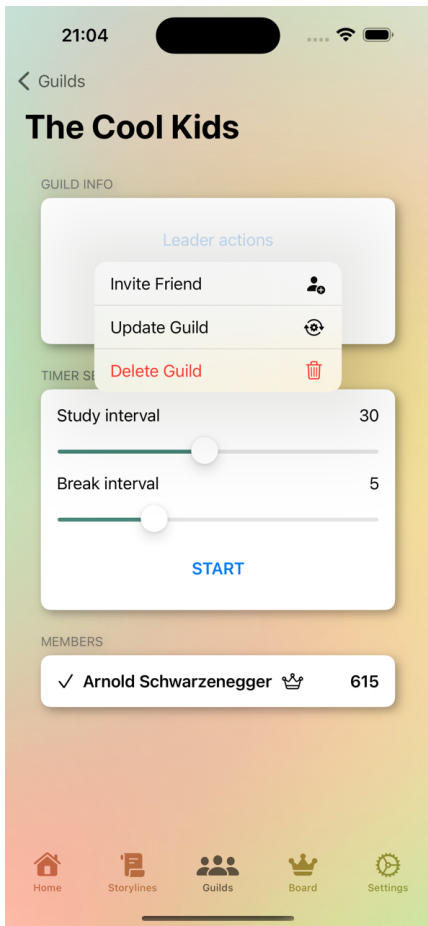


Figure D.25: Leader Actions

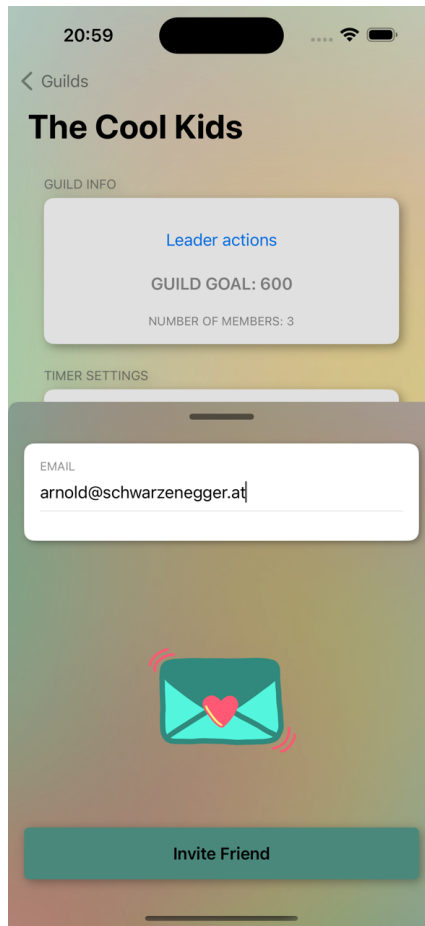


Figure D.26: Invite Friend

D. FINAL APPLICATION INTERFACE SCREENSHOTS

---

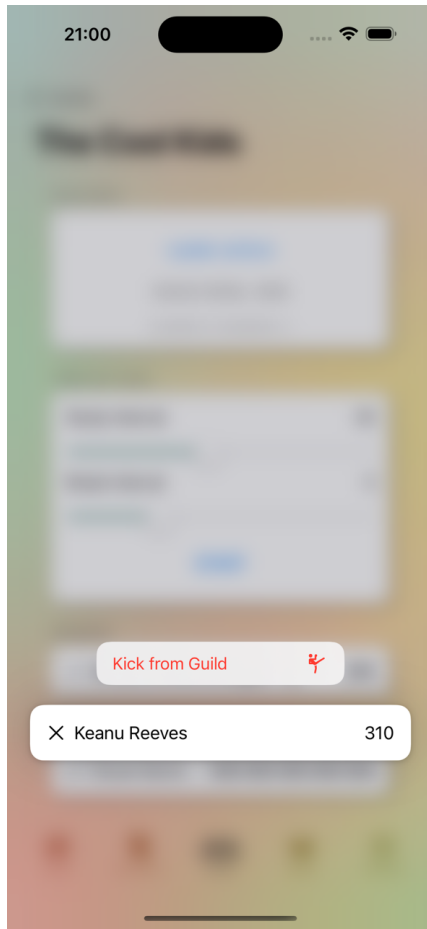


Figure D.27: Member Context Menu (Leader's Point of View)

