



FEE

**Faculty of Electrical Engineering
Department of Computer Science**

Bachelor's Thesis

System for blogging and planning one-day tourist trips including points of interest (POI)

Anhelina Rudzenka

May 2024

Supervisor: RNDr. Ladislav Serédi

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Rudzenka** Jméno: **Anhelina** Osobní číslo: **510637**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Softwarové inženýrství a technologie**
Specializace: **Enterprise systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Blogovací systém pro záznam a plánování tras jednodenních výletů včetně bodů zájmu (POI)

Název bakalářské práce anglicky:

System for blogging and planning one-day tourist trips including points of interest (POI)

Pokyny pro vypracování:

Po prozkoumání stávajících řešení, navrhnete architekturu webové aplikace, která bude nabízet uživatelům možnost plánovat jednodenní výlety.

Prostudujte existující veřejná API mapových aplikací a jejich potenciál pro využití ve Vaší aplikaci..

Aplikace bude disponovat funkcionalitou blogu, umožňující cestovatelům detailně popisovat a publikovat své zážitky z výletů. Zveřejněné příspěvky budou strukturované a skládat se z informačních bloků, které charakterizují města, památky, muzea, restaurace a další zajímavé body (POI – points of interest). Příspěvky budou moci obsahovat obrázky a geografické polohy, propojené s online mapovými službami. V závislosti na typu POI bude každý blok obsahovat povinná pole k vyplnění.

Systém bude schopen pomoci uživateli při plánování výletů na základě již vložených příspěvků. K tomuto účelu bude možné uložené příspěvky cestovatelů prohledávat a filtrovat. Když tímto způsobem uživatel najde pro sebe vhodný výlet, do svých záložek uloží relevantní příspěvky nebo pouze vybrané POI.

Aplikaci bude tvořit serverová a klientská část. Zprávu uživatelů včetně autentifikace zajišťuje server. Klientská část bude navržena ve formě single page aplikace ve vhodně zvoleném frameworku (např. Angular). Data se ukládají do databáze, přístupné z back-end, kde bude rovněž implementována obchodní logika (business logic).

Na základě vašeho návrhu implementujte klíčové části systému takovým způsobem, aby bylo možné otestovat klíčové uživatelské scénáře. Výsledky testů vyhodnoťte, diskutujte dosažené výsledky, případně i zjištěné nedostatky a možný budoucí vývoj aplikace.

Seznam doporučené literatury:

CHAUDHARI, Kinjal; THAKKAR, Ankit. A comprehensive survey on travel recommender systems. Archives of Computational Methods in Engineering, 2020, 27: 1545-1571.

KYSELA, Jiří. Analysis of usability of various geosocial network POI in tourism. In: Applied Informatics: Second International Conference, ICAI 2019, Madrid, Spain, November 7–9, 2019, Proceedings 2. Springer International Publishing, 2019. p. 32-42.

ZHAO, Pengpeng, et al. Where to go next: A spatio-temporal gated network for next poi recommendation. IEEE Transactions on Knowledge and Data Engineering, 2020, 34.5: 2512-2524.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ladislav Serédi kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **01.02.2024**

Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

RNDr. Ladislav Serédi
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studentky

Acknowledgement / Declaration

I would like to express my gratitude to RNDr. Ladislav Serédi for agreeing to serve as thesis advisor, for his help and advice.

I thank my parents for their support and unconditional belief in me.

I thank my friends Mikita Citarovič and Veronika Ovsyannikova, who were always there for me and greatly helped me in my studies.

I thank my friend Elizabeth Ryzhevich for her support and understanding.

I thank my friend Juri Golomako for technical advice and experience sharing.

I thank all the teachers, classmates and colleagues who have helped me during my studies.

I hereby declare I have written this thesis work independently and quoted all the sources of information used following methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, 24. May 2024

.....

Abstrakt / Abstract

Bakalářská práce je věnována vytvoření webové aplikace, která disponuje funkcionalitou blogu a umožňuje cestovatelům vytvářet, sdílet a číst příspěvky (plány cest) popisující krátkodobé výlety. Tyto příspěvky jsou sestaveny z jednotlivých bloků míst, z nichž každý představuje konkrétní místo zájmu (POI – point of interest), jako jsou muzea, restaurace, památky a další. Uživatelé mají možnost vyhledávat plány cest v jejich blízkém okolí zadáním místa odjezdu a/nebo cílového místa. Projekt využívá platformu Google Maps jako zdroj geografických dat a pro vykreslení online mapy.

Projekt se zaměřuje na krátkodobé (jednodenní) plány cest skládající se z míst v blízkosti uživatelského místa odjezdu a/nebo cílového místa, což jej odlišuje od existujících řešení.

Architektura aplikace byla navržena jako monolitický klient-server, přičemž byla použita třívrstvá architektura se samostatnou prezentační vrstvou, vrstvou logiky a vrstvou přístupu k datům. Klientský projekt byl vyvinut pomocí technologie Angular, zatímco serverový projekt byl vyvinut pomocí technologií .NET.

Aplikace prošla neformálním manuálním testováním a akceptačním testováním.

Klíčová slova: blogovací systém, plánování výletů, body zájmu, místa zájmu, cestování, klient-server, REST, API, .NET, C#, Angular, PostGIS, PostgreSQL, Google Maps Platform, backendová aplikace, frontendová aplikace.

Překlad titulu: Blogovací systém pro záznam a plánování tras jednodenních výletů včetně bodů zájmu (POI)

The bachelor thesis is dedicated to creating a web application built on blog functionality, allowing travelers to create, share and read posts (trip plans) describing short-term journeys. These posts are constructed using discrete places blocks, each representing a specific place of interest (POI), such as museums, restaurants, monuments, and more. Users have the capability to search for trip plans in their nearby area by providing the departure and/or destination locations. The project utilizes the Google Maps Platform as a source of geographical data and for rendering the online map.

The project focuses on short-term (one day) travel plans consisting of places near the user's departure and/or destination locations, which makes it different from existing solutions.

The application's architecture was designed as a monolithic client-server, adopting a Three-Tier architecture with separate Presentation Layer, Business Logic Layer, and Data Access Layer. The client project was developed using Angular, while the server project was developed using .NET technologies.

The application underwent informal manual and acceptance testing.

Keywords: POI, place of interest, travelling, trip planning, blog, client-server, REST, API, .NET, C#, Angular, PostGIS, PostgreSQL, Google Maps Platform, backend, frontend.

Contents /

1 Introduction	1
1.1 Motivation	1
1.2 Project description	1
1.3 Target audience	2
1.4 Research of existing solutions	2
1.4.1 TripAdvisor	2
1.4.2 Wanderlog	3
1.4.3 PlanYourTrip	3
1.4.4 RoutePerfect	4
1.4.5 Conclusion	4
1.5 Goals	4
2 Analysis and Architecture	5
2.1 Application requirements	5
2.1.1 Business requirements	5
2.1.2 Functional requirements	5
2.1.3 Non-functional re- quirements	7
2.2 Use Cases	7
2.3 Business domain model	9
2.4 Low fidelity prototype of GUI	9
2.5 Architecture	10
2.6 Backend technologies	10
2.7 Frontend technologies	11
2.8 Database	11
2.9 External APIs	13
2.9.1 Google Maps	13
2.9.2 Mapy.cz	13
2.9.3 API selection	13
3 Implementation	15
3.1 Component diagram	15
3.2 Sequence diagram	16
3.3 Frontend	17
3.3.1 Components library	19
3.3.2 Files upload	21
3.3.3 Google Maps Platform	21
3.3.4 GUI	23
3.4 Backend	23
3.4.1 Data Access Layer	23
3.4.2 Business Logic Layer	25
3.4.3 Presentation Layer	26
3.4.4 Dependency injection	27
3.4.5 PostGIS	29
3.4.6 Storing uploaded files	30
3.4.7 Data mapping	31
3.4.8 Generating TypeScript contracts from C# DTOs	31
3.4.9 Unit of Work pattern	32
3.4.10 Security	33
3.5 Project setup and configu- ration	34
3.5.1 Running the frontend project	34
3.5.2 Running the backend project	34
4 Testing	35
4.1 Static testing	35
4.2 Manual testing	35
4.3 Unit testing	35
4.4 Usability testing and com- parison with TripAdvisor	36
4.5 Acceptance testing	37
5 Conclusion	38
5.1 Further prospects of the project	39
Bibliography	41
A Low fidelity prototype of GUI	45
B GUI	55
C Test cases	64
D Acronyms	73

Tables / Figures

2.1 Key differences between MySQL and PostgreSQL	12
4.1 Time spent to find trip plan on bachelor project application and the TripAdvisor application.....	36
C.2 Test case: Successful registration	64
C.5 Test case: Logout	64
C.1 Test case: Registration form validation	65
C.3 Test case: Registration failed because user with same email or username already exists.	66
C.4 Test case: Successful login.	66
C.6 Test case: Search for a place to visit.	67
C.7 Test case: View trip plan	67
C.8 Test case: Bookmarks management.....	68
C.9 Test case: Comment the trip plan	69
C.10 Test case: Trip plan and trip plan block rating	69
C.11 Test case: Create a trip plan ..	70
C.12 Test case: Change the trip plan publicity.....	70
C.13 Test case: Update the trip plan	71
C.14 Test case: Delete the trip plan .	72
1.1 Trip plan page with places blocks	2
2.1 Use Cases actors	7
2.2 Account management Use Cases.....	8
2.3 Trip planning Use Cases	8
2.4 Business domain model	9
2.5 DB-Engines Ranking of Relational database engines	11
2.6 Global popularity comparison between Google Maps and Mapy.cz	14
2.7 Comparison of Google Maps and Mapy.cz popularity in Czech Republic	14
3.1 Component diagram	15
3.2 Sequence diagram illustrating trip plan creation	16
3.3 Suggested places example.....	17
3.4 Authorization guard	17
3.5 Authentication interceptor.....	18
3.6 <i>SafeUrlPipe</i> and its usage.....	19
3.7 PrimeNG gallery view	20
3.8 PrimeNG file upload component customized with gallery view and actions	20
3.9 Methods in Angular Google Maps service.....	22
3.10 <i>OneDayTrip.DataAccess</i> project structure	23
3.11 POI entity configuration in DB context	24
3.12 Generated migration that adds <i>TripPlanBlockReactions</i> table	24
3.13 Token generation method in JWT service	25
3.14 Searching trip plan with parameters implementation	26
3.15 DTO validation using validation attributes.....	26
3.16 List of endpoints in Swagger ..	27
3.17 DAL dependencies registration	28
3.18 Correct services registration in DI container	28

3.19	Displayed saved geography data in pgAdmin 4 Geometry Viewer	29
3.20	Visualization of Cartesian and Spherical coordinate systems.....	29
3.21	Usage of PostGIS with the help of NetTopologySuite library and Entity Framework ..	30
3.22	<i>TripPlanReaction</i> mapping configuration between DAL entity and BLL model	31
3.23	Trip plan TypeScript contract generation configuration .	32
3.24	Generated trip plan contract ..	32
3.25	Unit of Work pattern visualization	33
3.26	ASP.NET Identity configuration	33
A.1	Desktop registration page prototype	45
A.2	Mobile registration page prototype.....	45
A.3	Desktop login page prototype .	46
A.4	Mobile login page prototype ...	46
A.5	Desktop home page prototype .	47
A.6	Mobile home page prototype ..	47
A.7	Desktop search results page prototype	48
A.8	Mobile search results page prototype	48
A.9	Desktop trip plan page prototype.....	49
A.10	Mobile trip plan page prototype	50
A.11	Desktop create new trip plan page prototype	50
A.12	Mobile create new trip plan page prototype	51
A.13	Desktop user's bookmarks page prototype	51
A.14	Mobile user's bookmarks page prototype	52
A.15	Desktop user's trip plans page prototype	52

A.16	Mobile user's trip plans page prototype	53
A.17	Mobile user's account page prototype	53
A.18	Desktop user's account page prototype	54
B.19	Desktop registration page.....	55
B.20	Mobile registration page	55
B.21	Desktop login page	56
B.22	Mobile login page	56
B.23	Desktop home page	57
B.24	Mobile home page.....	57
B.25	Desktop search results page ...	58
B.26	Mobile search results page	58
B.27	Desktop trip plan page	59
B.28	Mobile trip plan page	60
B.29	Mobile create new trip plan page.....	60
B.30	Desktop create new trip plan page.....	61
B.31	Desktop user's bookmarks page.....	62
B.32	Mobile user's bookmarks page .	62
B.33	Desktop user's trip plans page .	63
B.34	Mobile user's trip plans page ..	63

Chapter 1

Introduction

1.1 Motivation

The idea of this project was born during a trip to a city Mělník in Czech Republic. This trip has led me to the thought that many people who would like to travel and explore tourist attractions may face the same problem as me, which is a lack of time for trip planning and the trip itself.

