



Assignment of bachelor's thesis

Title:	Application for booking beauty services
Student:	Arpan Tyagi
Supervisor:	Ing. Ondřej Guth, Ph.D.
Study program:	Informatics
Branch / specialization:	Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

Study applications for searching and booking various types of services. Design and implement a client-server solution for beauty services. The application should be capable of listing services close to a given location, filtering based on various criteria, and booking selected services. Deploy the server part using services of an online cloud provider.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

Application for booking beauty services

Arpan Tyagi

Department of Software Engineering
Supervisor: Ing. Ondřej Guth, Ph.D.

May 18, 2023

Acknowledgements

I would like to express my profound gratitude to my supervisor, Ing. Ondřej Guth, Ph.D., for his invaluable guidance and enduring patience throughout the process of this work. His willingness to provide answers and clarifications to my inquiries was instrumental to my understanding and progression.

Furthermore, I would express special thanks to Nikita Dvoriadkin, who has been a close friend through this journey and provided valuable comments to improve this work. Lastly, I wish to extend my sincere appreciation to my family for their unwavering support and my girlfriend who has been a pillar of strength during this academic journey. Their understanding and encouragement have contributed significantly to the completion of this project.

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 particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 18, 2023

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2023 Arpan Tyagi. 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

Tyagi, Arpan. *Application for booking beauty services*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Tato práce popisuje návrh a vývoj prototypu vyvíjejícího se systému pro usnadnění rezervace schůzek v zařízeních kosmetických služeb. Prototypová architektura zahrnuje aplikaci pro Android a server. Celý systém je vyvíjen v souladu s nejlepšími postupy vývoje softwaru. Očekává se, že server bude nasazen v cloudovém prostředí, od kterého se očekává, že po úplné implementaci poskytne optimální škálovatelnost a udržitelnost.

Výsledkem této práce je vytvoření prototypu systému, který by měl sloužit jako základní platforma pro budoucí vylepšení. Očekává se, že tento prototyp připraví cestu pro komplexní vývoj a případné veřejné vydání produktu. To představuje první, ale zásadní krok na širší cestě uvedení tohoto inovativního řešení na trh

Klíčová slova Rezervace, klient-server, Android, Kotlin, Heroku, Ktor

Abstract

This thesis outlines the design and prototype development of an evolving system aimed at facilitating the booking of appointments at beauty service establishments. The prototype architecture encompasses an Android application and a server. The entire system is developed adhering to software development

best practices. The server deployment is envisaged within a cloud environment, which is expected to ensure optimal scalability and maintainability once fully implemented.

As an outcome of this thesis, a prototype system has been established that is intended to serve as the foundational platform for future enhancements. It is anticipated that this prototype will pave the way for comprehensive development and eventual public release of the product. This represents an initial, yet critical, step in the broader journey of bringing this innovative solution to market.

Keywords Booking, client-server, Android, Kotlin, Heroku, Ktor

Contents

Introduction	1
1 Aims and Objectives	3
1.1 Specific Objectives	3
1.2 Conclusion	4
2 Market Research	5
2.1 Review of existing solutions	5
2.1.1 Booksy	5
2.1.2 Fresha	6
2.2 Possible Improvements.	7
2.3 Conclusion	8
3 Analysis	9
3.1 User Roles	9
3.2 Requirements	10
3.2.1 Functional Requirement	10
3.2.2 Non-Functional Requirement	11
3.3 Use cases	11
3.4 Domain model	11
4 Assessing the Tech Stack Choices	15
4.1 Development of Client	15
4.1.1 Native vs Cross-Platform Development	15
4.1.2 Mobile Platform Selection for Prototype Development	16
4.1.3 Choice of Programming Language	17
4.2 Selection of Server Framework	19
4.2.1 Spring Framework	20
4.2.2 Ktor	21
4.2.3 Final Decision	21

5 Design	23
5.1 Wireframes	23
5.2 Client Server Architecture	24
5.3 API design	26
5.4 System Architecture	26
5.4.1 User Authentication	26
5.5 Final Architecture	28
5.6 Android Design	28
5.6.1 Application architecture	29
5.6.2 UI layer	29
5.7 Server design	30
6 Implementation	31
6.1 Android Implementation	31
6.1.1 Libraries	31
6.1.1.1 Jetpack Compose	31
6.1.1.2 Hilt	31
6.1.1.3 Compose Destinations	32
6.1.1.4 Retrofit	32
6.2 Server Implementation	33
6.2.1 Libraries	33
6.2.1.1 Ktor	33
6.2.1.2 Exposed	33
6.2.2 Image Storage	34
6.2.3 Database	34
6.3 Deployment	34
6.3.1 Cloud infrastructure	34
6.3.2 Android deployment	35
7 User Testing, Feedback and Future Improvements	37
7.1 User testing	37
7.2 Feedback	38
7.2.1 Summary	40
7.3 Future improvements	40
Conclusion	43
Bibliography	45
A Acronyms	47
B Contents of enclosed CD	49

List of Figures

3.1 General use cases	12
3.2 Domain model	13
5.1 Login and Sign-up wireframes	23
5.2 Business list wireframe	24
5.3 Business details view wireframe	25
5.4 Create bookings and list bookings wireframes	25
5.5 Recommended architecture by Google	29
6.1 Database model	35
6.2 Cloud infrastructure	36

Introduction

Growing up in the suburbs of Gurugram, India, the simplicity of life was mirrored in our routine visit to the local barber shop. It was a stone's throw away from our home, and my father knew the barber personally. We would leisurely wait our turn, flipping through magazines, a practice that became as much a part of the haircut experience as the haircut itself.

As I transitioned into adulthood and moved away from home, the task of finding a new barber in a sea of choices was not as straightforward as it once seemed. Navigating the options listed on Google, making sense of the ratings, and then managing to book an appointment was unexpectedly challenging. I found that appointment booking methods were primarily split into two categories: calls and online reservations, and neither seemed ideal.

Making phone calls was restricted to business hours and often involved tedious back-and-forths to align schedules. On the other hand, online reservations, while seemingly convenient, were marred by clunky and slow user interfaces that were anything but user-friendly.

It was during this quest for a new barber that I realized the need for a unified, seamless platform that could bring together all the salons in a city. A platform that would not only let users sift through various establishments but also compare them based on reviews, ratings, and prices, and effortlessly book appointments. A solution like this was conspicuously absent in the Czech Republic, and thus, the seed for my project was sown.

Aims and Objectives

The primary objective of this thesis is to design and develop a prototype solution for discovering and booking beauty services in an easy and efficient manner. The system aims to facilitate browsing a selection of available beauty businesses within a given city, allowing users to filter and sort establishments based on various criteria.

The resulting prototype system should comprise two main components: a server and a mobile client application. The server is responsible for handling data storage, retrieval, and processing, while the mobile client provides users with an accessible and intuitive interface for interacting with the system. Both components should be designed using modern software development best practices to ensure scalability, maintainability, and robustness.

1.1 Specific Objectives

1. Undertaking a meticulous analysis of existing beauty service booking solutions to ascertain their strengths and weaknesses, thus gaining valuable insights into the current market landscape.
2. Architecting a client-server system that supports a diverse range of functionalities, including searching, filtering, booking, and managing bookings, catering to the varying needs of users.
3. Engineering a mobile application prototype that enables users to interact seamlessly with the system, prioritizing quick scheduling of bookings.
4. Implementing a server component that capably manages data storage, retrieval, and processing information.
5. Deploying the server component on a cloud infrastructure to guarantee scalability and ease of maintenance, anticipating future growth and expansion of the system.

1.2 Conclusion

In fulfilling these objectives, the overarching aim of this thesis is to construct an inclusive prototype system that addresses the complex task of locating and scheduling appointments with beauty service establishments. This prototype is set to incorporate a selection of anticipated features indicative of a fully developed product. It establishes a solid foundation that paves the way for subsequent enhancement and fine-tuning, ultimately leading to a robust product ready for public release.

Market Research

This chapter explores the existing landscape of booking systems that enable users to secure bookings at nearby businesses that offer beauty services. A thorough examination of the strengths and weaknesses of prominent systems in the market will be conducted, identifying potential opportunities.


To begin with the current market analysis I will shortlist a list of systems already offering the service of booking beauty services. The process for selecting potential reservation systems to analyze will involve browsing the Google Play Store. We'll be focusing on applications with more than one hundred thousand downloads and a user review rating of at least 4.5 stars. This approach will generate a comprehensive list of the current leading apps in the market. Subsequently, these applications will be evaluated for their strengths and weaknesses, providing valuable insights into the landscape of reservation systems. Since the Apple app store doesn't publish the number of times an app has been downloaded, I will rely on the numbers from the Google Play store for the number of downloads.