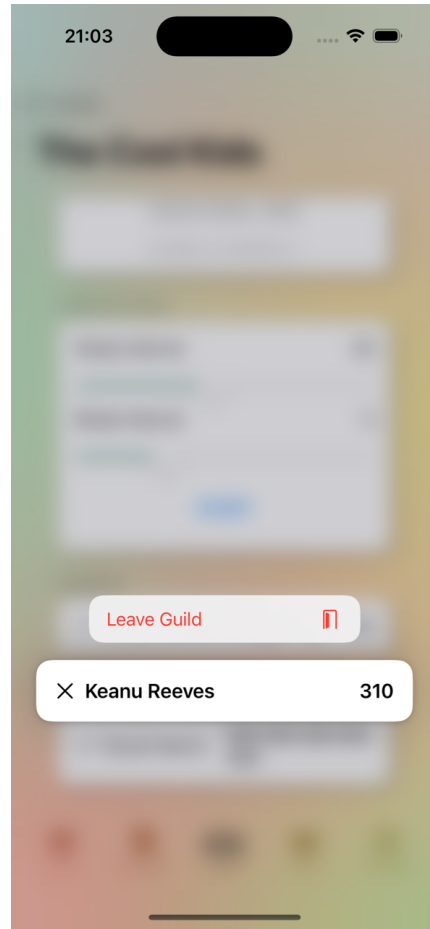


Figure D.28: Member Context Menu (Common Member's Point of View)