I concluded that having a source of information about nearby tourist destinations, just a short distance from our homes, would be helpful, offering users trip guides suitable for a weekend or a day visit. The user could enter their departure location and the service could then provide suggested trip plans in the nearby area consisting of places of interest (POI) such as museums, castles, restaurants and more, along with detailed descriptions for each.

I also suppose that there are active tourists who would like to document their travel experiences and recommend lesser-known places to others. Such individuals might be interested in sharing their insights through such a service.

Based on my experience, I haven't come across any popular solutions that offer travel guides. While there are numerous services providing place ratings, suggestions, and paid guided tours, I have never come across solutions which publicly offer travel guides comprised of places and their detailed descriptions. Additionally, I have never encountered services specifically focused on short-term one-day or weekend trips. Such guides could help to explore unknown places nearby and make lives more diverse.

There are different ways to create such a service and one of them is a web application.

1.2 Project description

The web application is structured around blog functionality, enabling travelers to compose and share posts (trip plans) detailing their journeys. These posts are constructed using discrete places blocks, each representing a specific place of interest (POI), such as museums, restaurants, monuments, and more. Concrete places will be linked to external service, allowing users to access detailed information and utilize the online map by navigating to this external service.

Users planning their trips can conveniently search for trip plans by filtering them based on their proximity to either the starting point or the destination. Furthermore, users have the option to bookmark trip plans that satisfy their interest or save concrete places of interest to bookmarks for future reference. Users can revisit their bookmarks later and leverage them to create their own travel itineraries.

The app has a social aspect, enabling users to share their trip plans publicly, making them accessible to other users. Published travel plans will be available to the public for the benefit of other users. Additionally, users can rate trip plans and places, as well as leave comments on trip plans.

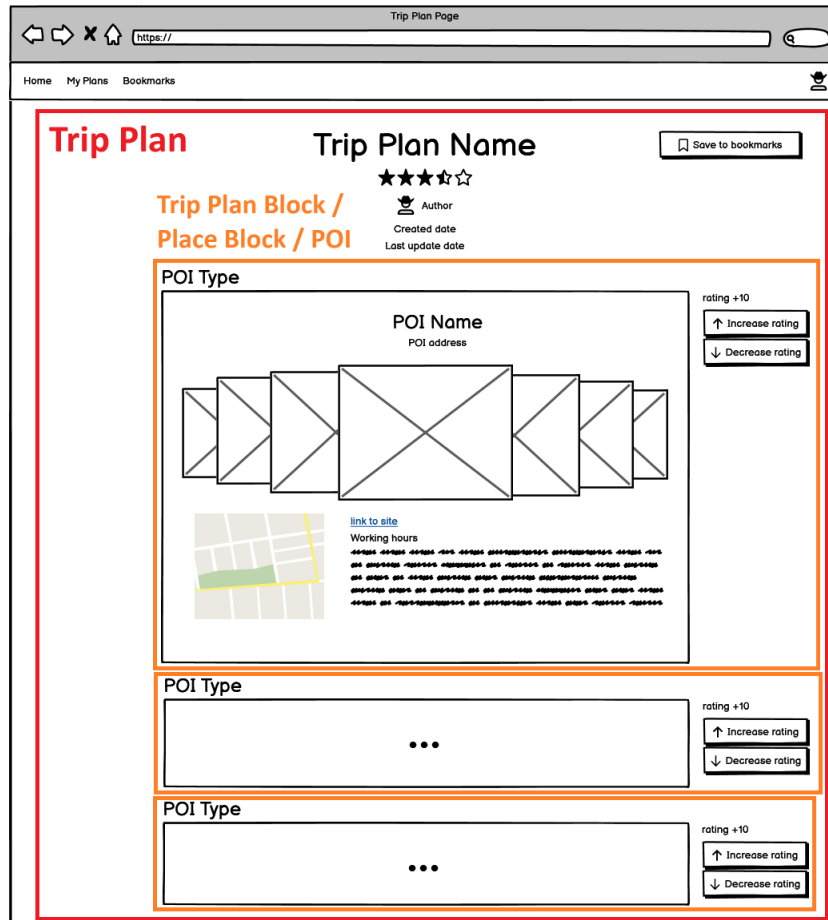


Figure 1.1. Trip plan page with places blocks.

1.3 Target audience

Target audience is travelers:

- Who want to document their journeys and share experience with the public.
- Who plan their journeys and lookup for a source of other travelers' knowledge.
- Who are searching for places to explore.
- Who have limited time for their trip.

In summary, the primary target audience for the application consists of people who want to travel or are already traveling.

1.4 Research of existing solutions

There are numerous travel-related services. The following sections will describe the advantages and disadvantages of the popular services that encompass functionality relevant to the project concept.

1.4.1 TripAdvisor

TripAdvisor is a travel-focused website with a vast database of POIs a places reviews [1]. The site also hosts a tours marketplace and a traditional forum where users engage in discussions about various travel-related topics. However, the website suffers from an overloaded interface.

Advantages:

- Vast POIs database.
- Big community.
- Guided tours marketplace.
- Suggest POIs nearby.
- Bookmarks functionality.
- Have travel guides written by users.
- During the development of this bachelor project, TripAdvisor introduced a beta version of a feature that assists in creating trip plans with the help of AI.

Disadvantages:

- The application is overloaded with functionality and navigation, making it challenging to interact with and find the necessary information. For instance, navigation bar differs between the main page, city-specific pages, and POIs pages.
- While the city pages feature paid guided tour recommendations available on a marketplace, there is a notable absence of suggestions for free trip plans. Users can only access these free travel guides written by users through a manual search, making them less discoverable.

■ 1.4.2 Wanderlog

Wanderlog is a travel planning and itinerary management platform that assists users in creating and organizing their trips, providing features for mapping out routes, discovering points of interest, and optimizing travel plans [2]. The platform also has features for reservations and expense tracking, complemented by travel guides and map views. However, there is a minor bug in the `like` function and a lack of bookmark functionality.

Advantages:

- Complex tool for planning. Users can add flights, hotel, car reservations, build itinerary, leave notes, and plan expenses.
- Travel guides.
- Great map view.
- Integration with trip advisor and google maps.

Disadvantages:

- Error in the `like` function. When you press the like button, the number of likes decreases. If there were no likes on the post, the number of likes becomes negative.
- No option to save guide to bookmarks.
- POI can't be saved to bookmarks standalone. It can only be added to the existing guide.

■ 1.4.3 PlanYourTrip

PlanYourTrip is a free itinerary planning tool offers users the ability to personalize their trips with advantages including estimated budget information and a map view, but it has a restricted list of destinations, relies on company-created guides without user-generated content, and has limited functionality [3].

Advantages:

- Estimated budget for proposed trip.
- Map view.

Disadvantages:

- Limited list of destinations.
- The company offers and publishes its own guides, lacking authentic user-generated content.
- Poor functionality.
- Recommendations for attractions but not restaurants options.

■ 1.4.4 RoutePerfect

RoutePerfect is a travel tool that helps users plan trips based on their preferences and budget, offering route advice, hotel suggestions, and booking integration [4]. The tool can interact with users through an AI-powered chat interface. However, it requires a minimum trip duration of four days and lacks user reviews.

Advantages:

- Advise road from starting point to destination with road full of places of interest.
- Hotel suggestions and integration with booking services.
- Unique nontrivial places suggestions.

Disadvantages:

- Can't set duration less than four days.
- Absence of content created by real users.

■ 1.4.5 Conclusion

None of the services mentioned above are solving exactly the same problem that I aim to tackle with the current project. The project must be focused on short-term travel plans consisting of places close to the user's departure location.

Considering all the features and characteristics of existing solutions, a decision has been made that the application will meet the following requirements:

- A simple and straightforward navigation, understandable even to inexperienced users
- The application will specialize in one day trips.
- A bookmarks functionality that provides the ability to save POIs or other users public guides.

The best aspects of existing solutions must be considered and known disadvantages must be avoided.

■ 1.5 Goals

The primary goal of the project is to create a web application built on blog functionality, allowing travelers to create, share and read posts (trip plans) describing short-term journeys. Additionally, users must have the capability to search for trip plans in their nearby area by providing the departure and/or destination locations. This involves proposing the application's architecture and researching publicly available travel data sources.

Key Objectives:

- BG-1 Research publicly available travel data sources.
- BG-2 Suggest web application architecture.
- BG-3 Create a web application.

Chapter 2

Analysis and Architecture

2.1 Application requirements

The purpose of this section is to define and describe the project requirements which are based on existing solutions analysis and target audience's preferences. The requirements are categorized into three groups [5]:

- Business requirements (BRQ) refer to tasks that are needed to fulfil to achieve a high-level objective. Explain what the result of a business goal should look like.
- Functional requirements (FRQ) define how a system needs to operate to achieve a business goal. Require an action to be taken by a person, system, or process.
- Non-functional requirements (NFR) define attributes or characteristics that the final solution needs to have.

Certain requirements will be designated as optional, as they are not essential to the core functionality and do not impede its operation. The primary emphasis is placed on the trip plans management functionality.

2.1.1 Business requirements

- BRQ-01 Account management: The user must have the capability to manage their accounts.
- BRQ-02 Trip planning functionality: The user must possess the ability to read published posts (trip plans) describing short-term journeys and manage their own trip plans. Users must have the capability to search for trip plans in their nearby area by providing the departure and/or destination locations. Additionally, users must be able to save trip plans and POI they liked to bookmarks and revisit their bookmarks later.

2.1.2 Functional requirements

FRQ-01 Account management:

- FRQ-011 Registration: The unregistered user should possess the capability to register.
- FRQ-012 Login: The registered user must be able to log in with credentials including email and password.
- FRQ-013 Logout: The authorized user must have the option to log out.
- FRQ-014 Change password (optional): The authorized user must be capable of changing their password.
- FRQ-015 Delete account (optional): The authorized user must have the ability to delete their account.
- FRQ-016 Change profile information (optional): The authorized user must possess the ability to change their profile information, including their username and email.

- FRQ-017 Review received ratings statistics (optional): The authorized user must be able to review ratings of their public trip plans.
- FRQ-018 Review given ratings (optional): The authorized user should possess the capability to review the ratings they gave to other users' trip plans.

FRQ-02 Trip planning functionality

- FRQ-0201 Search place to visit: The user (authorized and unauthorized) must be able to search for places nearby and trip plans created by other users. This must be accomplished by providing the departure and/or destination locations.
- FRQ-0202 View trip plan: The user (authorized and unauthorized) should possess the capability to view public trip plans from search result.
- FRQ-0203 Add the trip plan to bookmarks: The authorized user must have the ability to save the public trip plans they like to their bookmarks.
- FRQ-0204 Add the POI to bookmarks: The authorized user must have the capability to save POIs they like to their bookmarks.
- FRQ-0205 Delete the trip plan from bookmarks: The authorized user must have the ability to delete the trip plans from their bookmarks.
- FRQ-0206 Delete the POI from bookmarks: The authorized user must have the capability to delete POIs from their bookmarks.
- FRQ-0207 Make the trip plan public: The authorized user must be able to make their trip plan public or keep it private. Public trip plans will be available for search by other users, including those who are unauthorized.
- FRQ-0208 Delete the trip plan: The authorized user should be able to delete a trip plan they've created.
- FRQ-0209 Comment the trip plan: The authorized user must have the capability to comment on public trip plans.
- FRQ-0210 Rate a trip plan: The authorized user must have the ability to rate trip plans created by other users.
- FRQ-0211 Rate the place block: The authorized user must have the ability to rate place blocks created by other users.
- FRQ-0212 Create a trip plan: The authorized user must have the ability to create a trip plan.
- FRQ-0213 Update the trip plan: The authorized user must have the capability to update their previously created trip plan.
- FRQ-0214 Add the place block to the trip plan: The authorized user must possess the ability to add a block describing the place of interest to their trip plan.
- FRQ-0215 Update the place block in the trip plan: The authorized user should be able to update the place block, containing information about POI, within their trip plan.
- FRQ-0216 Delete the place block from the trip plan: The authorized user must have the ability to delete the place block from their trip plan.
- FRQ-0217 Select POI from external API to add it to place block: Authorized user must be able to find POI provided by external API and add it to the place block in their trip plan.
- FRQ-0218 Create POI manually to add it to place block: The authorized user must have the capability to manually create a POI if it is not found in external API data. They can then add it to the place block within their trip plan.
- FRQ-0219 Add POI to place block from bookmarks (optional): The authorized user must be able to select POI from their bookmarks and add it to the place block in their trip plan.