2.1 Review of existing solutions

2.1.1 Booksy

Downloads: 10 Million +

Rating: 4.9

Booksy  is a platform designed to facilitate bookings at nearby businesses. As a mobile app and online platform, Booksy enables users to schedule appointments with local beauty and wellness professionals, including hairstylists, nail technicians, and massage therapists. The app offers features such as real-time availability, booking confirmations, and appointment reminders. Additionally, users can browse and search for service providers based on location,

2. MARKET RESEARCH

specialty, and availability, as well as view ratings from other users to support informed decision-making.

Strengths

- Booksy is a large business and operates in six global markets - the US, UK, Poland, Spain, Brazil and South Africa. The company claims to have 13 million active users. [2]
- Mobile and web presence: The Booksy app is available on both iOS and Android platforms, in addition to their web-based platform, providing flexibility and convenience for users.
- Marketing tools for businesses: Booksy provides businesses with tools for promotions, discounts, and loyalty programs, helping them attract and retain customers.

Weakness

- Limited geographical coverage: Booksy is not available in all regions or cities, potentially restricting its user base. Notably, Booksy is not available in the Czech Republic.
- Subscription-based pricing for businesses: The subscription pricing model may deter smaller businesses or those with limited budgets from using the platform.
- Dependency on the network effect: Booksy's success relies on attracting both businesses and customers to its platform, which may be challenging in areas with low adoption rates.

2.1.2 Fresha

Fresha [3] is an online platform and app that provides free business software for salons, spas, gyms, and other wellness businesses. It allows these businesses to manage their scheduling, point-of-sale, inventory, and other operational needs. For customers, it provides an easy way to discover, book, and pay for services.

Downloads: 100 Thousand +

Rating: 4.7

Strengths:

- Fresha operates in more than 120 countries, making it a widely accessible platform for users and businesses.[\[4\]](#)
- The platform is user-friendly and supports both web and mobile applications, available on iOS and Android, providing flexibility and convenience for users.
- Fresha offers a range of marketing tools and business management features for service providers, such as inventory management, staff management, and reporting.
- Integration of customer reviews and ratings helps users make informed decisions while selecting service providers.
- Fresha offers a free booking platform for businesses, making it an attractive option for small businesses and those with limited budgets.

Weaknesses:

- As a global platform, Fresha may face challenges in providing localized support or addressing region-specific requirements.
- In areas with low adoption rates, users may find limited options for service providers, affecting the platform's utility.

Out of the numerous mobile applications analyzed for booking appointments at beauty businesses, only two met the selection criteria of having over 100,000 downloads and a user rating of 4.5 or higher on the app store. These applications are Booksy and Fresha, which have demonstrated a significant user base and positive reviews. Although there were two other applications that facilitated appointment booking at nearby beauty businesses, they had less than 100,000 downloads, indicating a smaller market presence. To maintain a focused analysis, the decision was made to concentrate on the applications with higher downloads and user ratings. This approach ensures that the most prominent and successful applications in the market are evaluated, providing valuable insights into their features, strengths, and weaknesses.

2.2 Possible Improvements.

An examination of the strengths and weaknesses of the identified platforms reveals several opportunities for enhancing the technical implementation of the proposed product. To address the gaps in existing solutions and differentiate the new product, the following measures can be taken:

1. **Localized focus:** The proposed product will cater specifically to the Czech market, with an emphasis on Prague and other Czech cities. This approach differentiates it from the U.S.-based solutions, which have limited presence in the region. Focusing on the local market allows the application to cater to the unique needs of Czech users and businesses.
2. **Limited but focused feature set:** Unlike existing applications that offer a multitude of features, some of which may not be useful to users (e.g., Booksy's articles and blog posts), the proposed product will prioritize a streamlined prototype with a limited, yet focused, feature set. The primary goal is to facilitate quick and easy bookings for users, eliminating extraneous features that may detract from the core functionality of the reservation system. This approach aims to improve overall user satisfaction and efficiency in booking beauty services.

2.3 Conclusion

In conclusion, the comprehensive analysis of existing reservation systems has yielded valuable insights into the market landscape, user-interface design, and best practices for developing a booking application. This knowledge has served as a foundation for the design and development of a solution for reserving beauty services, targeting specifically the Czech market. By creating a streamlined prototype with a focused feature set, the proposed application aims to address the unique needs and preferences of users within this region. As a result, the developed prototype application is poised to establish a competitive advantage within the Czech market.

Analysis

In this chapter, the objective is to build upon the research conducted in the previous chapter by identifying potential user roles within the application. This will involve the formulation of functional and non-functional requirements, as well as the definition of specific use cases. The intent is to provide a comprehensive understanding of the system's requirements, ensuring that the proposed solution effectively addresses the needs of its users in a systematic and well-structured manner.

3.1 User Roles

In this system designed for finding and booking beauty services, there will be several user roles, with distinct privileges and responsibilities. Some common user roles in such a system could include:

1. **Customer:** Customer is the primary users of the system. They can browse and filter through available beauty businesses, view detailed information about the services offered and finally make bookings. They can also leave reviews and ratings.
2. **System Administrator:** System administrators are responsible for managing the overall platform. They can oversee and moderate user accounts (both customers and business administrators), monitor and enforce compliance with the platform's terms and policies, handle disputes and support requests, and maintain the platform's overall functionality, security, and performance.
3. **Business Administrator(out of scope):** Business administrators represent the beauty businesses within the system. They have the ability to see and manage their business profile, including uploading images, providing descriptions, and listing services and their respective prices. Business administrators can also manage their schedules, respond to

customer reviews on their business, track customer history, and view customer reviews and ratings

In this thesis, the primary objective is to develop a customer-facing mobile application for discovering and booking beauty services. By focusing on a mobile application explicitly designed for customers, the user experience can be more targeted, ensuring that features and functionalities cater to the unique requirements of customers seeking and scheduling beauty services. By concentrating on the customer-facing mobile application, this thesis strives to deliver an optimized solution that addresses the core aim of simplifying the process of finding and booking beauty services for customers. The development of a separate business-facing application or web interface, tailored to the specific needs of beauty businesses, will be deemed beyond the scope of this thesis.

3.2 Requirements

This section will describe the functional and non-functional requirements. These requirements were formulated based on the goals of this thesis and modern standards of software development.

3.2.1 Functional Requirement

- F1: Browse Business and Services offered by them.** The user should be easily able to browse businesses offering his desired services near him. Moreover, he should be able to filter them through some criteria, like location proximity or business rating.
- F2: User Sign up and Log In** The user should be able to create a new account or sign in to his existing account.
- F3: Booking Management** The user should be able to create, update and delete bookings created by him. A user should be able to easily browse through his bookings and manage them.
- F4: Browse Reviews** A user should be able to read reviews and see ratings for a business that have been written by other users.

While a comprehensive system should ideally provide numerous features catering to various aspects of the business, it is essential to acknowledge the distinct requirements of businesses compared to those of general users, as discussed in section [3.1](#). The unique needs of business users warrant the development of a separate application tailored to their specific demands. For the sake of conciseness, this thesis will focus on MVP features intended for general users and the development of a user-centric application. A dedicated business application would serve as an advantageous extension of this thesis, warranting further exploration in future research.

3.2.2 Non-Functional Requirement

NF1: Client. A client application that can run on a modern smartphone.

NF2: Server A Server that can communicate with the mobile client.

NF3: Localization Application should be in English, although the client should be developed in such a way that languages can be quickly be added in the future.

NF4: Extensibility Because the goal of this thesis is to implement a prototype of client and server application, the application itself must be extensible, that is, it must provide the ability to easily add new features or extend current ones.

3.3 Use cases

This section describes the use cases of the application.

UC1: Log in Anonymous user should be able to log into the application using their email and password.

UC2: Register Anonymous user should be able to register and create an account in the system.

UC3: Browse Businesses Registered user should be able to browse available business in his location, he should be able to filter the listed businesses based on some criteria.

UC4: Browse Services offered by a business Registered user should be able to select a listed business and be able to look at services offered by the business.

UC5: Browse Reviews of a business Registered user should be able to select a listed business and read reviews left by other users.

UC6: Create Bookings Registered user should be able to create a booking at a business listed on the system.

UC7: View Bookings Registered user should be able to see all of their present and past bookings.

UC8: Filter Businesses Registered users should be able to filter through businesses based on their location and their ratings.

3.4 Domain model

Analysis of the requirements and use cases, which were described before, has led to the creation of the domain model, shown in the Figure [3.2](#).