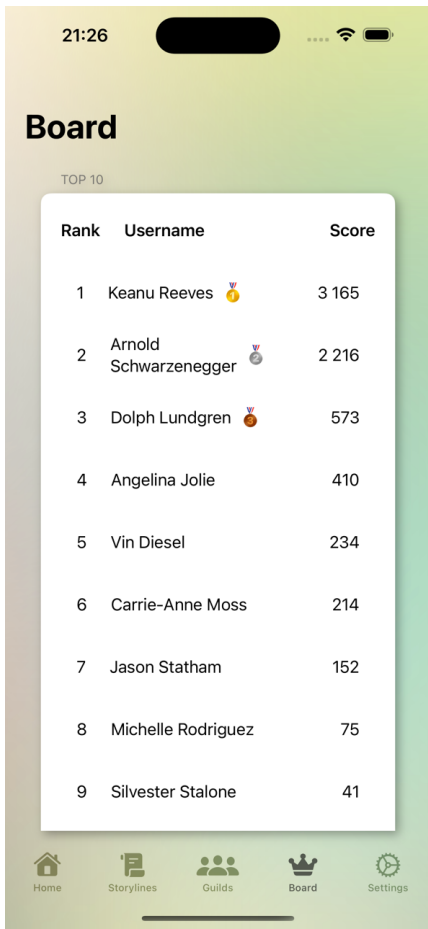


Figure D.29: Board Tab

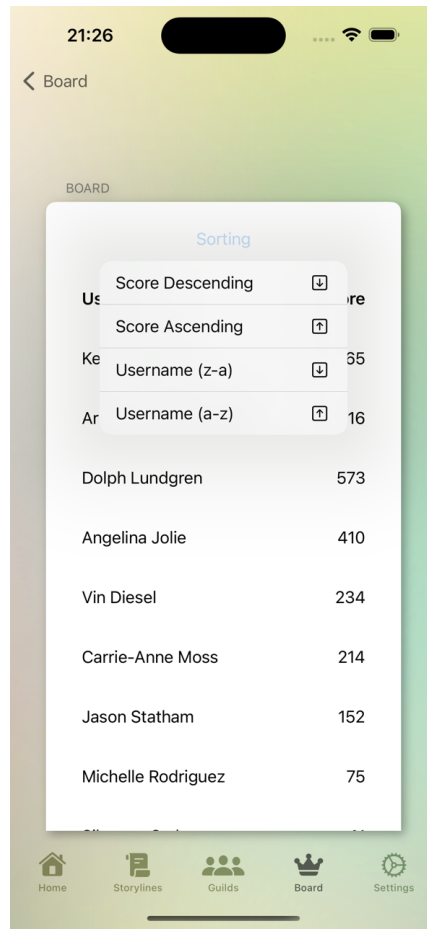


Figure D.30: Board Detail

## D. FINAL APPLICATION INTERFACE SCREENSHOTS

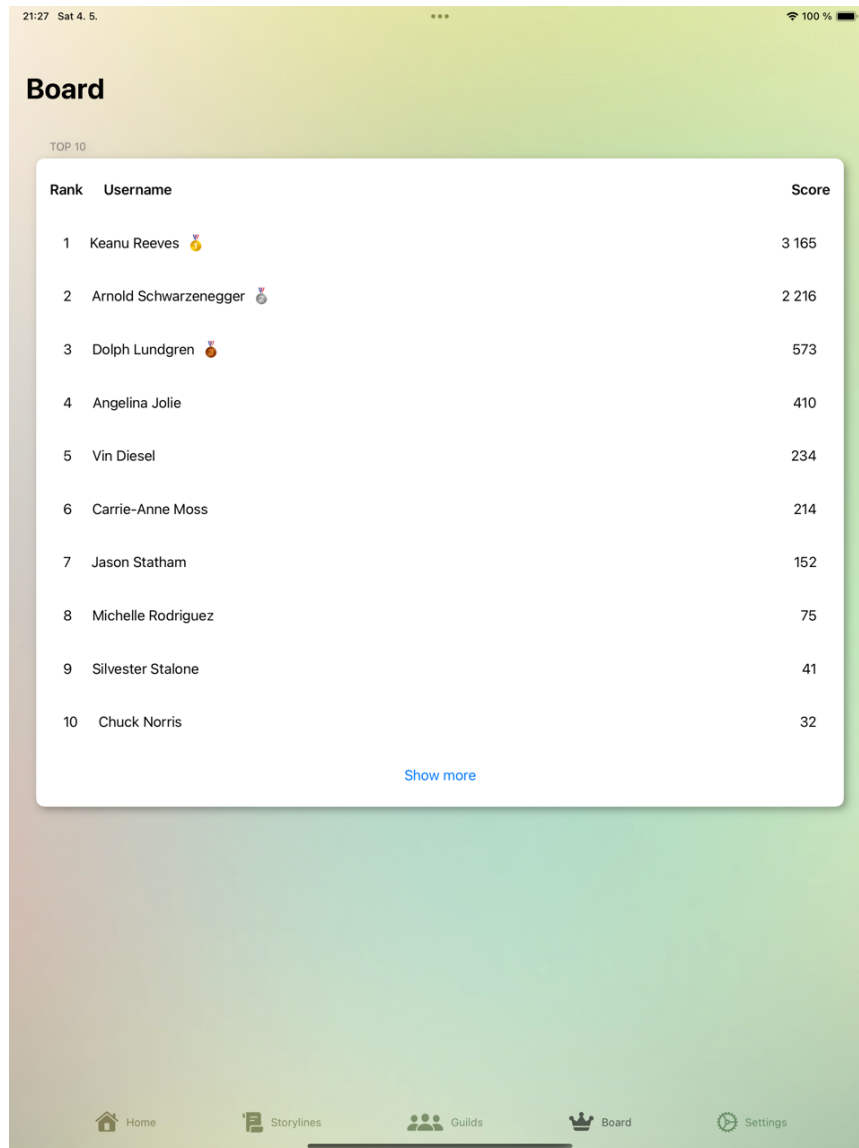


Figure D.31: Board Tab (iPad OS)