- FRQ-0220 Pagination: The system must paginate search results.

■ 2.1.3 Non-functional requirements

- NFR-01 Swagger documentation: The application must provide swagger REST API documentation for development purposes.
- NFR-02 System performance: The application can be used by several users simultaneously.
- NFR-03 User friendly interface: The application must have clear navigation and understandable UI.
- NFR-04 Support by popular browser: The application must work correctly in Google Chrome, Firefox, and Microsoft Edge browsers.
- NFR-05 Linux hosting server support: The application should function properly when deployed on a Linux server.

■ 2.2 Use Cases

This section is dedicated to Use Case diagram. A Use Case diagram captures the requirements of a system and offers a visual representation of the possible interactions between the system and entities external to the system [6]. These external entities are referred to as actors. Actors represent roles which may include human users, external hardware, or other systems.

The system involves the following actors:

- User – represents both authorized and unauthorized application users. Unauthorized users have limited access to application functionality which is searching for travel plans and viewing them.
- Authorized user – an individual who has logged into the application.
- Regular user – an authorized user without access to administrative functionality.
- Admin user – a user with access to administrative functionality. This user can approve and manage other users' posts and is responsible for ensuring that all users comply with the rules of conduct and public order. Nevertheless, it's important to note that administrative functionality will not be implemented within the scope of the bachelor project.

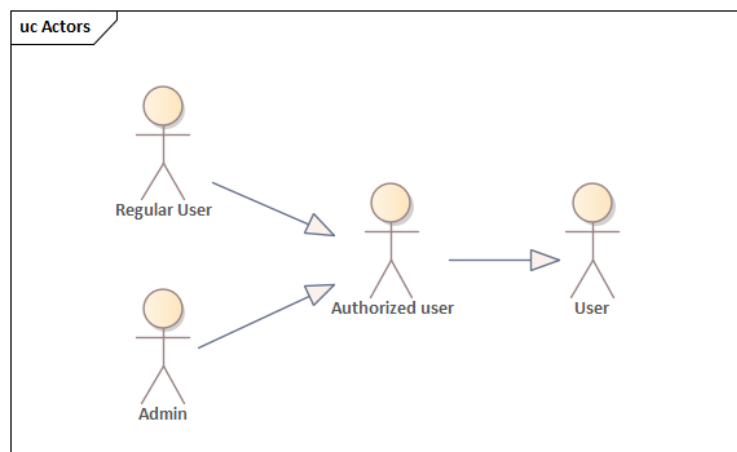


Figure 2.1. Actors.

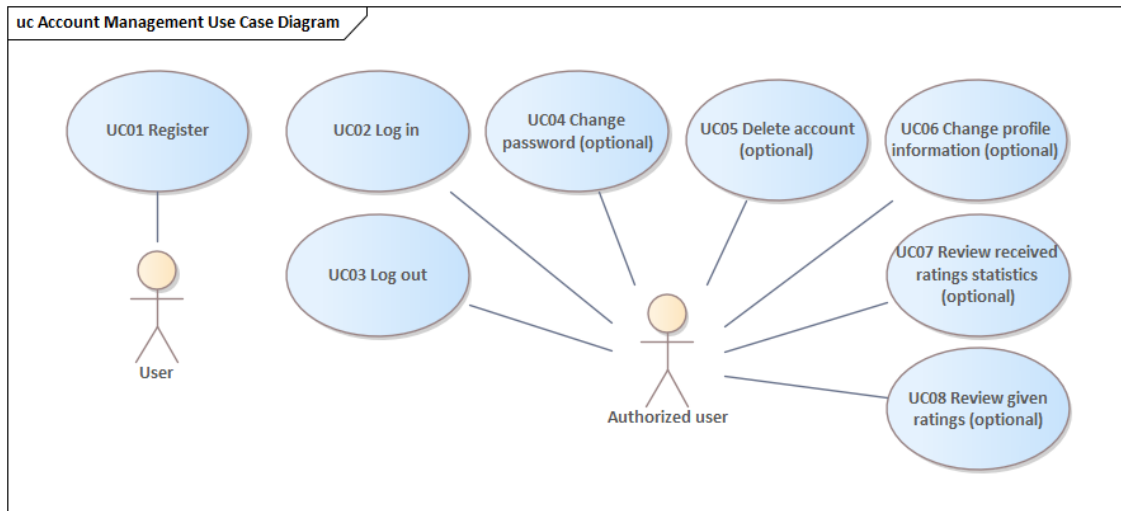


Figure 2.2. Account management Use Cases.

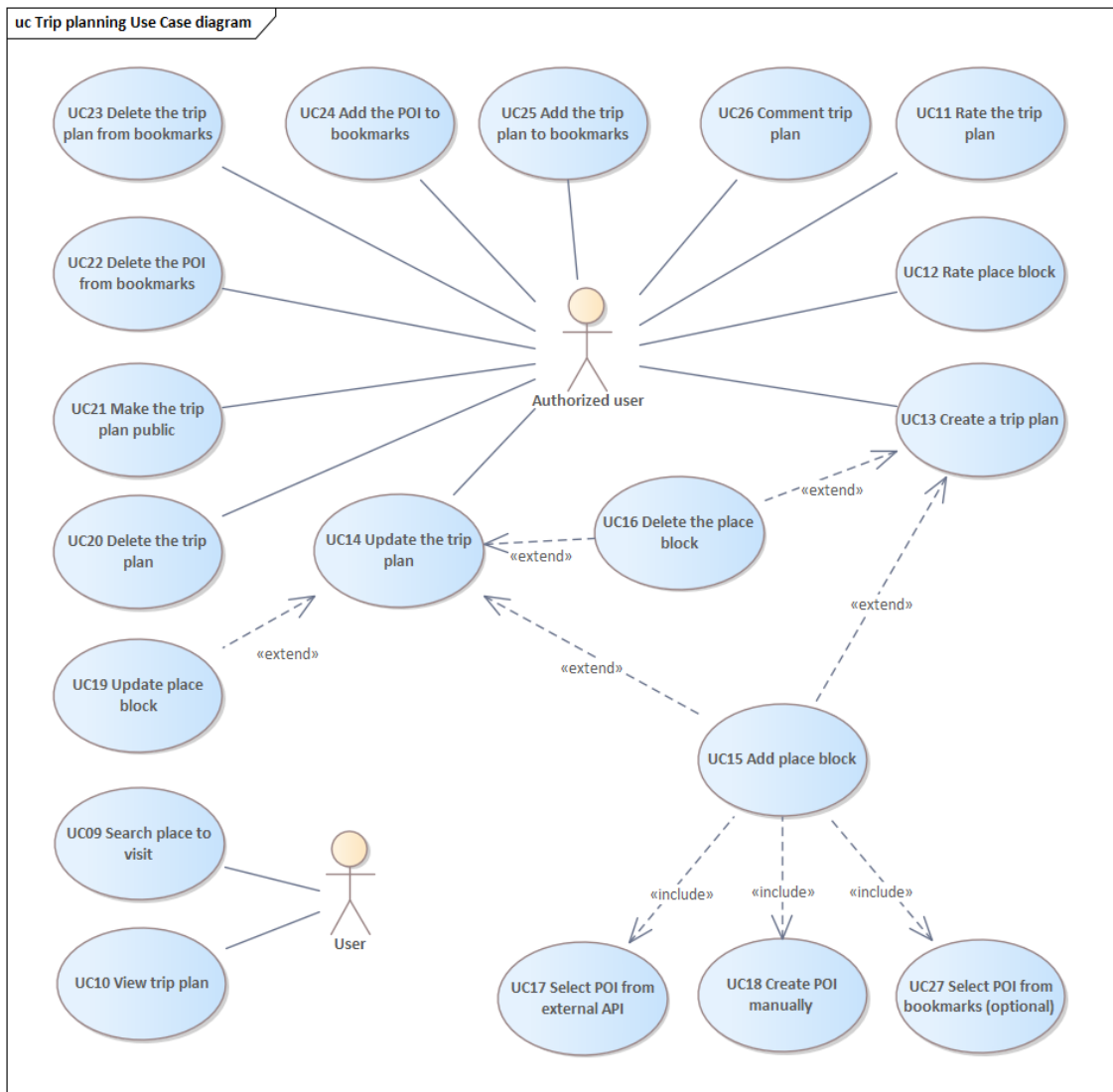


Figure 2.3. Trip planning Use Cases.

2.3 Business domain model

For data structurization and entities visualization the business domain model was created. The business domain model represents the conceptual view of the system and focuses on the key concepts, entities, and relationships within the business domain, without getting into technical details. It describes the core concepts of the business and the relationships between them.

A TripPlan class represents an entire blog post that consists of multiple places blocks (PlaceBlock). Each place block represents one POI. Such a block may have images (PlaceImage) and text about an experience of traveling to the place. Users may rate the entire post (TripPlanReaction) or standalone blocks (PlaceBlockReaction).

The POI class represents a place linked to Google Maps. The Point of Interest (POI) includes an address as a string. This address is not stored in a separate table because it is only used for display purposes on the user interface. There is no need to decompose the address into a standalone type, as all geographical operations are handled using the Google Maps ID, latitude, and longitude, rather than the address string.

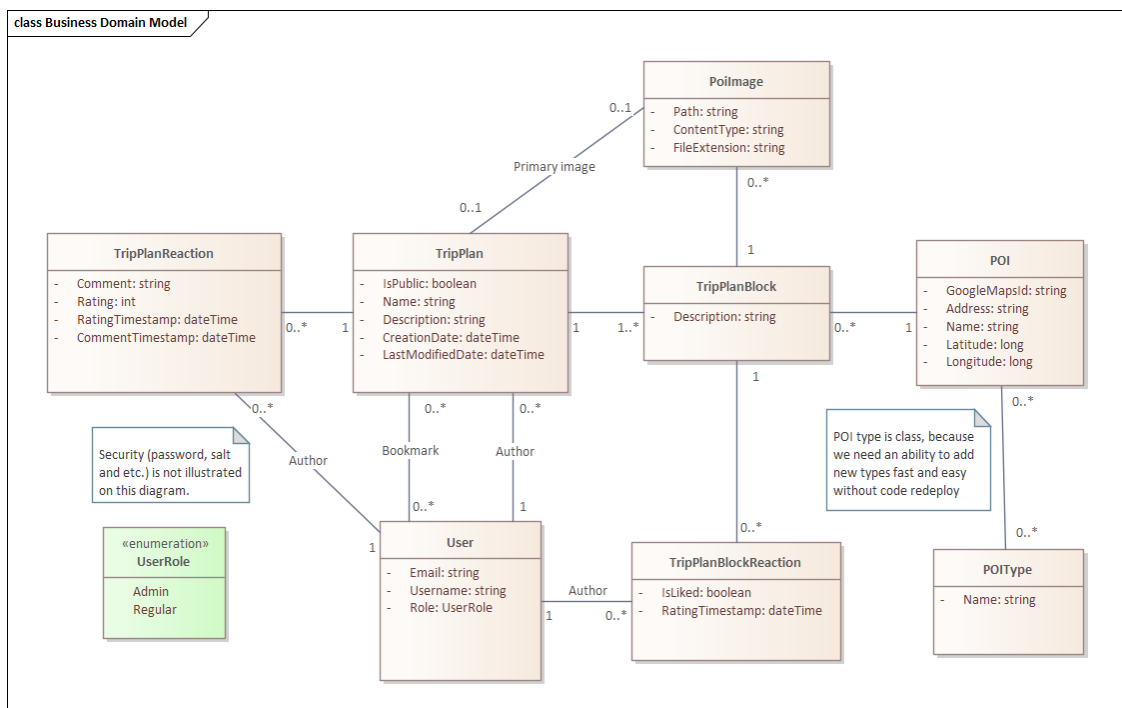


Figure 2.4. Business domain model.

The User can have the role of an administrator or a regular user. Nevertheless, in this project, administrative functionality will not be implemented.