3. ANALYSIS

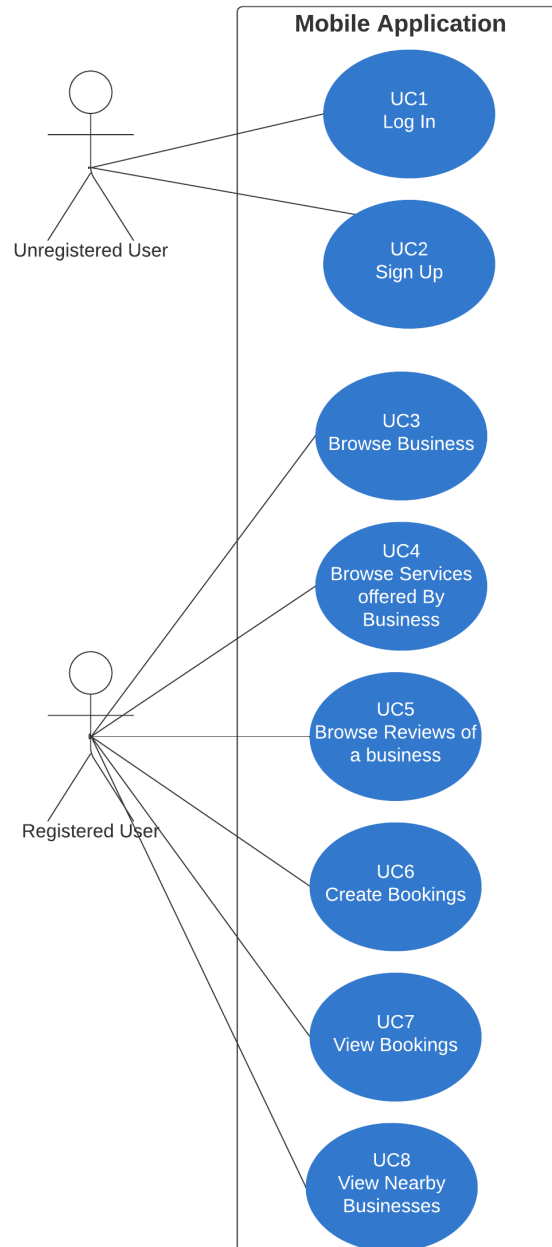


Figure 3.1: General use cases

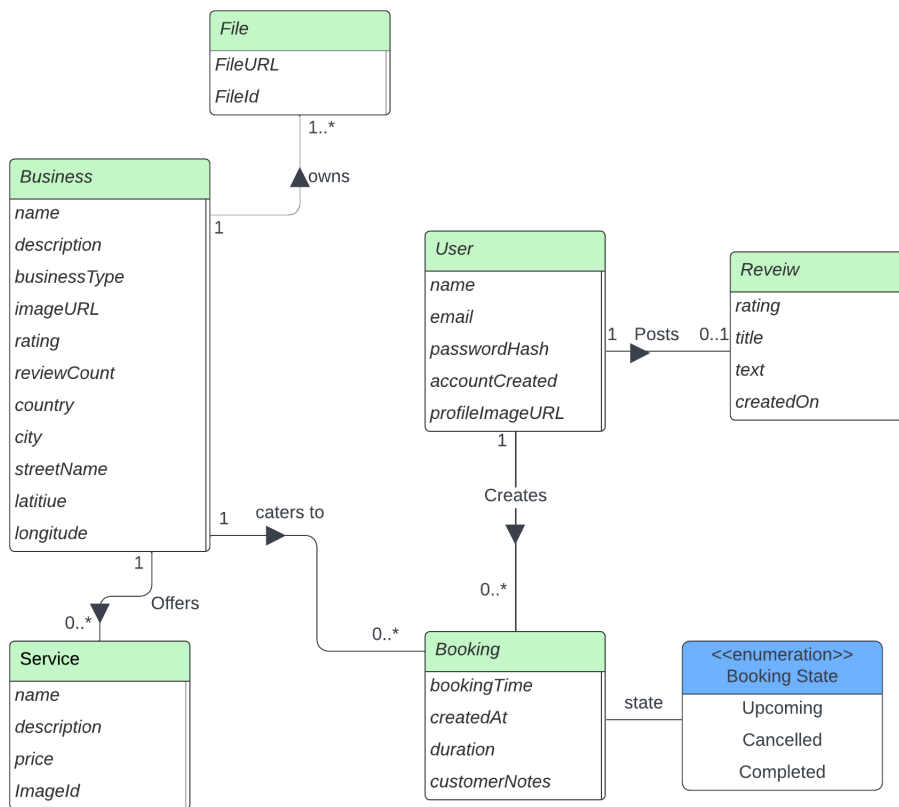


Figure 3.2: Domain model

Assessing the Tech Stack Choices

4.1 Development of Client

When making mobile application there are two paths which a developer can take either Native Mobile Development or Cross-Platform development. The next sections tries to analyze the advantages and disadvantages of both.

4.1.1 Native vs Cross-Platform Development

The following section is based on [\[5\]](#)

When making a mobile applications, there are two paths that a developer can take: either Native Mobile Development or Cross-Platform Mobile Development. Native Mobile Development refers to the process of building an application specifically for a single platform, like Android or iOS, using their respective programming languages - Kotlin/Java and Swift/Objective-C. These applications are optimized for the operating system they're built for, providing a high performance and seamless user experience that aligns with the platform's guidelines.

On the other hand, Cross-Platform Development involves creating applications that can operate on multiple platforms, usually both Android and iOS, from a single codebase. This is made possible by frameworks such as React Native, Flutter, or Xamarin. While this approach may increase development speed and reduce costs by reusing code, it might not always be able to provide the same level of performance and native-like user experience.

The next sections attempt to analyze the advantages and disadvantages of both Native Mobile Development and Cross-Platform Development.

Performance: Native mobile applications generally offer superior performance compared to cross-platform applications, as they are optimized

for specific platforms and compiled using the platform's core programming language and APIs. Cross-platform applications may experience reduced performance due to the additional layer of computation introduced by the custom runtime.

Access to Platform-Specific Features: Native applications can access the full feature set of the device, providing a richer and more integrated user experience. Cross-platform frameworks may lag in integrating the latest platform-specific features and updates, potentially limiting the application's capabilities. For our prototype this is useful, since we want to quickly access the device's location.

Security: Native mobile applications are more secure than their cross-platform counterparts, as they have access to the built-in security features of the device's operating system. Cross-platform applications may not provide the same level of security.

Development Costs and Time: Although cross-platform development can save time and resources by sharing code between platforms, the associated complexities and limitations may offset these benefits. Native development ensures a more streamlined development process and easier maintenance due to the singular focus on a specific platform.

Upon evaluating these factors, it is clear that native mobile development presents several advantages for the prototype application, including optimal performance, access to platform-specific features and a streamlined development process. While cross-platform development offers advantages such as reduced development costs and rapid deployment, the potential drawbacks in terms of performance, feature availability, and user experience make native development a more suitable choice for the prototype application. Therefore, after carefully considering the pros and cons of both native mobile development and cross-platform tools, the decision has been made to adopt native mobile development for the prototype application.

4.1.2 Mobile Platform Selection for Prototype Development

For the prototype, given the time constraints, it has become necessary to develop a client for only one of the two popular mobile platforms: Android or iOS. The decision will be based on the following factors:

- 1. Market Share:** Android holds a significantly larger market share compared to iOS. By targeting Android, the application has the potential to reach a wider audience, which could contribute to an increased user base and overall success. In the Czech Republic, the market share of Android stands at 71.83%, whereas iOS holds 27.48% (as of April 2023) [6].

- 2. Development Language** The choice of development language for the prototype can influence the decision between Android and iOS platforms. If the individual developer possesses proficiency in Kotlin or Java, Android may emerge as a more suitable choice. On the other hand, if the developer demonstrates expertise in Swift or Objective-C, selecting iOS could be a more appropriate fit.
- 3. Development Cost and Time** Android development can prove to be more time-consuming and costly due to device fragmentation. This requires the developer to ensure compatibility across a diverse range of devices and screen resolutions. Conversely, iOS development is generally faster and less expensive because of the limited number of devices and hardware configurations.

Upon careful evaluation of these factors, it is clear that selecting a mobile platform that aligns with my expertise will facilitate faster prototype development. Moreover, considering the market share advantage, it is prudent to opt for the Android platform for the prototype development. This decision not only leverages my familiarity with the tech stack but also capitalizes on Android's dominant market share.

4.1.3 Choice of Programming Language

In the realm of Android application development, there are two primary programming languages to consider: Java and Kotlin. Both languages offer unique advantages and disadvantages that must be carefully weighed before making a decision. Java, as the original language for Android development, has established a robust ecosystem, extensive libraries, and a large developer community. However, Kotlin gained official support from Google in 2017 and became Google's recommended language for Android development in 2019. Most importantly it offers interoperability with Java. [\[7\]](#)

Advantages of Java

1. Mature ecosystem: Java has been the primary language for Android development since its inception, leading to a rich ecosystem of libraries and resources.
2. Large developer community: Java's longstanding presence in the Android development landscape has resulted in a vast community of developers who can provide support and share knowledge.