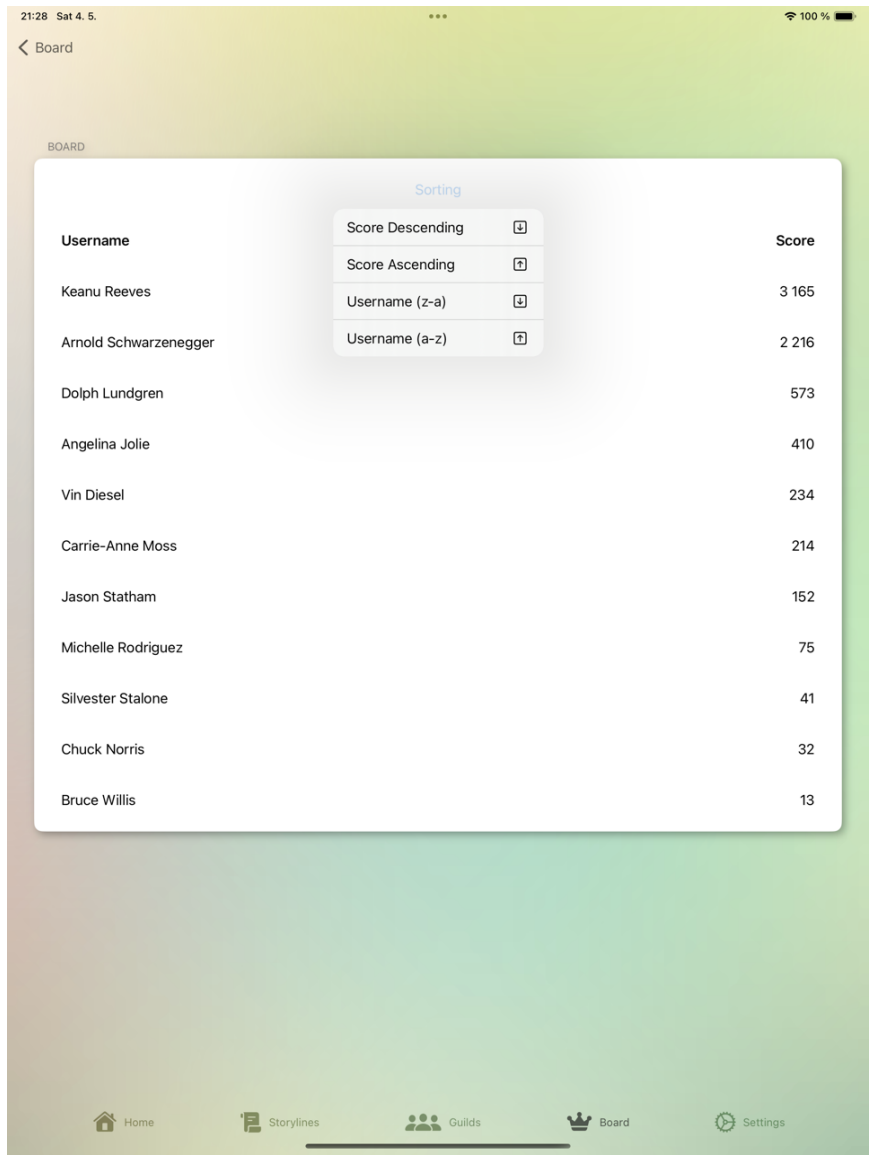


Figure D.32: Board Detail (iPad OS)