2.4 Low fidelity prototype of GUI

The application must demonstrate adaptability, ensuring optimal presentation on both desktop and mobile devices. A low-fidelity prototype of the web application is presented in Appendix A.

2.5 Architecture

Due to the small size of the application and the low complexity of the business logic, a client-server architecture was chosen.

The client-server architecture is a system model involving client and server systems communicating over a network. Clients continuously connect to servers and send requests, while servers listen for and respond to requests from multiple clients [7]. In the current project, the frontend will act as the client, and the backend will function as the server.

In order to enhance the separation of concerns, the project will adopt a Three-Tier architecture that encompasses the following application layers [8–9]:

- The Presentation Layer is visible to the user. The user gives the inputs and instructions through this, and the output is also displayed on it. This layer encompasses the Angular frontend application and the backend API (Controllers). Furthermore, the frontend within this layer will use the external geographical data provider API.
- The Business Logic Layer contains models and logic, offering essential functionality to the application. This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also acts as an intermediate between the presentation and the data layer. Backend logic belongs to this layer.
- Data Access Layer stores and retrieves information from the database or filesystem. The retrieved data is then passed back to the logic tier for processing. Data manipulation on the backend belongs to this layer.

2.6 Backend technologies

There are numerous programming languages that allow the development of the backend for web applications and every language has its pros and cons. The programming language C# and the .NET platform were selected as the primary technologies for backend implementation. The primary reason for this is that I have sufficient experience in developing applications using this technology.

C# is a modern, open-source, cross-platform, strongly typed object-oriented programming language and one of the top 5 programming languages on GitHub and is consistently one of the most loved languages on Stack Overflow's developer survey [10–11]. It has a lot of syntactic sugar, which makes it very convenient and fast to develop with. A comparison of C# and Java was made by my colleague Vadym Rudenko in his bachelor project [12].

The .NET is a cross-platform and open-source set of runtime, library and compiler components which can be used in various configurations for building web, desktop, and mobile applications [13].

There still exists a misconception that C# and .NET are not cross-platform. This delusion is often mentioned when comparing C# to other programming languages. The brief history of the platform below will explain the origin of this misconception and dispel it.

The .NET platform evolution starts in 2002 with the .NET Framework which had a limitation of running exclusively on Windows operating systems [14]. This implies that web applications could only be hosted on Windows servers. In response to the framework's limitation, a new version named .NET Core was released in 2016. It was no longer limited to running on Windows OS. Both versions of the platform continued

to develop in parallel. With the release of .NET 5 in 2020, the “Core” part of the name was dropped, and the new platform versions were simply referred to as .NET [15].

2.7 Frontend technologies

Single-Page Applications (SPAs) have become a very common choice in building out frontend, as they allow for great customer experiences in terms of speed and responsiveness [16]. Once the application has loaded into a customer’s browser, further interactions only have to care about loading the additional data needed, without reloading the entire page.

The most popular SPAs frontend libraries according to Stack Overflow are React, Vue and Angular [11]. For this project, the Angular framework will be used due to my prior experience with this tool.

Angular enables developers to build scalable web applications with TypeScript, a strict syntactic superset of JavaScript [17]. Angular is based on the most modern web standards and supports all modern browsers. The power of the Angular platform is based on the combination of the following characteristics: cross-platform, advanced tooling, easy onboarding and worldwide usage.

2.8 Database

For the project’s objectives, a relational database is the most suitable choice. At the time of writing this thesis, the most widely favored database management systems are [18]:

- Oracle.
- MySQL.
- Microsoft SQL Server.
- PostgreSQL.
- Microsoft Azure SQL Database.

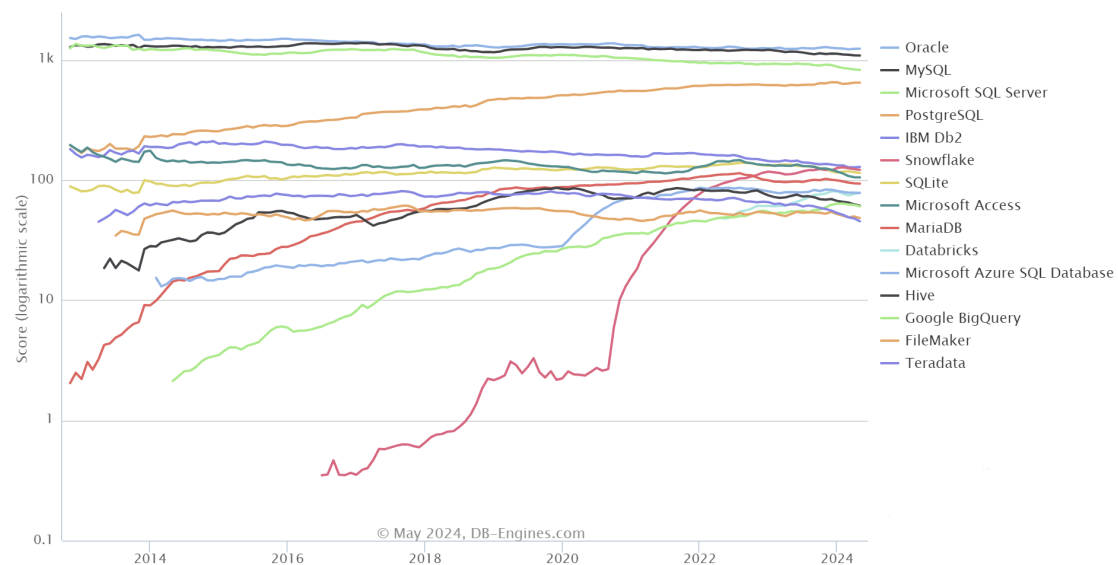


Figure 2.5. DB-Engines Ranking of Relational database engines [18].

Oracle, Microsoft SQL Server, and its cloud-based version Microsoft Azure SQL Database are highly reliable database engines widely utilized by large enterprise companies. Nevertheless, they are all commercial solutions with a very significant cost of licensing. While they offer free initial plans with limited functionality, as the project scales, the licensing fees can become prohibitively expensive.

MySQL is a fast, reliable, scalable and easy-to-use open-source relational database system designed to handle mission-critical, heavy-load production applications [19]. It is a common and easy-to-start database with low memory, disk and CPU utilization, managed by a relational database management system. MySQL Community Edition is a free downloadable version supported by an active online community.

PostgreSQL is an open-source relational database with a strong reputation for its reliability, flexibility and support of open technical standards [19]. PostgreSQL supports both non-relational and relational data types. It has been called one of the most compliant, stable and mature relational databases available today and can easily handle complex queries.

Category	MySQL	PostgreSQL
Database technology	Purely relational database management system.	Object-relational database management system.
Features	Limited support of database features like views, triggers, and procedures.	Supports most advanced database features like materialized views, instead of triggers, and stored procedures in multiple languages.
Data types	Supports numeric, character, date and time, spatial, and JSON data types.	Supports all MySQL data types along with geometric, enumerated, network address, arrays, ranges, XML, hstore, and composite.
ACID Compliance	ACID compliant only with InnoDB and NDB Cluster storage engines.	Always ACID compliant.
Indexes	B-tree and R-tree index support.	Supports multiple index types like expression indexes, partial indexes, and hash indexes along with trees.
Performance	Has improved performance for high frequency read operations.	Has improved performance for high frequency write operations.

Table 2.1. Key differences between MySQL and PostgreSQL [20].

Both MySQL and PostgreSQL are suited for the project. The decisive factor in choosing the DB engine is that PostgreSQL, when extended with PostGIS extension, becomes a powerful spatial database management system. It is well-regarded for its performance of geospatial databases and its versatility in mapping and spatial analysis

tools [21]. PostGIS provides robust geospatial data management and a variety of functions to enable GIS processing within the database itself, making PostgreSQL the preferred choice for the current project.

2.9 External APIs

One of the most critical aspects of the project is selecting the appropriate geographical data provider. All further development and user interaction will depend on the chosen data provider service, which must meet the following requirements:

- Provide an API for displaying an interactive map. There must be an ability to open the map directly at the provider service. For example, clicking on the map and opening it in a mobile application or website, which will provide more functionality for the user.
- Provide an API for places search.
- Provide an API for suggesting places nearby.
- The service must possess rich functionality to accommodate further development of the project.
- The service must be popular so that users will be familiar with it.

The criteria mentioned limit the choice to two services: Google Maps and Mapy.cz.

2.9.1 Google Maps

Google Maps' key strengths include high popularity, a large development community, data richness, rich functionality, and a substantial amount of documentation. The main disadvantage of the platform is the high cost for complex requests. However, at the time of writing this thesis, the platform does offer services worth 200 USD at no charge every month [22].

2.9.2 Mapy.cz

Mapy.cz has several key advantages, including data richness, a comparatively lower cost than Google, and a simple API. The platform offers 250,000 free credits (equivalent to approximately 17 USD) for the basic tariff and 10,000,000 credits (around 707 USD) for the extended tariff every month [23]. However, due to its simplicity, the API functionality is limited compared to Google.

2.9.3 API selection

When we look at the popularity of services over the last 12 months using Google Trends, Mapy.cz service is widely recognized in the Czech Republic but remains relatively unknown worldwide. Google Trends is a tool that provides insights into the relative popularity of search queries over time and across different regions. The numbers on the left axis of the charts represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. A score of 0 means there was not enough data for this term.

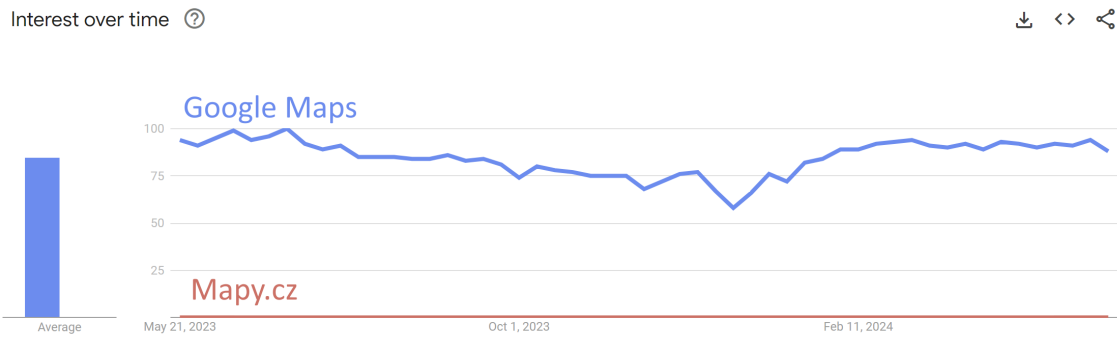


Figure 2.6. Global popularity comparison between Google Maps and Mapy.cz [24].

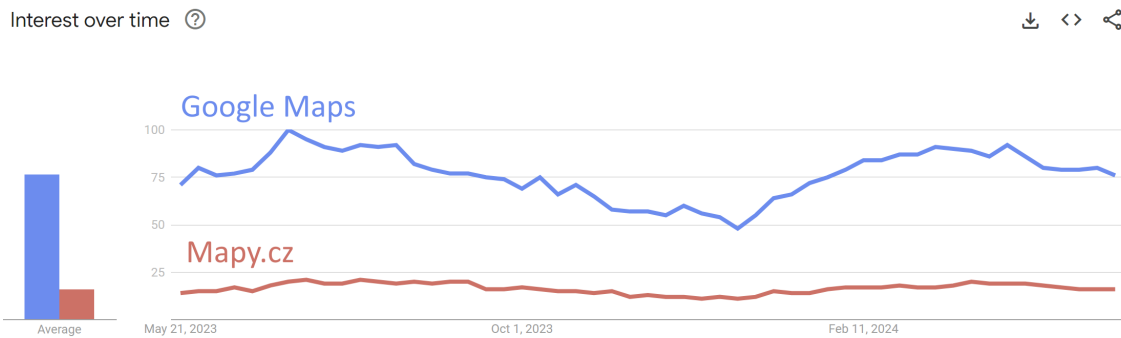


Figure 2.7. Comparison of Google Maps and Mapy.cz popularity in Czech Republic [24].

The significant advantage of Google Maps services is that Angular supports libraries for interacting with its API, making the development process easier, unlike Mapy.cz, which lacks any library support.

Considering the popularity of services, the richness of functionality, and the potential for further project growth, Google Maps has been chosen as the primary geographical data provider due to its extensive functionality and wide popularity.

Chapter 3

Implementation

3.1 Component diagram

The component diagram represents the architecture of a web application divided into two main subsystems: frontend and backend. The frontend project interacts with both the backend system and the Google Maps Platform.