Advantages of Kotlin

1. Concision: Java's syntax is verbose compared to Kotlin, which can make code more difficult to read and maintain.

4. ASSESSING THE TECH STACK CHOICES

- a) Kotlin does not require a class declaration for the main method, making the code more concise.
 - b) Kotlin uses 'val' and 'var' for declaring immutable and mutable variables, respectively, simplifying the syntax.
 - c) Kotlin supports default arguments and named parameters, reducing the need for method overloading and leading to more concise code.
 - d) Kotlin has built-in support for coroutines, which allow for easier asynchronous programming, without requiring verbose callback structures like those often found in Java.
 - e) Kotlin has powerful type inference while Java has limited type inference capability. This means that in Kotlin, you often don't have to explicitly specify the type of every variable you declare, leading to more concise code.
2. Null Safety: Kotlin's null safety is one of its most significant advantages over Java. It helps prevent null pointer exceptions, which are common errors in Java. Listing [4.1](#) demonstrates the null safety of Kotlin over Java.
- a) In Kotlin, by default, variables cannot hold a null value. To declare a nullable variable, you must use the "?" modifier (e.g., "String?"). This explicit declaration makes it clear which variables can be null, leading to fewer null-related errors.
 - b) The safe call operator "?." in Kotlin allows you to call a method or access a property on a nullable type without risking a NullPointerException. If the variable is null, the entire expression evaluates to null.
 - c) Kotlin provides the Elvis operator ("?:") for providing a default value if a nullable expression evaluates to null
3. Compatibility: Kotlin is designed to be fully interoperable with Java, allowing developers to utilize existing Java libraries and frameworks while gradually transitioning to Kotlin. This compatibility ensures a smooth migration and enables Kotlin code to coexist with Java code in the same project. Kotlin

Taking into account the various advantages of Java and Kotlin, it becomes clear that Kotlin is the better choice for modern Android application development. Kotlin's concise syntax, improved null safety, and full interoperability with Java make it an attractive option for developers seeking a more efficient and enjoyable programming experience. Furthermore, with Google's

Listing 4.1: Kotlin vs Java conciseness

```
//Java
public class NullSafety {
    public static void main(String[] args) {
        String str = null;

        // This will throw a NullPointerException
        int length = str.length();
    }
}

//Kotlin
fun main() {
    var str: String? = null

    // This won't compile, preventing a NullPointerException
    // Exception
    val length = str.length

    // To safely call a method on a nullable type,
    // use the safe call operator
    val length = str?.length
}
```

official support and recommendation for Kotlin as the preferred language for Android development, the Kotlin ecosystem and community are expected to continue growing, leading to more resources, libraries, and tools tailored to the needs of Android developers.

In conclusion, Kotlin offers a more modern, expressive, and safer approach to Android development while maintaining compatibility with Java's extensive ecosystem. Adopting Kotlin for Android development is a forward-thinking decision that is likely to benefit developers and their projects in the long run.

4.2 Selection of Server Framework

When selecting a server framework for your application, it is essential to consider various factors such as performance, ease of use, and compatibility with the chosen programming language. Popular server frameworks include Node.js, Django, Ruby on Rails, and Express.js, among others. Each of these frameworks has its strengths and weaknesses, and your choice should

ultimately be driven by your application's specific requirements and your familiarity with the underlying technology.

In order to maintain consistency throughout the development process, it is advantageous to select a server framework that supports Kotlin as the primary development language. By utilizing Kotlin across both client and server sides, developers can leverage their existing knowledge and expertise, streamlining the development process and enhancing overall productivity. This consistency in the technology stack will enable seamless integration and facilitate easier maintenance of the entire system. Moreover, the unified language choice minimizes the learning curve associated with mastering multiple languages and frameworks, allowing me to focus on creating a high-quality, coherent application.

After applying the filter of server framework supporting Kotlin as development language I have found two popular frameworks for my system Spring and Ktor.

When comparing the server frameworks Spring and Ktor for Kotlin-based development, several pros and cons emerge for each option. An analysis of these factors can guide the decision-making process for selecting the most suitable framework for my application.

4.2.1 Spring Framework

The Spring Framework, first released in 2004, is a robust tool utilized predominantly for Java application development. Its primary objective is to streamline server development, providing pre-made solutions to common challenges including security, dependency injection, and testing. This comprehensive framework, however, carries a certain complexity due to its expansive nature, which can make its mastery a challenging endeavor. Thus, while it offers numerous advantages, its sheer size could potentially be perceived as a drawback, especially for those seeking to fully comprehend its extensive capabilities.

Advantages:

- Since being first released in 2004, the framework has matured and established a large community.
- Comprehensive documentation and support resources.
- Wide array of tools and integrations.

Disadvantages:

- A steeper learning curve for developers new to the framework.
- Relatively higher resource consumption and larger a footprint.
- Doesn't support Kotlin coroutines, since the framework is Java First.

4.2.2 Ktor

Ktor, a relatively novel framework dedicated to server creation, is currently undergoing active development. Distinguished by its "Kotlin-first" approach, Ktor leverages advanced features of the Kotlin language, such as its succinct syntax and support for coroutines. However, as an emergent framework, Ktor's ecosystem of plugins and libraries is comparatively limited, and it may not offer pre-built solutions for certain problems. This potential drawback necessitates that developers using Ktor be prepared to devise their own solutions for any issues not addressed by the existing framework.

Advantages:

- Designed specifically for Kotlin, ensuring seamless integration and compatibility.
- Lightweight and relatively easy to learn.
- Simple and intuitive API, leading to a shorter learning curve.

Disadvantages:

- Smaller community and fewer resources compared to Spring.
- Since the framework is very new, it might be difficult to find support, if some roadblocks are encountered.

4.2.3 Final Decision

Both frameworks offer distinct advantages and disadvantages, In light of the comparison, Ktor emerges as the preferred choice for the application, primarily due to its design focus on Kotlin, lightweight nature, and simplicity. These advantages align with the goal of maintaining consistency in the technology stack and enable me to create an efficient, high-performing application with ease.

Design

5.1 Wireframes

To facilitate a comprehensive understanding of the application's interface and to aid in designing the server requests, wireframes have been prepared. These wireframes, which serve as visual blueprints of the application's screens, are grounded in the use cases identified during the analysis phase. This approach ensures that the design and functionality of the application align with its intended user interactions and requirements.

Figure 5.1 demonstrates the screens related to sign in and sign up scenarios, and covers use cases UC1 and UC2.

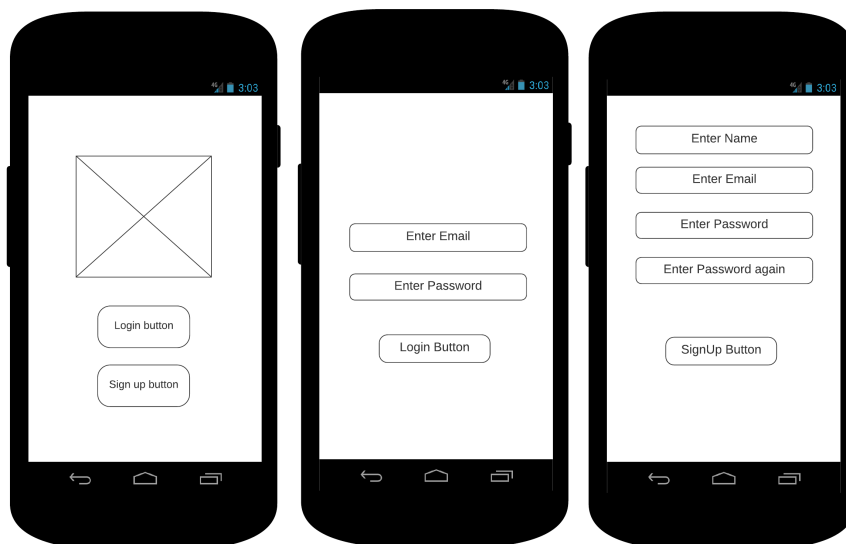


Figure 5.1: Login and Sign-up wireframes

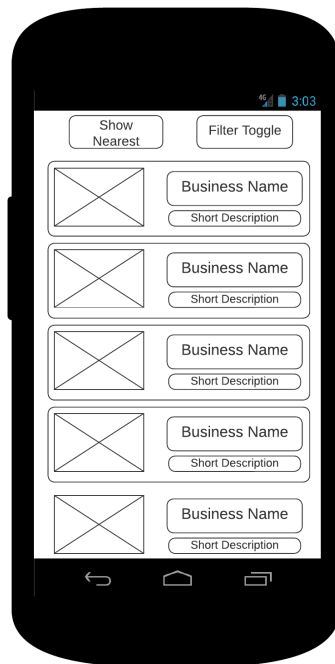


Figure 5.2: Business list wireframe

Figure 5.2 demonstrates the screen which allow users to see all businesses in a list view with brief details. The screen is also capable of listing businesses based on their distance to user's location. As is demonstrated by the top button which sorts the list based on their distance from the user. This Covers UC3 and UC8.