## D. FINAL APPLICATION INTERFACE SCREENSHOTS

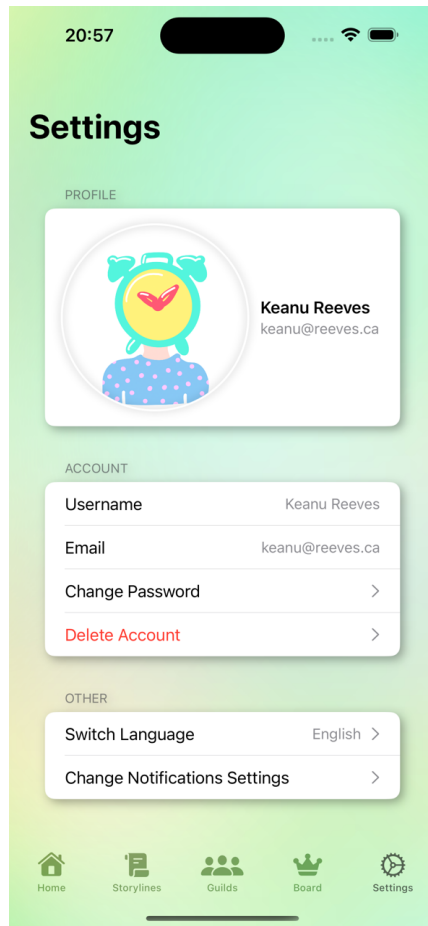


Figure D.33: Settings Tab

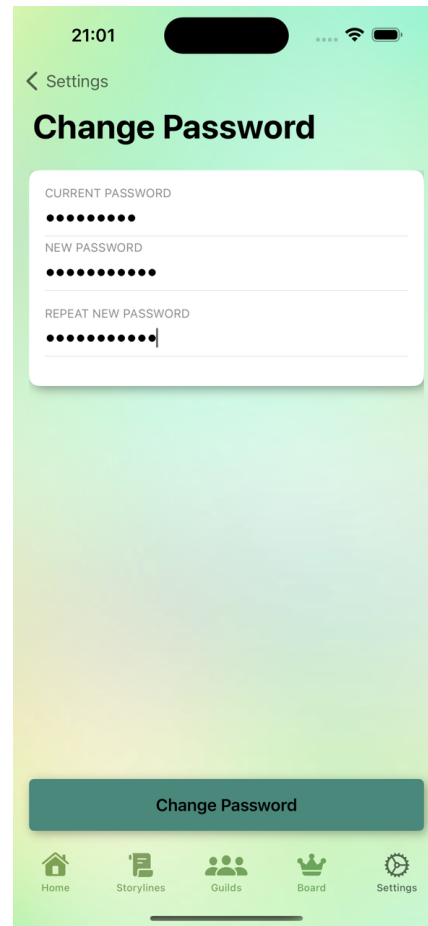


Figure D.34: Change Password

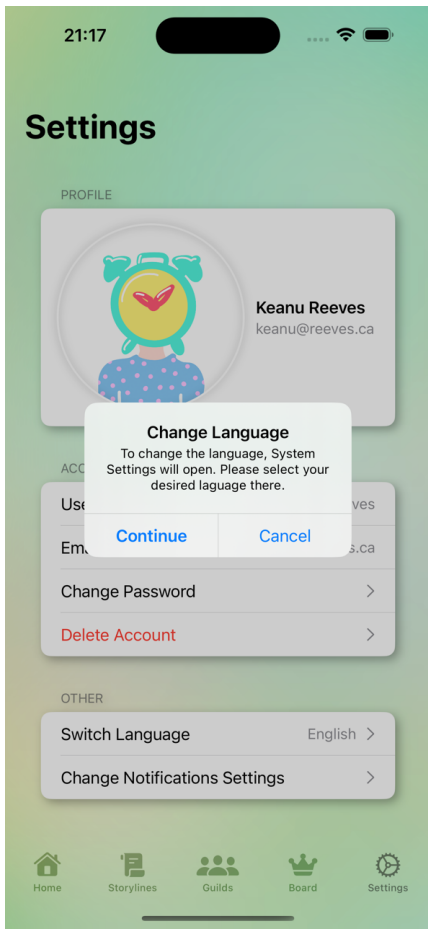


Figure D.35: Change Language

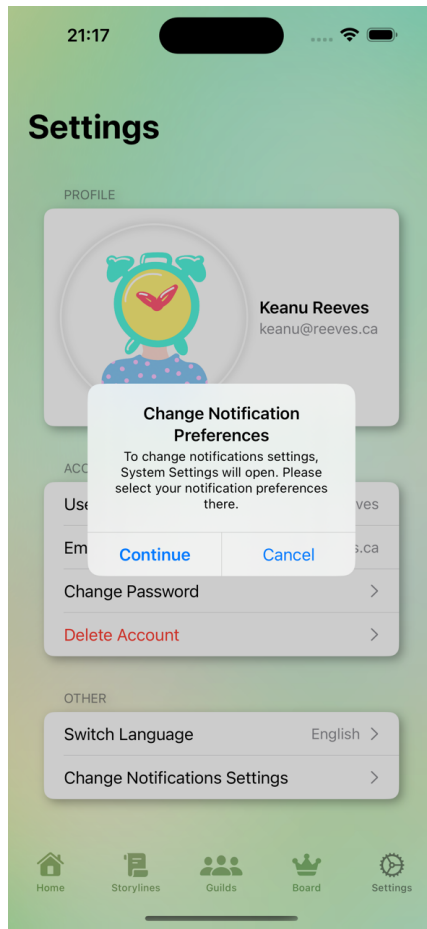


Figure D.36: Change Notifications

D. FINAL APPLICATION INTERFACE SCREENSHOTS

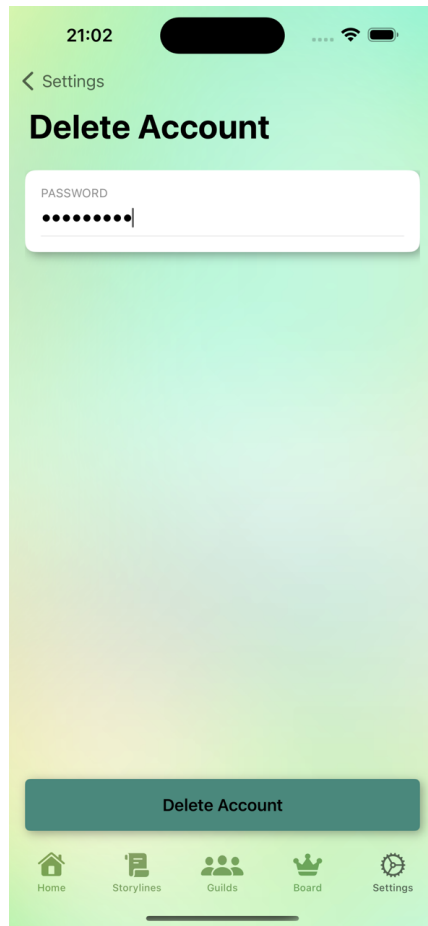


Figure D.37: Delete Account 1st

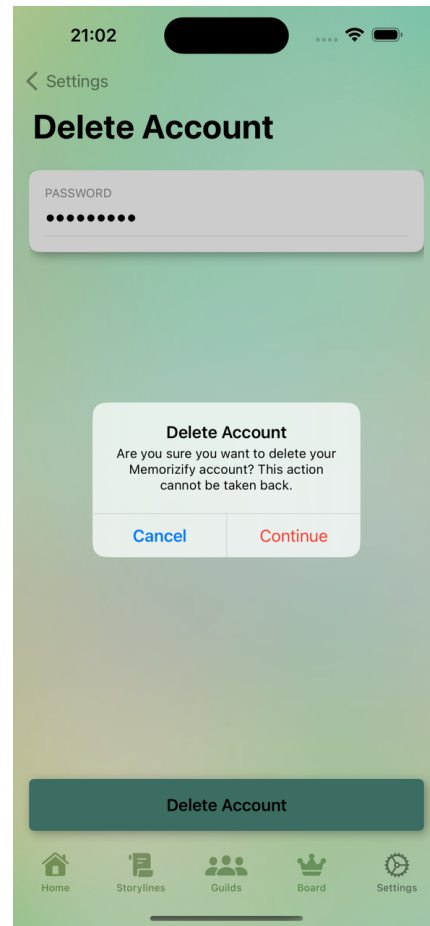


Figure D.38: Delete Account 2nd

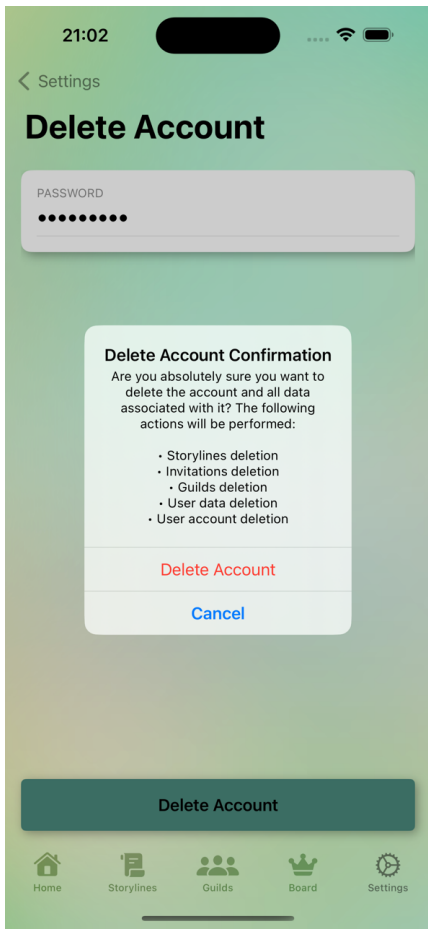


Figure D.39: Delete Account 3rd

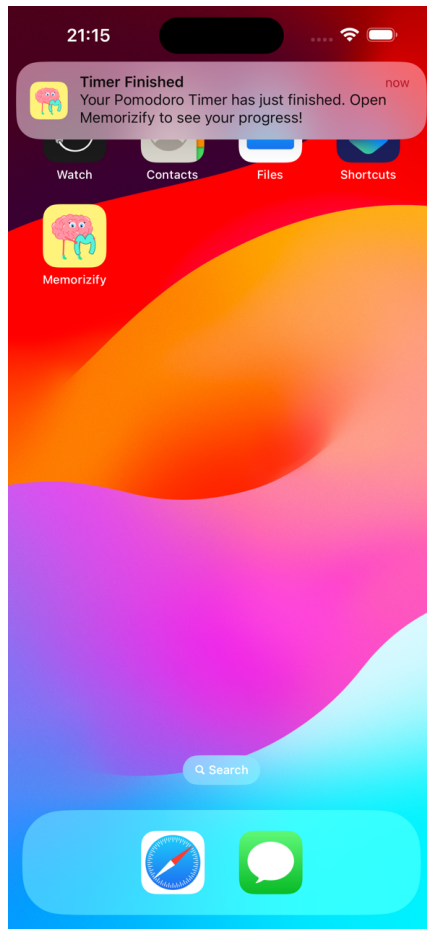


Figure D.40: Push Notification

D. FINAL APPLICATION INTERFACE SCREENSHOTS

---

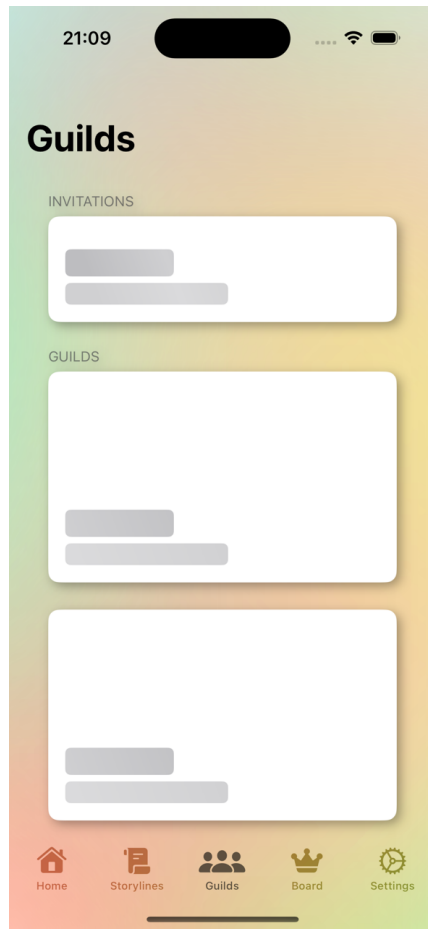


Figure D.41: Loading Placeholders

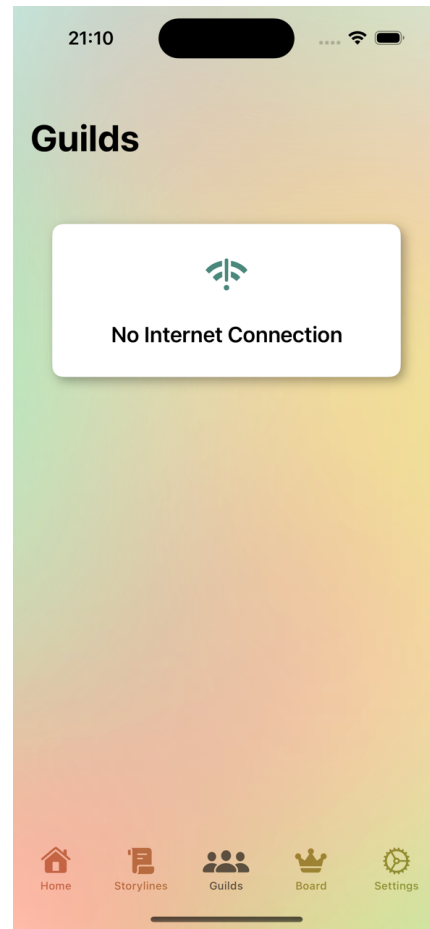


Figure D.42: No Connection



---

## Usability Testing Scenarios

### 1. Identifier: UTS1

**Name:** User Registration and Authentication

**Prerequisites:** The application is running and the initial screen is presented.

**Context:** You want to start using the application, so create a new account with *testuser* username, *testmemorizify@gmail.com* email and *ABCabc123* password.

**Success:** Successful completion of the registration and login process, granting the user access to the application.

**Expected Steps:**

- a) The user taps the *Sign Up* button.
- b) The user fills in all the required form fields.
- c) The user consents to the application's *Privacy Policy*.