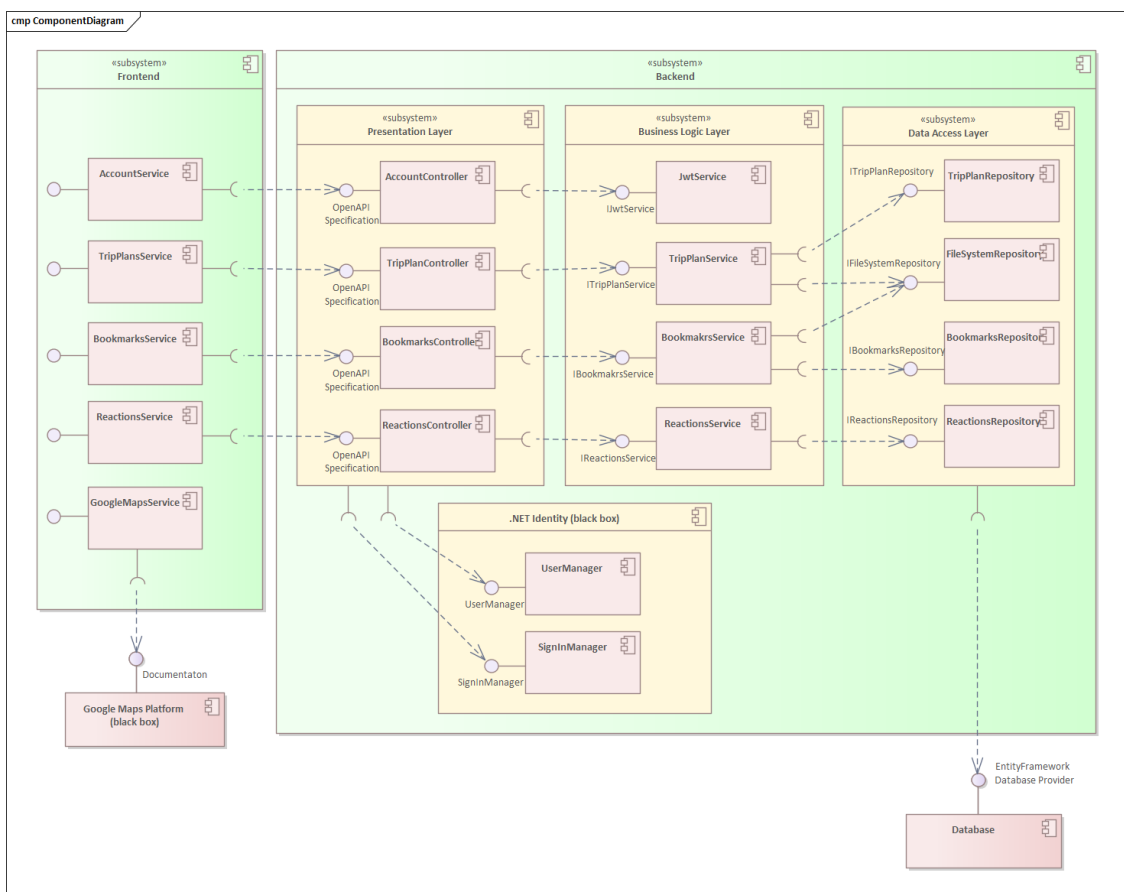


Figure 3.1. Component diagram.

The backend solution is divided into three layers:

- **Presentation Layer:** Contains controllers that handle incoming requests and direct them to the appropriate services in the Business Logic Layer.
- **Business Logic Layer:** Comprises various services that encapsulate the core functionalities, interfacing with repositories in the Data Access Layer.
- **Data Access Layer:** Consists of repositories that manage data persistence and retrieval, interfacing with a database via EntityFramework.

Additionally, the backend integrates with .NET Identity for user management.

3.2 Sequence diagram

The sequence diagram illustrates the primary functionality of creating a trip plan, which utilizes both the Google Maps Platform and the backend. The user can add multiple places of interest (POIs) to the trip plan. While adding a place, the user must type the place name or address. As the user types, requests are sent to the Google Maps Platform with each character entered, and suggestions are displayed. Once the desired place is found, the user selects it, triggering a final request to Google, which returns the place details, including the full address, latitude, and longitude. The selected place is then displayed on the map. When the trip plan is complete, the user can save it. The selected place is then displayed on the map. When the trip plan is complete, the user can save it.

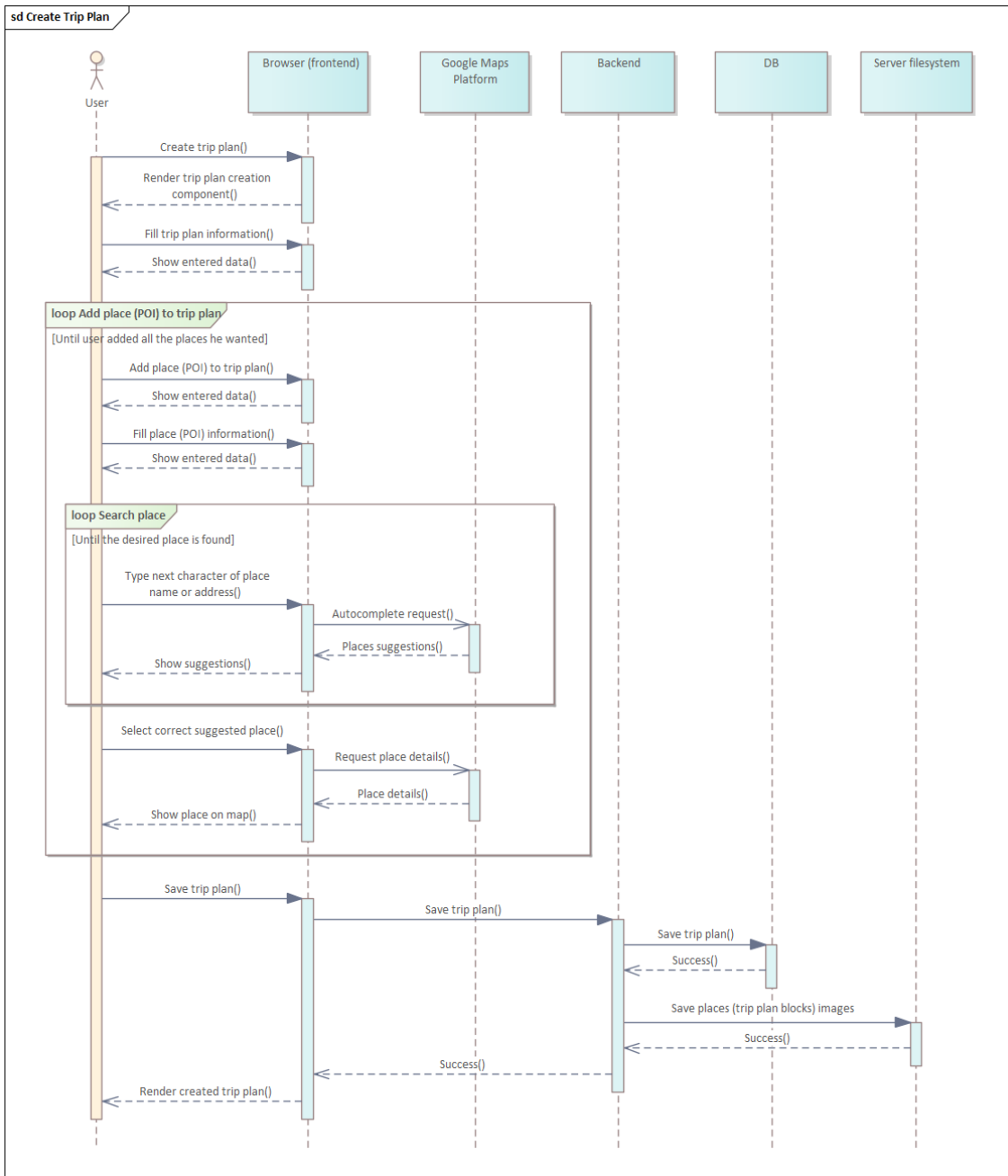


Figure 3.2. Sequence diagram illustrating trip plan creation.

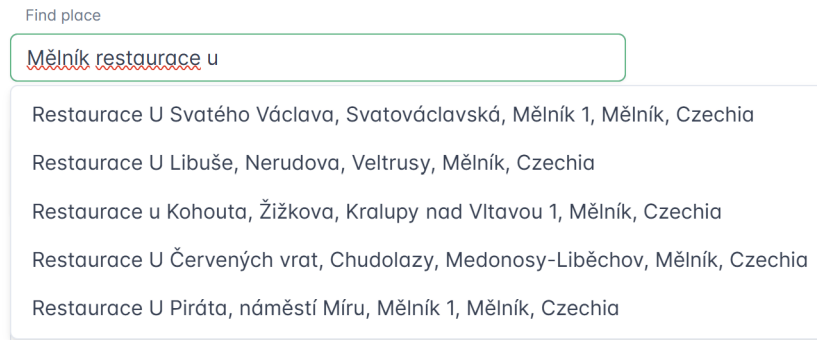


Figure 3.3. Suggested places example.

Other use cases are straightforward and primarily involve a single request-response pair.

3.3 Frontend

The Angular frontend application is structured as follows.

Components directory encapsulates reusable components that serve various functions and pages within the application. The “common” folder contains atomic components, each characterized by fundamental functionality, designed for reuse across various components and contexts. These include items such as carousel, comment block, embedded Google Map, Google Place autocomplete field, search trip plan form, trip plan block card and trip plan card, each encapsulated as individual components. This architecture provides substantial benefits, as any necessary changes to a component can be made in one centralized location, rather than having to update it in multiple instances across the application. Components located directly within the root of the components folder typically represent pages, such as the main page, search results page, trip plan page, bookmarks page and others.

Guards section contains the authorization guard, which serves to prevent unauthorized access to specific system components.

```
export const authorizationGuard: CanActivateFn = (route, state) => {
  const accountService = inject(AccountService);
  const router = inject(Router);

  return accountService.user$.pipe(
    map((userInfo: UserInfo | null) => {
      if (userInfo) {
        return true;
      } else {
        router.navigate(["sign-in"], { queryParams: { returnUrl: state.url } })
        return false;
      }
    })
  );
};
```

Figure 3.4. Authorization guard.

Interceptors folder incorporates authentication interceptor, responsible for appending authentication headers on outgoing requests. The interceptor checks if the request

URL matches the Google Maps API URL. If so, it modifies the request by adding Content-Type and X-Goog-API-Key headers with values retrieved from the environment configuration. Otherwise, it retrieves the JWT from an account service, and if available, adds an Authorization header with the JWT to the request. Finally, it forwards the modified request to the next interceptor or HTTP handler in the chain. This interceptor enables seamless integration of authentication mechanisms and facilitates communication with external services within the application.

```
export const authInterceptor: HttpInterceptorFn = (request, next) => {

  if (request.url.includes(environment.googlePlacesApiUrl)) {
    // Google Maps API request
    request = request.clone({
      setHeaders: {
        'Content-Type': 'application/json',
        'X-Goog-API-Key': environment.googleApiKey,
      }
    });
    return next(request);
  }

  const accountService = inject(AccountService);

  const jwt = accountService.getJwt();
  if (jwt) {
    request = request.clone({
      setHeaders: {
        Authorization: `Bearer ${jwt}`
      }
    });
  }

  return next(request);
};
```

Figure 3.5. Authentication interceptor.

Models directory holds data models that may be returned from both the backend and the Google Maps Platform. The “autogenerated” folder contains TypeScript interfaces that are generated from the backend C# DTOs. This process is described in the backend chapter of the documentation.

Pipes folder offers utility pipes, which can be used to manipulate data before presenting it on the UI. The *SafeUrlPipe* is utilized exclusively for rendering embedded Google Maps iframes. The necessity of this pipe arises from Angular’s default security behavior, which blocks iframes against Cross-Site Scripting (XSS) attacks. By employing the *bypassSecurityTrustResourceUrl* method within the pipe, the application explicitly trusts and permits the specified URL, thus overriding Angular’s protective measures.

```

@Pipe({
  name: 'safeUrl',
  standalone: true
})
export class SafeUrlPipe implements PipeTransform {

  constructor(private sanitizer: DomSanitizer) { }

  transform(url: any) {
    return this.sanitizer.bypassSecurityTrustResourceUrl(url);
  }
}

/* pipe usage in EmbedGoogleMapComponent
<iframe
  loading="lazy"
  allowfullscreen
  referrerpolicy="no-referrer-when-downgrade"
  [src]="googleMapsService.getGoogleMapsEmbeddedMapUrl(place) | safeUrl">
</iframe>*/

```

Figure 3.6. *SafeUrlPipe* and its usage.