Figure 5.3 shows the business detail screen, which allows the user to see the services offered and the reviews at a business. Hence, it covers UC4 and UC5.

Figure 5.4 shows the create booking screen, which allows the user to create a new booking and also lists all the past and upcoming bookings of a user. This finally covers the use cases UC6 and UC7

5.2 Client Server Architecture

The client-server model is a distributed application architecture that divides tasks or workloads between providers of a resource or service, known as servers, and service requesters, known as clients.

In the proposed system, a pragmatic approach has been employed to adhere to the client-server model, separating responsibilities to ensure an efficient and modular system. The server-side is responsible for tasks such as storing

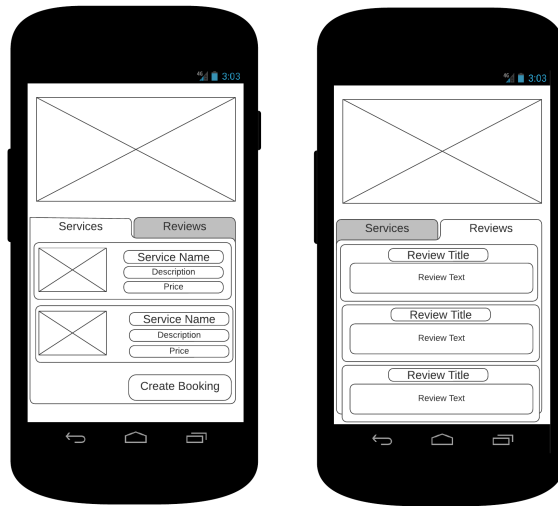


Figure 5.3: Business details view wireframe



Figure 5.4: Create bookings and list bookings wireframes

and retrieving data related to beauty services, businesses, and user bookings. It also manages user authentication, authorization, and overall system security. The server communicates with the client through well-defined APIs, facilitating a structured and consistent data exchange.

Conversely, the client-side is responsible for offering a user-facing interface, enabling users to interact with the system. Prohibiting them from performing illegal operations The client retrieves data from the server, processes it, and presents it to users in an organized and comprehensible manner, ensuring a seamless user experience.

5.3 API design

Prior to commencing the implementation of the server and client components, it is imperative to establish a well-defined API specification that will govern the communication between these two elements. The design of the API endpoints adheres to the best practices of Representational State Transfer (REST) API design, as delineated in the relevant literature. Drawing upon the database model and established best practices, three primary resources have been identified, and corresponding endpoints have been devised to facilitate seamless interaction between the client and server components. By creating a robust API specification, the foundation is set for the successful development and integration of the system's various components, ultimately ensuring an efficient and reliable communication channel.

The design of the endpoints themselves follows the best practices of Representational state transfer (REST) API design from [8]. Based on the Database model and practices mentioned, I have identified 3 resources and created endpoints based on them:

Businesses: Those endpoints start with “/businesses” and cover operations that are related to business and resources associated with a business like reviews and services.

Bookings: They start with “/bookings” and allows the client to get and manage bookings.

User: Endpoints from this group start with “/users” and expose the functionality of logging in and registering a new user.

5.4 System Architecture

5.4.1 User Authentication

Ktor's documentation provides a valuable resource on the available options for authentication and authorization, Subsequently, the below paragraph draws inspiration from the source[9].

Requirement F2 states a crucial need to equip users with the capability to authenticate themselves, a common requirement in contemporary information systems, including the one under development in this thesis. I spent time trying to analyze all the options supported out of the box by Ktor. The below list will briefly discuss the available options.

HTTP Basic A widely used, protocol-based authentication method where a client sends a user and password to the server with each request. It is a simple and stateless scheme.

Bearer Token (JWT) Utilizing JSON Web Tokens (JWT), this method provides a stateless and secure way of authenticating users. The server issues a token to the client, which the client then includes in subsequent requests.

LDAP Lightweight Directory Access Protocol is a complex and robust method primarily used in corporate environments. It authenticates users against an existing directory server, often containing user credentials and other organizational data.

OAuth A comprehensive protocol that allows third-party applications to grant limited access to an HTTP service. It can be used for various tasks, such as letting users authenticate with external services like Google or Facebook.

Session They provide a way to preserve certain data across subsequent accesses. They allow maintaining data across requests, offering a more sophisticated way of tracking and authenticating users over multiple requests.

Custom Ktor also provides an API for creating custom plugins, which can be used to implement your own plugin for handling authentication and authorization.

Upon evaluating the available options, A custom solution was dismissed because of the potential for severe security vulnerabilities. Among the remaining options, JSON Web Tokens were selected as the most suitable for this application, largely due to my familiarity with the technology and several inherent benefits discussed below.

JWTs are stateless[10], meaning they don't require the server to store session information about the user. This attribute greatly enhances load balancing efficiency, which is crucial for potential distributed deployment scenarios in the future. Moreover, JWTs are versatile, supporting various authentication schemes and securely containing user information, thereby enhancing data security and integrity.

Another significant advantage of JWTs is their capacity to offer a smooth authentication experience across a range of services or microservices. This quality will be particularly beneficial if the application architecture expands or diversifies in the future. The choice of JWTs aligns with the objective of a secure, efficient, and adaptable application design.

Here is a simplified overview of user authentication flow using JWT:

- 1. User Authentication** When a user logs into the application using his username and password, the server verifies the user's credentials.
- 2. Token Generation** If the credentials are valid, the server generates a JWT. This token includes a payload containing information about the user. Ktor's JWT feature can be used to generate this token.
- 3. Token Issuance** The server then sends this JWT back to the client application.
- 4. Authenticated Requests** The client app securely stores this JWT, and for each subsequent requests that require authentication, the client app adds the JWT to the request's Authorization header.
- 5. Token Validation** The server, upon receiving these requests, validates the JWT. If the token is valid, the server processes the request. If the token is not valid (or is missing), the server responds with an error.
- 6. Token Refresh** The token will have a predefined expiration set and once the token expires the user will need to request a new token.

5.5 Final Architecture

Having established the tech stack, crafted the wireframes, defined the REST endpoints, and selected JWT for user authentication, it's now time to delineate the complete system architecture. The system is designed to meet all stipulated requirements, encompassing a mobile client and a server. User authorization is facilitated through JWT tokens, and all communication is conducted via HTTP. The database, also hosted in the cloud, communicates using Java Database Connectivity. This comprehensive layout illustrates the cohesive, interconnected structure of the system, ensuring clarity and efficiency in its operation.

5.6 Android Design

This section describes the important decisions in the Client Application.

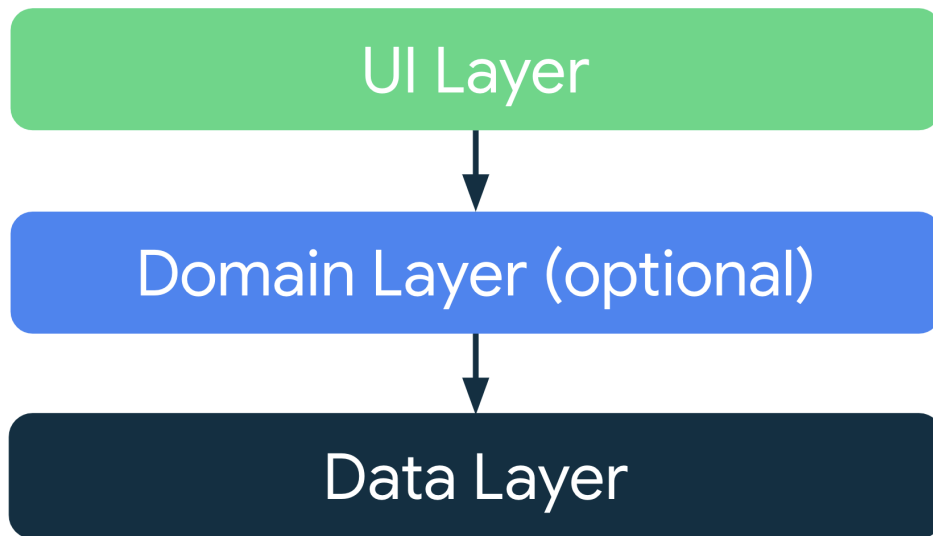


Figure 5.5: Recommended architecture by Google

5.6.1 Application architecture

The selection of an appropriate architectural framework is an essential decision that necessitates careful deliberation before the commencement of the development process. In this context, the architecture proposed by Google [11], as depicted in Figure 5.5, was the chosen blueprint for our project. The principal merit of embracing this recommended architectural design lies in its inherent ability to facilitate the segregation of concerns. This is effectuated by the systematic partitioning of different The inclusion of an optional domain layer is contingent upon its necessity. This design philosophy ensures a cleaner, more manageable codebase, reinforcing the decision to adopt this architecture.