- d) The user confirms registration by selecting the *Sign Up* button.
- e) The user verifies their email by accessing the email sent by the application and following the enclosed link.
- f) The user returns to the application after email verification.
- g) The user taps the *Sign Up* option.
- h) The user fills in the necessary login credentials.
- i) The user logs in by tapping the *Sign Up* button.

2. **Identifier:** UTS2

**Name:** Plain Timer Usage

**Prerequisites:** The user is logged into the application and located on the Home tab.

**Context:** You just want to start working as soon as possible, so turn on the Pomodoro timer with a 25-minute study interval and 5-minute break period.

**Success:** Successful activation of the timer with the specified settings.

**Expected Steps:**

- a) User taps the *Plain Timer* button.
- b) User configures the study interval to 25 minutes.
- c) User sets the break interval to 5 minutes.
- d) User initiates the timer by selecting the *Start* button.

---

### 3. Identifier: UTS3

**Name:** Storyline Setup and Usage

**Prerequisites:** The user is logged into the application and located on the Home tab.

**Context:** You want to enhance your motivation by introducing a storyline to read during your study breaks. Set up the *Embers Tale* storyline with a total goal of 50 hours, 30-minute study intervals, and 5-minute breaks, then proceed to start working immediately.

**Success:** Successful setup of the *Embers Tale* storyline and activation of its timer.

**Expected Steps:**

- a) User navigates to the *Storylines* tab.