Services section provides various services, including those responsible for fetching data from both the backend and the Google Maps Platform.

The *AccountService* is designed to handle various user-related functionalities. This service facilitates user registration, login, and management tasks such as refreshing user information, retrieving JWT tokens, and logging out users. It employs observables to manage user information, utilizing a *ReplaySubject* to store and broadcast user data updates to subscribed components.

The *GoogleMapsService* serves for integrating Google Maps API functionalities. It facilitates generating URLs for embedded maps based on specified locations and provides capabilities for place autocomplete and retrieving detailed information about the places. The service supports the management of session tokens for Google Places API requests, allowing to conclude sessions through the *finishSession* method.

The *TripPlansService* serves for managing trip plans and related operations. It offers functionalities such as retrieving trip plans based on search parameters, accessing user-specific trip plans, fetching details of a specific trip plan, obtaining trip plan suggestions, creating new trip plans, updating existing ones, adjusting their publicity status, and deleting trip plans.

The *BookmarksService* provides functionality for managing bookmarks related to trip plans and trip plan blocks. It facilitates the addition, retrieval, and deletion of bookmarks.

The *ReactionsService* facilitates the management of user reactions and interactions with trip plans and trip plan blocks. It enables users to retrieve their reactions to specific trip plans, rate trip plans, comment on trip plans, and rate individual trip plan blocks.

Subsequent chapters will offer insights into frontend development.

■ 3.3.1 Components library

Initially, I opted to utilize the free and open-source Angular Material component library due to its simplicity and official endorsement by the Angular Team at Google.

Angular Material offers a minimalist set of components, encompassing foundational elements such as buttons, forms, text fields, and more, while also providing support for basic customization [25]. It also provides a Google Maps Angular component that implements the Google Maps JavaScript API [26]. However, I encountered limitations within the library that were critical for application's needs.

The library lacked essential components such as an image carousel and a file upload button. While I could have developed these components independently, doing so would have consumed time and resulted in stylistic inconsistencies compared to other components. Exploring alternative solutions, I found that libraries offering single image carousel and file upload button components sometimes were not free or had vastly different styles. Additionally, many of these alternatives were outdated or lacked active community support.

Considering the factors mentioned above, I had to search for another component library and transition the project to the new library.

After careful consideration, I selected the PrimeNG components library. It offers an extensive component suite, has positive feedback from the community, and is utilized in some real-world production applications [27]. Additionally, it is open-source and offers a variety of themes, along with the availability of a paid plan, which offers extended support for commercial products.

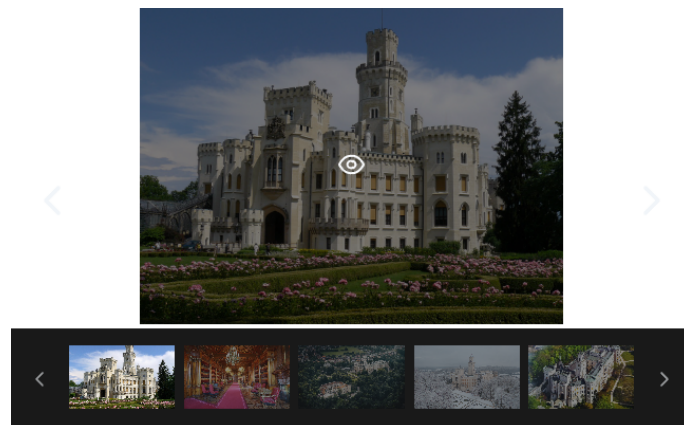


Figure 3.7. PrimeNG gallery view.

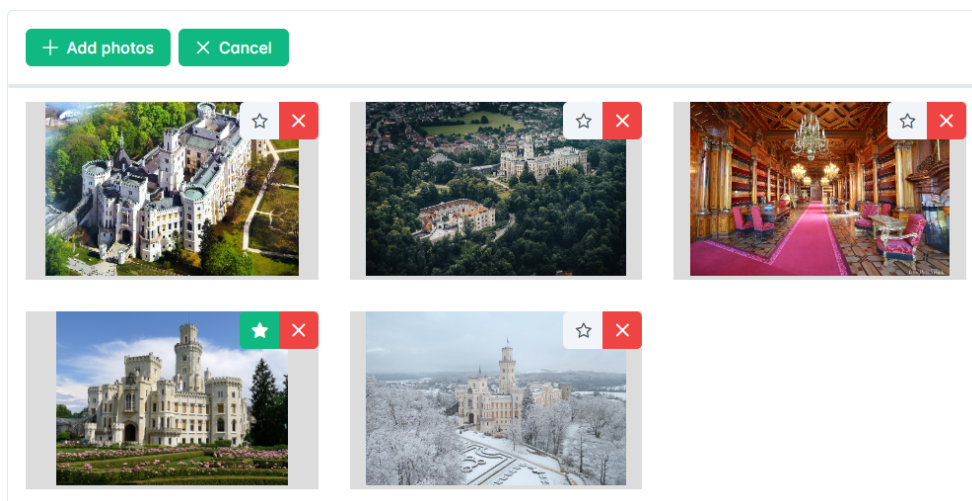


Figure 3.8. PrimeNG file upload component customized with gallery view and actions.

Working with PrimeNG proved to be an exceptional experience, owing to its comprehensive suite of components and extensive configurability.

This experience taught me the importance of thoroughly reviewing a library's components before integration. I also realized that such decisions should be made during the analysis phase rather than the implementation phase.

■ 3.3.2 Files upload

Files upload implementation was an interesting challenge that I never solved before. I discovered that a reliable method for uploading files to the backend is to send a request encrypted as multipart/form-data, which transmits images as binary data. The .NET platform provides the `IFormFile` type to manage the received files. However, this approach introduces a complication, requiring the development of logic for mapping complex custom data types to a `FormData` type, which is essentially a collection of key-value pairs. This task becomes particularly complex when dealing with types that contain multiple levels of nested structures. As a result, implementing and maintaining such mapping logic, especially in a generic manner, can pose significant challenges.

■ 3.3.3 Google Maps Platform

All interactions with the Google Maps Platform within the project occur on the frontend. I initiated my work with the Google Maps API by attempting to render a basic map. There are several methods available for rendering the map, and there are some libraries that can assist with this task. One such library is Angular Google Maps (AGM) [28], which has not seen any commits for over a year as of the time of writing this document. This library experienced a decline in popularity after the map component was incorporated into the Angular Material library and became an officially supported option. Hence, I gave preference to Angular Material.

When I experimented with the map component from the Angular Material library, I discovered that it utilizes the Google Maps JavaScript API [29], which comes with a cost [30]. However, I was aware that Google offers similar functionality for free through the Maps Embed API [31]. All requests made to the Maps Embed API are provided at no cost and come with unlimited usage [32].

The distinction between APIs lies in the fact that the Maps JavaScript API is more complex, allowing interaction with the map by writing JavaScript and offering a wider array of options and configurations. On the other hand, the Maps Embed API is much simpler – it merely consists of an HTML iframe with a source link that specifies the location to be displayed. The free tier of the Maps Embed API is entirely sufficient for this project, which is why I opted for it instead of the Angular Material Google Maps component.

For the project's purposes the following requests are required:

- An autocomplete request, which attempts to find a place based on the provided query string. It should provide place suggestions based on the place name or address entered by the user. Such requests only return very simple data such as place name, Google Maps ID, and address.
- A Place Details request, which will return detailed information about a place. It takes the place ID and a list of required data fields that should be returned, such as latitude, longitude, detailed address, opening hours, and photos.

The next step I did was implementing the place autocomplete functionality. This feature suggests autocompleted options to users as they type the name or address of

a place, drawing from known locations on Google Maps. This functionality is facilitated by the Google Places API, which offers two versions: the existing Places API and the newer Places API (New) [33]. According to Google, the new version boasts enhanced security, efficiency, performance, and features, and recommends its use for new projects [34]. However, certain endpoints of the new API are labeled as **Preview** and are available for use free of charge. As the project implementation progressed, some of these endpoints have since transitioned to a **Stable** label. Considering the factors listed above, I chose to use the new version, which is the Places API (New).

```
public getGoogleMapsEmbeddedMapUrl(place: string): string {
  const url = environment.googleEmbedMapUrl;
  const apiKey = environment.googleApiKey;
  return `${url}/place?key=${apiKey}&q=${place.replace(" ", "+")}`;
}

public placeAutocomplete(queryParams: AutocompletePlacesRequest):
  Observable<PlaceSlimModel[]> {
  const url = `${environment.googlePlacesApiUrl}:autocomplete`;
  return this.http.post<AutocompletePlacesResponse>(url, queryParams)
    .pipe(
      map(response => response.suggestions.map(item => {
        return ({
          googleMapsId: item.placePrediction.placeId,
          shortName: item.placePrediction.structuredFormat.mainText?.text,
          fullName: item.placePrediction.text.text
        }) as PlaceSlimModel
      })
    );
}

public placeDetails(placeId: string, sessionToken: string):
  Observable<PlaceDetailsResponse> {
  const url = `${environment.googlePlacesApiUrl}/${placeId}?sessionToken=${sessionToken}`;
  return this.http.get<PlaceDetailsResponse>(url, this.placeDetailsHttpsOptions);
}
```

Figure 3.9. Methods in Angular Google Maps service.

The Places API (New) offers a wider range of billing plans, providing increased flexibility and potential cost savings. This updated API incorporates functionality from both the existing Places API and Geocoding API, providing a comprehensive suite of services.

Google’s billing structure is based on the data fields that are queried. Each field is assigned to a billing tier, and billing is determined by the price of the most expensive field queried. Therefore, it is crucial to only request the necessary data to avoid unnecessary costs.

Another factor that can contribute to cost savings on requests is the usage of sessions [35]. Without utilizing sessions, billing is calculated individually for each request. This becomes particularly significant in the context of autocomplete functionality, where requests are made with each typed letter. By employing sessions, each autocomplete request must include a session token, which is a randomly generated UUID (Universally Unique Identifier). Once a place is identified through autocomplete, a Place Details request should be made using the same session token. Subsequently, when the session terminates, it results in charges equivalent to a single Place Details call [36].

Google Maps Platform is currently transitioning some functionality to gRPC. However, I used the REST API for communication since it is simple, and Google Maps Platform has not migrated most of endpoints to gRPC yet.

3.3.4 GUI

The graphical user interface was implemented according to a low fidelity prototype. The GUI screenshots are presented in Appendix B.

3.4 Backend

The .NET backend solution was structured as a Three-Tier application, consisting of three distinct projects, each representing layers of the Three-Tier architecture. Given the well-encapsulated layers, the projects can be smoothly transitioned to a microservice architecture should there be a need for expansion. To achieve this, the library types of projects should be transformed into executable application types, and appropriate communication channels should be established.

The subsequent chapters will describe the implementation of each layer and will highlight aspects related to backend development.

3.4.1 Data Access Layer

The *OneDayTrip.DataAccess* library project serves as the Data Access Layer (DAL).

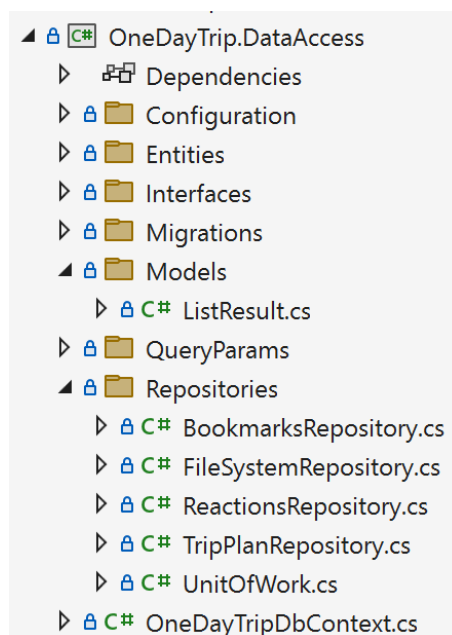


Figure 3.10. *OneDayTrip.DataAccess* project structure.