5.6.2 UI layer

This paragraph is based on article on article from Android documentation. [11] To Display the UI Layer, I have decided to use MVVM architectural pattern in my application, for its effectiveness in facilitating a structured and maintainable codebase in Android development. This pattern encourages a distinct separation of concerns, with each component assuming responsibility for a specific aspect of the application. The Model represents data and business logic, the View is in charge of user interface and presentation, and the ViewModel orchestrates communication between the Model and View. By implementing the MVVM pattern, developers can establish a modular structure, which in turn simplifies debugging, testing, and code reuse. Moreover, MVVM pro-

notes scalability and eases the process of accommodating new requirements, solidifying its status as a highly recommended best practice in Android development.

5.7 Server design

The architecture of server is a three-layered application consisting of (persistence layer, business layer, and presentation layer. As was taught in the course BIE-TJV (Java Technologies) [12]

Presentation layer This layer is responsible for creating routes that expose the server’s functionality via a REST API, using the HTTP protocol. It essentially provides an interface for users and other systems to interact with the application.

Business layer Situated between the presentation and data layers, the business layer encapsulates the core application logic of the server. This includes processing incoming data, applying business rules, and preparing data for storage or dissemination.

Data layer The underlying layer is the data layer, whose primary function is data persistence. In this architecture, it interfaces with an SQL Server database, ensuring that all data manipulated by the business layer is accurately stored and readily available for retrieval.

Maintaining a clean and efficient architecture is paramount. In this design, the Presentation Layer communicates exclusively with the Domain Layer, which in turn interfaces with the Data Layer. Importantly, there are no dependencies in the opposite direction, maintaining a unidirectional flow of communication and dependencies. This structure promotes loose coupling between the layers, enhancing the system’s adaptability. As future modifications become necessary, this architecture ensures that changes can be implemented with minimal impact on the overall system. This setup not only simplifies maintenance but also facilitates potential future expansions or adaptations, thus underscoring the architecture’s robustness.

Implementation

6.1 Android Implementation

In section [4.1.3](#), I already reason my choice of Kotlin for the development of the android mobile client.

6.1.1 Libraries

In this section I would like to go over the external libraries I have used in the development of this thesis.

6.1.1.1 Jetpack Compose

This passage is based on [\[13\]](#)

Jetpack Compose, a modern toolkit for building native Android UI, has been employed in the development of my application. It utilizes a declarative approach to UI design, significantly simplifying the process of creating complex interfaces. This reduces boilerplate code, allowing developers to focus on the core functionality of their app. Jetpack Compose also provides state management tools, that help to maintain a consistent UI state across the application. Furthermore, it integrates seamlessly with other Jetpack libraries, ensuring compatibility with existing Android infrastructure. Its adoption in my application has led to a streamlined development process, with a clear, maintainable codebase and an efficient, reactive UI system.

6.1.1.2 Hilt

Utilizing dependency injection (DI) in modern application development is critical, as it promotes modularity, maintainability, and test-ability. Hilt, a DI library recommended by Google for Android development, is employed in this application to achieve these benefits. Built on top of the Dagger library, Hilt aims to simplify the infrastructure, enhance readability, and provide superior

integration with the Android framework. Incorporating Hilt into the application is essential for several reasons, such as facilitating a modular codebase through its module system, enabling cleaner and more maintainable code by managing dependency instantiation, simplifying the process of testing by allowing developers to effortlessly replace dependencies with test doubles, and providing seamless integration with Android components for a cohesive development experience. In summary, the use of Hilt in this application is crucial for promoting a modular, maintainable, and testable codebase while ensuring the application's architecture adheres to best practices in modern application development.

6.1.1.3 Compose Destinations

This passage is based on [\[14\]](#) Compose-Destinations is a library extending Jetpack Compose Navigation, designed to improve navigation within Android applications. Jetpack Compose Navigation is a declarative framework facilitating seamless transitions between composables, while Compose-Destinations augments this functionality by providing a structured approach to defining and managing navigation routes. The key advantages of Compose-Destinations over Jetpack Compose Navigation include type-safe Kotlin DSL, which allows developers to declare navigation destinations using a Kotlin DSL, eliminating string-based route definitions and reducing runtime errors. This approach enhances readability and maintainability, promoting a more organized and maintainable codebase. Furthermore, Compose-Destinations serves as a comprehensive navigation solution by supporting deep-linking, nested navigation, and backstack management, offering a complete solution for various navigation scenarios. In conclusion, Compose-Destinations is a valuable addition to Jetpack Compose Android applications, providing an enhanced navigation experience and promoting a more organized and maintainable codebase.

6.1.1.4 Retrofit

In contemporary application development, efficient and streamlined network communication is essential. Retrofit, a type-safe HTTP client for Android and Java, is a widely adopted library that simplifies this task. Developed by Square, Retrofit is preferred by many developers due to its various advantages, including type safety, which ensures that the types of data being sent and received are consistent, thus reducing potential errors. The library allows developers to define network API calls using simple, declarative interfaces, resulting in clean and maintainable code. Retrofit seamlessly integrates with JSON parsing libraries, such as Gson and Moshi, simplifying the process of parsing API responses into domain objects. By incorporating Retrofit into the application, the developer can effectively manage network communication,

minimize errors, and streamline the development process, while adhering to best practices.

6.2 Server Implementation

In section [4.2](#) we have already reasoned are reasoning for selecting Ktor framework for our server. In section [5.7](#) I discussed the organisation of code in our server.

6.2.1 Libraries

In this section I will talk about the details of external libraries used in the development of the server for its implementation.

6.2.1.1 Ktor

Ktor was used for server development in this project. Being first released in 2018, I was afraid that it would be difficult to find help online, If I got stuck or found hidden bugs. But Ktor has very detailed and in depth documentation [\[15\]](#) that helped me whenever I got stuck. It is modular by design, enabling developers to select features they require without unnecessary bloat. Some noteworthy modules include Locations [\[16\]](#), which provide typed routing and are used to define REST endpoints. Authentication [\[9\]](#) is another essential module that manages the authentication of users out of the box. Ktor also provides Database and Exposed modules for connecting and interacting with databases.

When I got started with Ktor, I started with their Getting started with HTTP API tutorial [\[17\]](#) which taught me the basics of the framework, after that I immediately started applying my knowledge gathered from BIE-TJV [\[12\]](#) course to develop the server.

6.2.1.2 Exposed

Exposed [\[18\]](#) is a lightweight SQL library used within the Ktor framework for this project. It provides an idiomatic Kotlin API to connect and interact with relational databases. The library stands out due to its two distinct layers of database access: a typesafe SQL wrapping DSL and a light object-relational mapping (ORM) layer. The DSL layer allows developers to write SQL queries in a typesafe manner, while the ORM layer provides a more abstract way to work with databases, using Kotlin classes and objects. Additionally, the Exposed library handles connection pooling, a feature that significantly enhances performance when working with databases, particularly under high load. Its compatibility with various SQL dialects like PostgreSQL, MySQL, Oracle, SQLite, SQL Server, and H2 offers flexibility and ease of use, making it a valuable asset in the project.

6.2.2 Image Storage

Storing images for businesses and the services they offer raised a unique challenge in this project - the need for substantial storage space. High-resolution images that businesses would ideally want to display can be sizeable and storing such large images directly in the database would rapidly inflate the database costs. Thus, it was essential to explore alternative storage strategies.

Upon researching, I discovered that numerous cloud storage providers offer services tailored for storing data. The top three providers, namely Amazon AWS, Google GCP, and Microsoft Azure, all offer such services. Given my prior experience with AWS S3 Bucket services, I decided to opt for this platform.

With a swift configuration process, the S3 Bucket was ready to accept image data and return corresponding URLs. These URLs could then be used to retrieve images on the mobile client. As the business images are intended for public view, there are no inherent security risks, allowing me to configure the S3 Bucket to return publicly accessible URLs. This decision facilitated an efficient, secure, and cost-effective solution for storing and retrieving large image files.

6.2.3 Database

Azure[\[19\]](#) SQL Database, a cloud-based relational database service from Microsoft, was utilized in this project. It offers scalable and highly available SQL database functionality with minimal maintenance, making it an excellent choice for my Ktor server. Leveraging Azure SQL Database eliminated the need for manual setup and maintenance of the database server, allowing for more focus on application development. I quickly created tables and its attributes for my database on the server side code and the Exposed library generated the SQL code which will generate the actual tables on connected Azure Database instance Model is represented by figure [6.1](#).