- b) User selects the *Embers Tale* storyline.
- c) User chooses the *Setup Storyline* option.
- d) User sets the hour goal to 50.
- e) User sets the minute goal to 0.
- f) User configures the study interval to 30 minutes.
- g) User sets the break interval to 5 minutes.
- h) User confirms the setup by selecting the *Done* button.
- i) Upon being redirected to the *Home* tab, the user locates the *Embers Tale* storyline, accesses it, and activates the timer.

4. **Identifier:** UTS4

**Name:** Guild Creation

**Prerequisites:** The user is logged into the application and located on the Home tab.

**Context:** You aim to track study progress alongside three friends, identified by their email addresses: *chburns@email.com*, *daphchar@email.com*, and *aveburns@email.com*. Create a new guild named *School Crew* with your goal set to 40 hours.

**Success:** Successful creation of the *School Crew* guild and invitation of the three specified friends via their email addresses.

**Expected Steps:**

- a) User navigates to the *Guilds* tab.
- b) User taps on the *Tap here to create a new guild* button.
- c) User names the guild *School Crew*.
- d) User sets the hour goal to 40.
- e) User invites the three friends by their email addresses.
- f) User confirms creation by selecting the *Create Guild* button.

---

## 5. Identifier: UTS5

**Name:** Guild Members Management

**Prerequisites:** The user is logged into the application and located on the Home tab. The user is the leader of a guild called *Geek Guild*, which contains player with username *Charles Burns*.