This project is composed of Entities and Repositories that facilitate access to the database and file system. It employs the Entity Framework Object-relational mapping (ORM) for database access, featuring a DB context that configures all tables and their relationships. The DB context extends the *IdentityDbContext*, eliminating the need to manually implement user-related code such as user entity, user role entity and user claims entity.


```

modelBuilder.Entity<Poi>().HasKey(x => x.Id);
modelBuilder.Entity<Poi>().Property(x => x.Location)
    .HasColumnType("geography (point)");
modelBuilder.Entity<Poi>()
    .HasMany(x => x.PoiTypes)
    .WithMany(x => x.Pois);

```

Figure 3.11. POI entity configuration in DB context.

The project employs a code-first approach, whereby the Entity Framework generates the database schema based on the Entities and DB Context configuration. Essentially, the database schema is derived from the code. Additionally, this layer supports migrations, a method within the Entity Framework designed to incrementally update the database schema, ensuring it remains synchronized with the application's data model while preserving the existing data in the database.

```

public partial class TripPlanBlockReactions : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "TripPlanBlockReactions",
            columns: table => new
            {
                UserId = table.Column<string>(type: "text", nullable: false),
                TripPlanBlockId = table.Column<int>(type: "integer", nullable: false),
                IsLiked = table.Column<bool>(type: "boolean", nullable: false),
                RatingTimestamp = table.Column<DateTime>(type:
                    "timestamp with time zone", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_TripPlanBlockReactions",
                    x => new { x.TripPlanBlockId, x.UserId });
                table.ForeignKey(
                    name: "FK_TripPlanBlockReactions_AspNetUsers_UserId",
                    column: x => x.UserId,
                    principalTable: "AspNetUsers",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Cascade);
                table.ForeignKey(
                    name: "FK_TripPlanBlockReactions_TripPlanBlocks_TripPlanBlockId",
                    column: x => x.TripPlanBlockId,
                    principalTable: "TripPlanBlocks",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Cascade);
            });

        migrationBuilder.CreateIndex(
            name: "IX_TripPlanBlockReactions_UserId",
            table: "TripPlanBlockReactions",
            column: "UserId");
    }
    // ...
}

```

Figure 3.12. Generated migration that adds *TripPlanBlockReactions* table.

3.4.2 Business Logic Layer

The *OneDayTrip.BusinessLogic* library project represents the Business Logic Layer (BLL). This project contains models and services. It references the *OneDayTrip.DataAccess* project, allowing services to utilize methods from the DAL. Models can differ from entities in the Data Access layer, which is why automated generic type mapping was implemented, with AutoMapper configuration located in the class *AutoMapperProfile*.

Services encapsulate business logic, executing complex tasks with the assistance of the DAL, and performing additional data manipulations. Additionally, there is a service called *JwtService* responsible for generating authentication JWT.

```

/// <summary>
/// Generates an encoded JWT (JSON Web Token) for a user.
/// </summary>
/// <param name="userId">The ID of the user for whom the token is being generated.</param>
/// <returns>JWT as a string, ready for use in HTTP headers for authentication.</returns>
public string GenerateToken(string userId)
{
    // !!! No sensitive data must be in claims !!!
    // Initialize a list of claims for the JWT.
    // These include the user's ID and device information,
    // a unique identifier for the JWT, and the time the token was issued.
    var claims = new List<Claim>
    {
        new(ClaimTypes.NameIdentifier, userId),

        new(JwtRegisteredClaimNames.Sub, userId),
        new(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new(JwtRegisteredClaimNames.Iat, DateTime.UtcNow.Ticks.ToString(),
            ClaimValueTypes.Integer64),
    };

    // Create the JWT security token and encode it.
    // The JWT includes the claims defined above, the issuer and audience from
    // the config, and an expiration time.
    // It's signed with a symmetric key, also from the config,
    // and the HMAC-SHA256 algorithm.
    var jwt = new JwtSecurityToken(
        issuer: _jwtSettings.Issuer,
        audience: _jwtSettings.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddMinutes(_jwtSettings.ExpiresInMinutes),
        signingCredentials: new SigningCredentials(
            _symmetricSecurityKey, SecurityAlgorithms.HmacSha256Signature)
    );

    // Convert the JWT into a string format that can be included in an HTTP header.
    var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);

    return encodedJwt;
}

```

Figure 3.13. Token generation method in JWT service.

```

public async Task<ListResult<TripPlan>> GetTripPlansAsync(SearchTripPlanParams searchParams)
{
    var queryParams = _mapper.Map<DataAccess.QueryParams.SearchTripPlanParams>(searchParams);

    var tripPlans = await _unitOfWork.TripPlanRepository.GetTripPlansAsync(queryParams);

    foreach (var tripPlan in tripPlans.Items)
    {
        if (tripPlan.PrimaryImage != null)
        {
            await _unitOfWork.FileSystemRepository
                .EnrichTripPlanBlockImage(tripPlan.PrimaryImage);
        }
    }

    var result = _mapper.Map<ListResult<TripPlan>>(tripPlans);

    return result;
}

```

Figure 3.14. Searching trip plan with parameters implementation.

■ 3.4.3 Presentation Layer

The *OneDayTrip* project serves as the main executable and represents the Presentation Layer. It is a monolithic project that connects all components and manages dependency injection. This layer contains Controllers and Data Transfer Objects (DTOs). Some DTOs support validation, implemented with the help of validation attributes. The controllers implement the REST API, consume JSON, and respond with JSON. They also provide Swagger documentation, adhering to the OpenAPI specification. Additionally, Controllers utilize .NET Identity for user management. This layer also includes configuration for the *Reinforced.Typings* library, which generates TypeScript contracts from C# DTOs.

```

public class RegistrationDto
{
    [Required]
    [TrimStringLength(5, 20)]
    public string UserName { get; set; } = string.Empty;

    [Required]
    [EmailAddress]
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; } = string.Empty;

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; } = string.Empty;

    [Required]
    [Compare("Password",
        ErrorMessage = "The password and confirmation password do not match.")]
    public string RepeatPassword { get; set; } = string.Empty;
}

```

Figure 3.15. DTO validation using validation attributes.

The screenshot displays the Swagger UI for the 'OneDayTrip' API (version 1.0, OAS3). The endpoints are organized into four main sections:

- Account:**
 - POST /api/account/registration
 - POST /api/account/login
 - GET /api/account/user-token-refresh
- Bookmarks:**
 - POST /api/bookmarks/trip-plans/{tripPlanId}
 - DELETE /api/bookmarks/trip-plans/{tripPlanId}
 - GET /api/bookmarks/trip-plans
 - GET /api/bookmarks
 - POST /api/bookmarks/trip-plan-blocks/{tripPlanBlockId}
 - DELETE /api/bookmarks/trip-plan-blocks/{tripPlanBlockId}
 - GET /api/bookmarks/trip-plan-blocks
- Reactions:**
 - GET /api/reactions/trip-plan/{tripPlanId}
 - POST /api/reactions/trip-plan/{tripPlanId}/rating
 - POST /api/reactions/trip-plan/{tripPlanId}/comment
 - POST /api/reactions/trip-plan-block/{tripPlanBlockId}/rating
- TripPlan:**
 - GET /api/trip-plans
 - POST /api/trip-plans
 - PUT /api/trip-plans
 - GET /api/trip-plans/user-trip-plans
 - GET /api/trip-plans/{id}
 - DELETE /api/trip-plans/{id}
 - GET /api/trip-plans/suggestions
 - PATCH /api/trip-plans/publicity/{id}

Figure 3.16. List of endpoints in Swagger.

The following chapters will clarify the libraries used and highlight important aspects of the backend project.

■ 3.4.4 Dependency injection

.NET supports the dependency injection (DI) software design pattern, which is a technique for achieving Inversion of Control (IoC) between classes and their dependencies. Dependency injection in .NET is a built-in part of the framework, along with configuration, logging, and the options pattern [37].

Services can be registered with one of the following lifetimes:

- **Transient.** Transient lifetime services are created each time they're requested from the service container.

- **Scoped.** For web applications, a scoped lifetime indicates that services are created once per client request (connection). In apps that process requests, scoped services are disposed at the end of the request.
- **Singleton.** Singleton lifetime services are created either the first time they're requested or by the developer when an implementation instance is provided directly to the container. Every subsequent request of the service implementation from the dependency injection container uses the same instance.

The `OneDayTrip` as the main executable project handles dependency injection and registered dependencies.

```
builder.Services.AddSingleton(_ => new DbConnectionConfiguration
{
    ConnectionString = builder.Configuration.GetConnectionString("OneDayTrip")
    ?? throw new Exception("OneDayTrip connection string is missing")
});

builder.Services.AddDbContext<OneDayTripDbContext>();

var tripPlanImagesDirectory =
    builder.Configuration.GetValue<string>("TripPlanImagesDirectory")
    ?? throw new Exception("Value TripPlanImagesDirectory is missing in config");

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>(x =>
    new UnitOfWork(x.GetRequiredService<OneDayTripDbContext>(), tripPlanImagesDirectory));
```

Figure 3.17. DAL dependencies registration.

When registering services, it is important to remember and avoid the common “captive dependency” antipattern. The term “captive dependency” was coined by Mark Seemann, and refers to the misconfiguration of service lifetimes, where a longer-lived service holds a shorter-lived service captive [38]. This mismatch can lead to improper resource management, memory leaks, and unintended behavior because the shorter-lived service might be disposed of while the longer-lived service still expects it to be available. This undermines the design principles of dependency injection and can result in hard-to-debug issues.

The following picture illustrates an example where services couldn't be made singleton because they hold an *IUnitOfWork* implementation with a scoped lifetime. Conversely, the *IJwtService* doesn't have any limitations and doesn't hold state, which is why it could be made singleton without any issue. Thus, understanding the lifetime scope of services is crucial for effective implementation.

```
builder.Services.AddScoped<ITripPlanService, TripPlanService>();
builder.Services.AddScoped<IReactionsService, ReactionsService>();
builder.Services.AddScoped<IBookmarksService, BookmarksService>();
builder.Services.AddSingleton<IJwtService, JwtService>(_ => jwtService);
```

Figure 3.18. Correct services registration in DI container.

The container is responsible for cleanup of types it creates. Services resolved from the container should never be disposed by the developer. If a type or factory is registered as a singleton, the container disposes the singleton automatically.

3.4.5 PostGIS

One of the most important functionalities in the application is searching for places by departure and destination. Using geographical coordinates (longitude and latitude) retrieved from the Google Maps API, it is possible to filter places based on their proximity to the departure and destination locations. This can be achieved with the help of the PostGIS extension for PostgreSQL.

PostGIS enhances the PostgreSQL relational database by adding support for storing, indexing, and querying geospatial data [39]. This extension must first be installed and enabled in the PostgreSQL database [40].

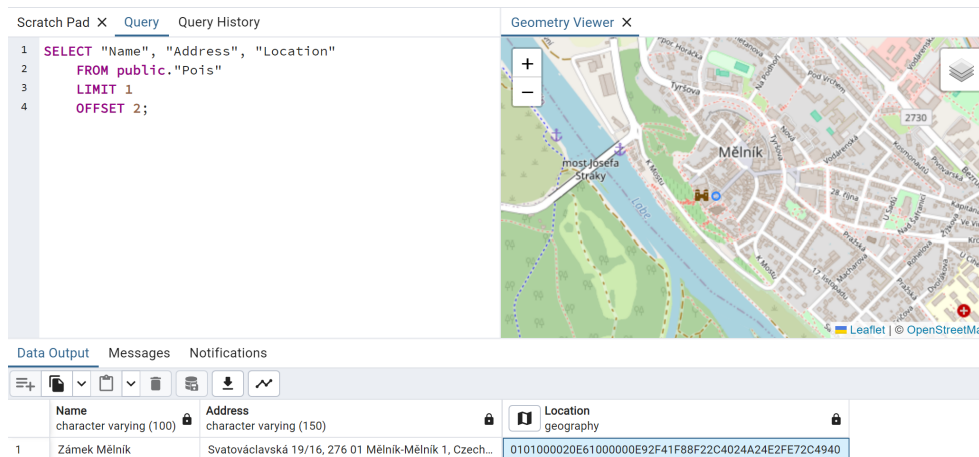


Figure 3.19. Displayed saved geography data in pgAdmin 4 Geometry Viewer.