6.3 Deployment

6.3.1 Cloud infrastructure

To make the server accessible from any location, I had to deploying it to a cloud environment. In exploring options for this requirement, I referred back to Ktor's deployment documentation [\[20\]](#). This source offers guides for deploying the server to three platforms, namely, AWS Elastic Beanstalk, Google GCP Engine, and Heroku.

After studying all three guides, I found Heroku's deployment process to be the most intuitive. A particularly appealing feature was the ability to directly connect my GitHub repository to Heroku, enabling automatic deployments to

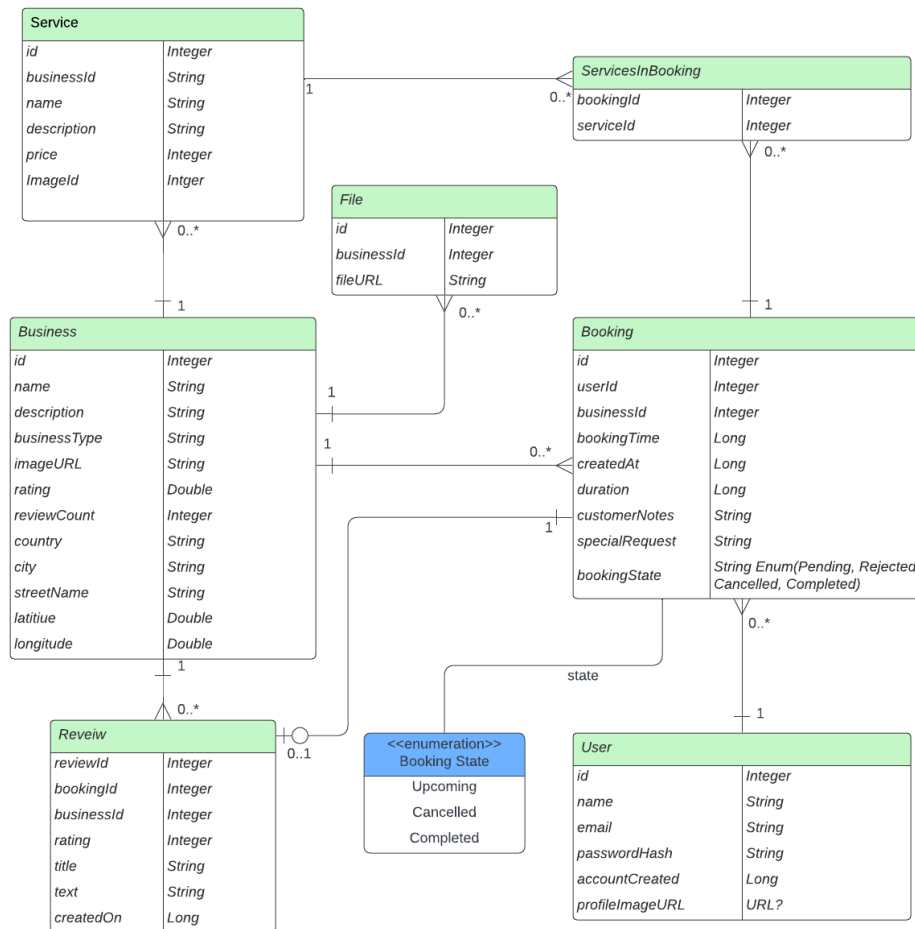


Figure 6.1: Database model

production with every push or merge to the main branch. Therefore, I decided to use Heroku for the deployment of the server.

The final architecture of the application, including this cloud deployment, is illustrated in Diagram [6.2](#). This setup ensures that the mobile client can access the server from anywhere, an essential feature for the application's usability.

6.3.2 Android deployment

At this stage, my prototype application is not prepared for public distribution, so I have not taken steps to publish it on the Google Play Store. However, the application's build system, Gradle, enables swift creation of an Android Package (APK). This APK is essentially an installation package, allowing

6. IMPLEMENTATION

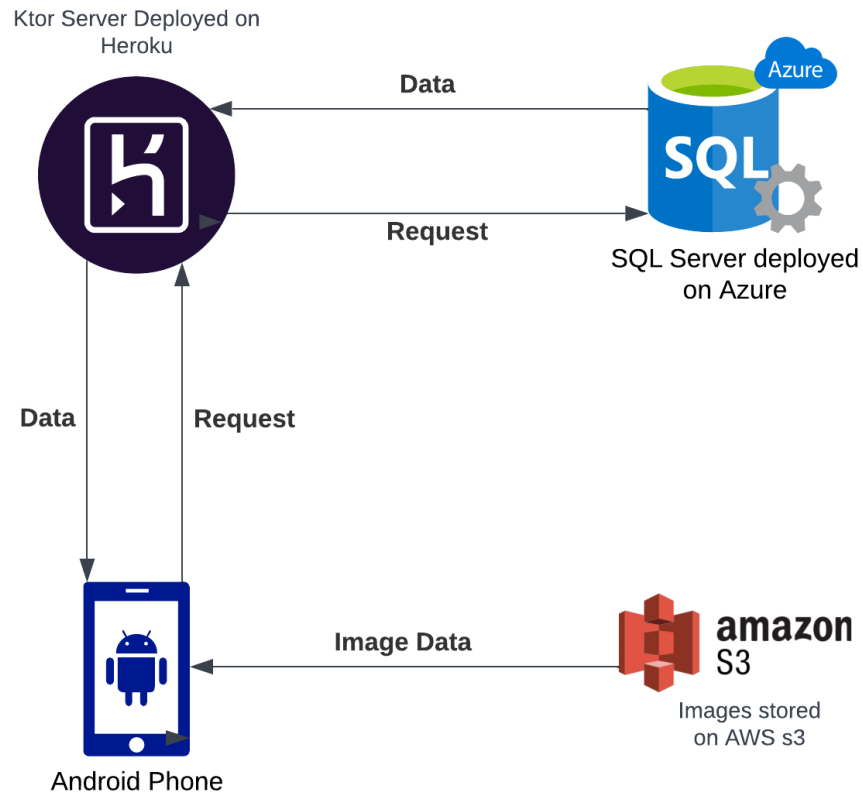


Figure 6.2: Cloud infrastructure

users to install the application on their devices. For now, I plan to distribute the APK manually to interested testers.

User Testing, Feedback and Future Improvements

7.1 User testing

In the process of developing the application prototype, I sought to gather diverse feedback to enhance the app's functionality and user experience. Conversations with friends and colleagues revealed a keen interest in the application, leading me to recruit them for user testing. To broaden the feedback spectrum, I also involved older relatives, ensuring insights from a wide age range and varied backgrounds.

Since the prototype wasn't ready for deployment on Google Play Store, I supplied the testers with the Android Application Package (APK) and an installation manual. Test credentials were provided, enabling them to log into the application and explore its features. To make the testing process more engaging, the database was pre-populated with dummy data, such as sample businesses, their services, and customer reviews. This allowed testers to experience the app more realistically and provide more relevant feedback. The tasks given to each tester were as follows:

1. Open the application and log in using the provided credentials.
2. Try to list all the businesses that are within 5 kilometers of your location.
3. Try to open a business's page and list all the services provided by them.
4. Try to open a business's page and list all the reviews left by other users.
5. Try to create a booking at a business of your choice.
6. Finally try to find the booking you just created in "My Bookings" Tab.

After the testers completed these tasks, I gave them a set of questions and asked for their honest feedback about the application.

7. USER TESTING, FEEDBACK AND FUTURE IMPROVEMENTS

1. Were you easily able to perform all of the requested tasks ?
2. Did you encounter any bugs or unexpected behaviour ?
3. What do you think about the concept of the application ? How can this be improved ?
4. What is the thing about the application you did not like ?

The majority of user testing participants successfully executed the tasks and conveyed positive feedback about the application prototype. However, it's possible that their relationship with me may have influenced their responses to be overly positive. Therefore, I will concentrate on presenting the more critical feedback, as it offers valuable insights for improving the application.

7.2 Feedback

User 1

1. Yes, But once I sorted the businesses by rating, I was unable to sort them back, the application provided no such functionality.
2. Yes, I found the behaviour of manually entering the distance to filter the businesses quite weird.
3. Possible, further polishing the UI, by maintaining more consistent design choices.
4. Although this is a prototype, I would like to see more features in the future version of the app.

User 2

1. Yes, the overall operation was smooth. However, I experienced some difficulty understanding the booking process as it lacked clear instructions.
2. The user interface seemed a bit cluttered. Simplifying and streamlining the display could improve the user experience.
3. I believe the concept is quite useful. One improvement could be the incorporation of a more personalized recommendation system based on user preferences.
4. More Features, Ability to select services that I want to receive to be listed in my Bookings.