**Context:** You have discovered that your friend *Charles Burns* is not meeting the guild's goals in *Geek Guild*. However, you know that *Lisa Montgomery* is a dedicated team player. Therefore, you intend to remove *Charles Burns* from *Geek Guild* and invite *Lisa Montgomery* to join the guild instead, using her email *lisam@email.com*.

**Success:** The user removes *Charles Burns* from *Geek Guild* and invites *Lisa Montgomery* to join the same guild.

**Expected Steps:**

- a) User navigates to the *Guilds* tab.
- b) User opens the detail of the *Geek Guild*.
- c) User performs a long press on the user *Charles Burns* in the *Members* section.
- d) After being presented context menu with possible actions, the user taps on *Kick from Guild* option.
- e) User taps on the *Leader actions* button and chooses *Invite Friend* from the context menu.
- f) User fills in the form with *lisam@email.com* and taps on *Invite Friend* button.

6. **Identifier:** UTS6

**Name:** Identifying the Player with the Fewest Points

**Prerequisites:** The user is logged into the application and located on the *Home* tab. There are more than 10 global users of the app in the user board. The user with the fewest points has the username *Charles Burns*.

**Context:** You aim to determine the user with the lowest number of points in the global rankings.

**Success:** The user identifies *Charles Burns* as the user with the fewest points in the global board.

**Expected Steps:**

- a) User navigates to the *Board* tab.
- b) User taps on the *Score*.
- c) After accessing the board details, the user taps on the *Sorting* button and selects *Score Ascending*.
- d) User locates the player *Charles Burns* at the top of the board.

---

7. **Identifier:** UTS7

**Name:** Switch Language to Czech

**Prerequisites:** The user is logged into the application and the application language is set to English. The user is located on the *Home* tab.

**Context:** You aim to demonstrate the application to your good friend, who speaks only Czech. You intend to switch the application language to Czech.

**Success:** The application language is successfully switched to Czech.

**Expected Steps:**

- a) User navigates to the *Settings* tab.
- b) User taps on the *Switch Language* setting in the *Other* section.
- c) After being presented the *Switch Language* alert, user clicks on *Continue* button to access *Settings*.
- d) User is redirected to the *System Settings* and chooses Czech language as the preferred language.
- e) User returns to the application.





---

## List of Abbreviations

- **API**: Application Programming Interface
- **ARC**: Automatic Reference Counting
- **AV**: AudioVisual
- **CLT**: Cognitive Load Theory
- **CPU**: Central Processing Unit
- **CRUD**: Create, Read, Update, Delete
- **DocC**: Documentation Compiler
- **ER**: Entity-Relationship
- **ERD**: Entity Relationship Diagram
- **FK**: Foreign Key
- **FURPS**: Functionality, Usability, Reliability, Performance, and Supportability
- **GPU**: Graphics Processing Unit
- **HTTPS**: Hypertext Transfer Protocol Secure
- **ID**: Identifier
- **IDE**: Integrated Development Environment
- **iOS**: iPhone Operating System
- **ISO/IEC**: International Organization for Standardization/International Electrotechnical Commission
- **JSON**: JavaScript Object Notation
- **MVC**: Model-View-Controller
- **MVVM**: Model-View-ViewModel

## F. LIST OF ABBREVIATIONS

---

- **MoSCoW**: Must have, Should have, Could have, and Won't have
- **NoSQL**: Not Only SQL
- **OAuth**: Open Authorization
- **PK**: Primary Key
- **PR**: Pull Request
- **SDK**: Software Development Kit
- **SDT**: Self-Determination Theory
- **SMART**: Specific, Measurable, Achievable, Relevant, and Time-bound
- **SQL**: Structured Query Language
- **UI**: User Interface
- **URL**: Uniform Resource Locator
- **UML**: Unified Modeling Language
- **UX**: User Experience
- **VIPER**: View, Interactor, Presenter, Entity, and Router
- **XML**: eXtensible Markup Language

---

## Contents of Attachments

demo.....	directory containing various demo files
├ screenshots.....	directory with screenshots of the final application
├ wireframes.....	directory with wireframes of the application
├ ui-tests.....	directory with UI test examples
src.....	directory with source files
├ implementation.....	source files for the Memorizify Xcode project
├ thesis.....	source files for the thesis LaTeX project
doc.....	directory with application documentation
├ Memorizify.doccarchive.....	DocC documentation archive
text.....	directory with the thesis text
├ thesis.pdf.....	thesis text in pdf format
other.....	additional related documents
├ research-study.pdf.....	research study on Pomodoro technique
└ readme.txt.....	readme file with description of this attachments