User 3

1. Yes, but it took a while to understand how to filter businesses by distance, the design for that is not intuitive.
2. Yes, It did.
3. The concept of the application is good, but I rarely visit a new saloon, I always get my haircut from the same barber, thus eliminating the need for such an application.
4. The user interface is a bit cluttered, making it hard to focus on the necessary information.

User 4

1. Mostly.
2. Yes.
3. The concept is interesting. An improvement could be to add a feature that allows users to see available time slots for each business.
4. The colors and design could be improved to be more pleasant.

User 5

1. Yes, but I struggled to find the log-out option. It was not intuitively placed.
2. Yes, I couldn't filter businesses based on the services they offer.
3. The concept is promising, but it could be improved by adding a feature that allows users to mark businesses as favorites.
4. I did not like the lack of a search function. It was cumbersome to scroll through all the businesses.

User 6

1. I was not able to find the log-out option.
2. Yes, I did not like the behaviour of searching by distance, which requires the number of kilometers typed out.
3. I loved the concept and would love to see it released with full feature set.
4. Despite the requirement to log-in to be able to browse the businesses, a user should be able to browse without being logged in.

7.2.1 Summary

The feedback from the prototype testing is encouraging, with most testers able to accomplish the assigned tasks, indicating a robust foundation for future application development. Importantly, the concept was widely appreciated, with a majority of testers expressing a desire for additional features. However, the feedback also highlighted areas for improvement, especially in terms of user interface design. A notable area of concern was the UI element that required users to input the search radius in kilometers, which was seen as not intuitive. The feedback suggests that addressing this issue should be a priority in the next iteration of the application. Several testers also raised issues regarding the application's log-out functionality and the inconsistency in the color scheme. Given this feedback, future development plans should include consulting with a designer to ensure a cohesive and user-friendly design aesthetic for the application. The aim would be to address these concerns while maintaining the application's functionality and improving the overall user experience.

7.3 Future improvements

Since the focus of the thesis was to develop a MVP prototype, It only implemented the most sought after feature, but the user feedback says that there is also room for improvement in the current feature set.

Browsing for Guest Users The testing process revealed that the obligation to log in to browse available businesses and services could potentially deter users. Future iterations should consider allowing users to explore offerings without needing to log in, thus facilitating a more user-friendly experience.

Caching At present, the application operates exclusively in online mode due to the absence of local database functionality. While this is acceptable for a prototype, it is recommended that future versions include data caching and paging implementation to optimize performance and minimize unnecessary network resource consumption.

Unit Testing and Integration Tests The rapid pace of prototype development frequently necessitated swift modifications, which made unit testing somewhat burdensome due to the constant need for updates. Nonetheless, future iterations should strive for industry-standard code coverage, emphasizing the importance of robust unit testing and integration tests.

Czech Language support Since the goal of our application is to capture the Czech market, It is therefore important to add support for Czech language in our application.

Business Application As noted during the analysis phase, the distinct needs and features related to businesses necessitate a separate platform for managing bookings and promoting their establishments. Therefore, a logical progression for further development would be the implementation of a dedicated business application. This addition would serve to complete the entire system, providing a comprehensive solution for all users.

Conclusion

The primary objective of this thesis was to design and develop a client-server application prototype that facilitates the process of searching for and booking various types of beauty services. The resulting prototype is proficient in listing services in proximity to user's location, filtering based on specified criteria, listing the services offered by the business, listing the reviews left by the user for the application and creating bookings. At the start, an investigation of existing applications for searching and booking beauty services was conducted. The current market landscape, including prominent platforms, were analyzed, and their strengths and weaknesses were identified. This research enabled the discovery of potential opportunities for a location-based reservation system that addresses the limitations of existing solutions. A client-server solution was designed and implemented, empowering users to find and book beauty services near their current location. The application effectively lists services close to a given location, filters results based on various criteria, and enables users to create bookings with ease. By leveraging modern technologies Ktor for the backend and an Android app for the client-side, a prototype system was created. The server part of the solution was deployed using an online cloud provider, ensuring that the system is easily accessible, scale-able, and maintainable. This deployment strategy simplifies future updates and enhancements to the system. In the end, possible areas for improvement were identified, which could be implemented to make this application market-ready and release it to the general public. In conclusion, this thesis successfully achieved its goal of designing and implementing a prototype client-server solution for beauty services and deploying the server part using an online cloud provider. The insights gained and the solution developed throughout this thesis contribute to a better understanding of client-server application architecture, Android applications, and principles of software engineering, and demonstrate the potential of location-based application in the beauty services market.

Bibliography

- [1] Booksy. Booksy the appointment booking app. [cit. 2023-05-14]. Available from: <https://booksy.com/en-us/p/about>
- [2] Reporter, S. Booksy, the uber of beauty and wellnes. [cit. 2023-05-14]. Available from: <https://www.uktech.news/news/booksy-the-polish-uber-of-beauty-and-wellness-scoops-51m-plans-uk-expansion-20210127>
- [3] Fresha. Fresha. [cit. 2023-05-14]. Available from: <https://www.fresha.com>
- [4] Lunden, I. Fresha Raises 100M. [cit. 2023-05-14]. Available from: <https://techcrunch.com/2021/06/11/fresha-raises-100m-for-its-beauty-and-wellness-booking-platform-and-marketplace>
- [5] Schmitt, J. Native vs CrossPlatform mobile dev. [cit. 2023-05-14]. Available from: <https://circleci.com/blog/native-vs-cross-platform-mobile-dev/>
- [6] Globalstats, S. Marketshare android vs ios. [cit. 2023-05-14]. Available from: <https://gs.statcounter.com/os-market-share/mobile/worldwide>
- [7] Foundation, K. Calling Java from Kotlin. [cit. 2023-05-14]. Available from: <https://kotlinlang.org/docs/java-interop.html>
- [8] Postman. What is API Design, Principals and best practices. [cit. 2023-05-14]. Available from: <https://www.postman.com/api-platform/api-design/>
- [9] JetBrains. Authentication and authorization. [cit. 2023-05-14]. Available from: <https://ktor.io/docs/authentication.html>

- [10] JetBrains. JSON Web Tokens in Ktor. [cit. 2023-05-14]. Available from: <https://ktor.io/docs/jwt.html>
- [11] Google. Android Developers. [cit. 2023-05-14]. Available from: <https://developer.android.com/>
- [12] Guth, O. Java Technology. [cit. 2023-05-14]. Available from: <https://courses.fit.cvut.cz/BIE-TJV/index.html>
- [13] Google. Jetpack Compose UI App Development Toolkit. [cit. 2023-05-14]. Available from: <https://developer.android.com/jetpack/compose>
- [14] Costa, R. Compose Destinations simple and safe navigation in compose. [cit. 2023-05-14]. Available from: <https://proandroiddev.com/compose-destinations-simpler-and-safer-navigation-in-compose-with-no-compromises-74a59c6b727d>
- [15] JetBrains. Create asynchronous client and server applications. Anything from microservices to multiplatform HTTP client apps in a simple way. Open Source, free, and fun. [cit. 2023-05-14]. Available from: <https://ktor.io>
- [16] JetBrains. Ktor Locations for creating routes. [cit. 2023-05-14]. Available from: <https://ktor.io/docs/locations.html>
- [17] JetBrains. Creating Http APIs. [cit. 2023-05-14]. Available from: <https://ktor.io/docs/creating-http-apis.html>
- [18] JetBrains. Exposed. [cit. 2023-05-14]. Available from: <https://github.com/JetBrains/Exposed/wiki>
- [19] Microsoft. Azure SQL Database. [cit. 2023-05-14]. Available from: <https://azure.microsoft.com/en-us/products/azure-sql/database>
- [20] JetBrains. Deployment — Ktor. [cit. 2023-05-14]. Available from: <https://ktor.io/docs/heroku.html>

Acronyms

API	Application Programming Interface
APK	Android Package
AWS	Amazon Web Services
DI	Dependency Injection
DSL	Domain-specific Language
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JPA	Jakarta Persistence
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVVM	Model–view–viewmodel
OS	Operating System
RDS	Relational Database Service
REST	Representational State Transfer
SDK	Software Development Kit
SQL	Structured Query Language

A. ACRONYMS

UI User Interface

UX User Experience

Contents of enclosed CD

exe.....	the directory with executables
├─ FreshCut-Server-v1.0.jar.....	executable of the server
├─ FreshCut-Client-v1.0.apk...	installation package of the application
implementation.....	source codes of implementation
├─ freshcut-client.....	source code of Android application
├─ freshcut-server.....	source code of the server
thesis.....	the thesis text directory
├─ thesis-sources.....	source code of the thesis
├─ thesis.pdf.....	the thesis text in PDF format