In PostGIS It is very common to have data in which the coordinate are “geographics” or “latitude/longitude”. It is important to understand that geographic coordinates are not Cartesian coordinates. Geographic coordinates do not represent a linear distance from an origin as plotted on a plane [41]. Rather, these spherical coordinates describe angular coordinates on a globe. In spherical coordinates a point is specified by the angle of rotation from a reference meridian (longitude), and the angle from the equator (latitude).

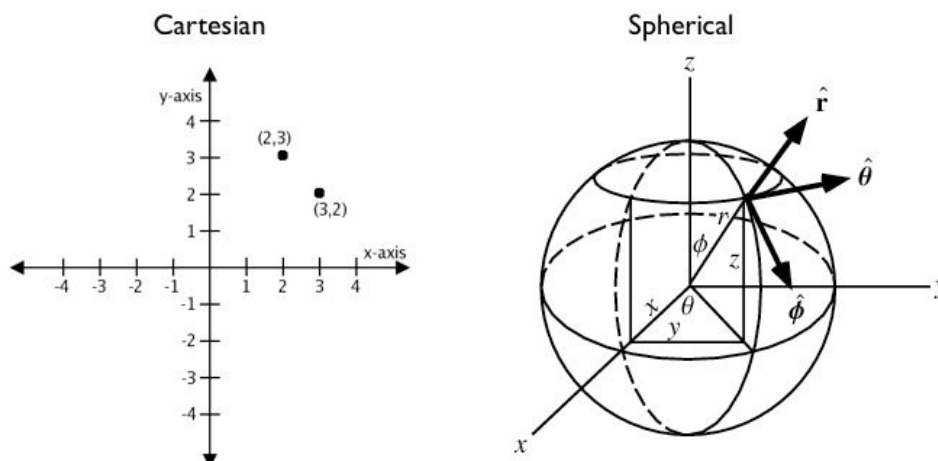


Figure 3.20. Visualization of Cartesian and Spherical coordinate systems.

You can treat geographic coordinates as approximate Cartesian coordinates and continue to perform spatial calculations [41]. However, measurements of distance, length

and area will be inaccurate. Since spherical coordinates measure angular distance, the units are in degrees. Furthermore, approximate results from indexes and true/false tests, such as intersects and contains, can become highly inaccurate. The distance between points increases significantly in problematic areas, such as near the poles or the International Date Line.

In GIS, there is an important identifier called the Spatial Reference System ID (SRID). Every geometric shape has an associated spatial reference system, and each reference system has an SRID [42]. A common SRID in use is 4326, which represents spatial data using longitude and latitude coordinates on the Earth's surface as defined by the WGS84 standard. This standard is also used for the Global Positioning System (GPS). Google Maps provides coordinates in the form of longitude and latitude, which correspond to SRID 4326.

To utilize the functionality of PostGIS with Entity Framework, the NetTopologySuite library must be installed, and the DB context must be configured [43]. Once this is done, the PostGIS method Distance will be available for use. In my case, I am sorting the places by distance from the departure location first, and then by distance from the destination location, resulting in a list of nearby places relative to the specified departure and destination locations.

```
query = query
    .OrderBy(x => x.TripPlanBlocks
        .Min(y => y.Poi.Location.Distance(searchTripPlanParams.DepartureLocation)))
    .ThenBy(x => x.TripPlanBlocks
        .Min(y => y.Poi.Location.Distance(searchTripPlanParams.DestinationLocation)));
```

Figure 3.21. Usage of PostGIS with the help of NetTopologySuite library and Entity Framework.

3.4.6 Storing uploaded files

One of the core functionalities of the project is the ability to upload POI images. There are multiple solutions for storing uploaded files on the backend. Most modern databases support file storage directly within the database. Alternatively, files can be stored on a disk. Each solution comes with its own set of advantages and disadvantages [44].

Storing files in a database offers convenience in terms of data querying and maintaining data consistency. Conversely, storing files on a disk necessitates additional effort, including managing CRUD operations, ensuring consistency with the database state, and upholding security. Tables containing images can be configured for cascade deletion, ensuring that images are deleted when the associated main entity is deleted. When opting for filesystem storage, such considerations must be carefully managed. Additionally, databases can be configured for backups and retention policies, which may be more challenging with disk storage.

Nevertheless, storing files in a database can lead to database overload and performance degradation, particularly with large image files. This can result in slower queries and increased memory requirements. From a cloud perspective, scaling disk space is generally easier and more cost-effective than scaling the database.

It is widely accepted to use databases for storing small files, such as profile pictures, while the filesystem is recommended for larger files. Considering these factors, I have decided to store POI images on disk. Additionally, for future project enhancement, a Content Delivery Network (CDN) can be utilized. A CDN is a network of interconnected servers that speeds up webpage loading for data-heavy applications [45]. When a user

visits a website, data from that website’s server has to travel across the internet to reach the user’s computer. If the user is located far from that server, it will take a long time to load a large file. Instead, the website content is stored on CDN servers geographically closer to the users and reaches their computers much faster.

■ 3.4.7 Data mapping

Due to the separation of layers, a challenge arises regarding data mapping. Each encapsulated project possesses its own contracts, which may vary. To address this, I utilized the AutoMapper library [46]. By simply specifying the types in AutoMapper configuration, all fields with matching names are automatically mapped from the source type to the destination type. The library is configurable and supports the exclusion of fields, as well as custom type and field mapping.

```

CreateMap<Models.TripPlanReaction, DataAccess.Entities.TripPlanReaction>();
CreateMap<DataAccess.Entities.TripPlanReaction, Models.TripPlanReaction>()
    .ForMember(
        dest => dest.AuthorUserName,
        opt => opt.MapFrom(
            src => src.User.UserName))
    .ForMember(
        dest => dest.AuthorId,
        opt => opt.MapFrom(
            src => src.UserId));

```

Figure 3.22. *TripPlanReaction* mapping configuration between DAL entity and BLL model.

■ 3.4.8 Generating TypeScript contracts from C# DTOs

It is a common situation during development that data contracts change rapidly. By contracts, I refer to DTOs returned by the backend and received by the frontend. Whenever the contract changes on the backend, the frontend contract must be updated accordingly. For example, if a field in the backend contract is renamed or deleted, the frontend will not throw an error, and the field will simply be empty. This can lead to errors and lack of transparency, making it challenging to maintain consistent contracts.

Using gRPC and Protocol Buffers contracts could solve this problem. However, I chose not to implement gRPC support because I aimed to make the backend and frontend communication as quick and straightforward as possible. Adopting gRPC would have taken more time, as I have not previously used gRPC in Angular projects and could not dedicate much time to it.

To address this problem, I knew a simple solution in the Reinforced.Typings library [47]. It offers a straightforward method for generating TypeScript contracts from C# classes. Once configured, TypeScript contracts will be automatically regenerated with every backend build.


```

builder.ExportAsInterface<TripPlanDto>()
  .WithPublicProperties()
  .WithProperty(x => x.CreationDate,
    config => config.Type(TsDateTypeName))
  .WithProperty(x => x.LastModifiedDate,
    config => config.Type(TsDateTypeName))
  .AutoI(false)
  .OverrideName("TripPlan")
  .ExportTo("trip-plan.ts");

```

Figure 3.23. Trip plan TypeScript contract generation configuration.

```

// This code was generated by a Reinforced.Typings tool.
// Changes to this file may cause incorrect behavior and will be lost if
// the code is regenerated.

import { PoiImage } from './poi-image';
import { TripPlanBlock } from './trip-plan-block';
import { TripPlanReaction } from './trip-plan-reaction';

export interface TripPlan
{
  id: number;
  name: string;
  description: string;
  creationDate: Date;
  lastModifiedDate: Date;
  isPublic: boolean;
  rating: number;
  primaryImage: PoiImage;
  tripPlanBlocks: TripPlanBlock[];
  reactions: TripPlanReaction[];
  authorId: string;
  authorUserName: string;
}

```

Figure 3.24. Generated trip plan contract.

There are more complex solutions available [48]. However, simply generating contracts suffices for the project's needs and proved to be a rapid and efficient solution.

■ 3.4.9 Unit of Work pattern

The Unit of Work pattern was used on Data Access Layer. This widely adopted design pattern helps to manage transactions and maintain data consistency in applications. The Unit of Work pattern is used to manage transactions and ensure that multiple operations are treated as a single logical unit [49]. It provides a way to group database operations together, ensuring that they either succeed or fail as a whole. The key principle behind the Unit of Work pattern is to maintain data consistency and integrity by committing or rolling back changes in a coordinated manner.

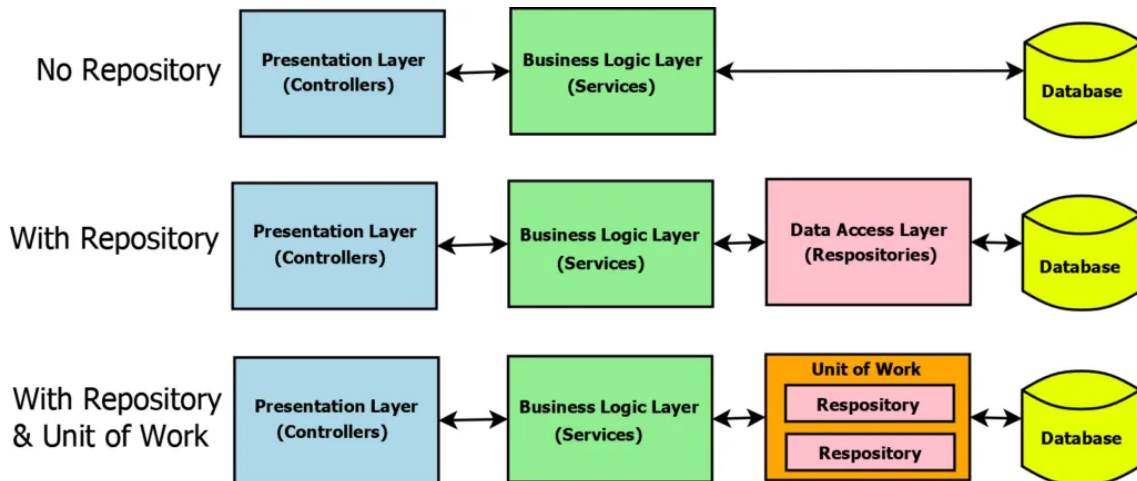


Figure 3.25. Unit of Work pattern visualization.

■ 3.4.10 Security

For application security, which includes authentication and authorization, I used the .NET Core Identity library. This library provides built-in functionality for user management, including registration, authorization, credential validation, password encryption, and data storage in a database [50]. It also supports configuration of password requirements, account blocking, access conditions, and more. Additionally, ASP.NET Identity is compatible with Entity Framework, enabling the generation of database schema for user management using the code-first approach.

ASP.NET Identity simplifies and accelerates user management development. Services are made available to the application through dependency injection and can be configured as needed. The following image demonstrates the configuration of ASP.NET Identity, where password validation parameters are set, a unique email requirement is enforced, and email confirmation is turned off. The configured ASP.NET Identity is also provided with an Entity Framework DB context to handle persistence operations.

```
builder.Services.AddIdentity<AppUser, IdentityRole>(options =>
{
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;

    options.User.RequireUniqueEmail = true;
    options.User.AllowedUserNameCharacters += " ";

    options.SignIn.RequireConfirmedEmail = false;
})
.AddEntityFrameworkStores<OneDayTripDbContext>();
```

Figure 3.26. ASP.NET Identity configuration.

After configuring ASP.NET Identity, I developed user registration using the *UserManager* class, which manages user accounts and performs various operations, including creating or removing user accounts, modifying passwords, and assigning or removing users from roles.

For sign-in functionality, I used the *SignInManager* class. After successful authorization by Identity, I create a JWT, which is sent as a response. The frontend then sends this token with each request that must be authorized.

3.5 Project setup and configuration

Web applications are designed to be deployed and hosted so they can be accessed by users over the internet. Due to time limitations, the application has not been deployed yet and has only been tested locally. This section describes how to configure the application to run on a local computer.

3.5.1 Running the frontend project

To run the frontend project, the following steps must be completed:

1. Install Node.js version 10.5.2 or higher [51].
2. Install Angular CLI version 17.2.1 by running the command `npm install -g @angular/cli`. Refer to the Angular setup guide for more details [52].
3. Install the project packages by executing the command `npm i` in the root folder of the Angular project, which is the `OneDayTrip/src/client` folder.
4. Obtain a Google Maps API key according to instructions [53].
5. Insert your Google Maps API key into the `googleApiKey` field in the `environment.development.ts` file located in the `OneDayTrip/src/client/src/environments` folder.
6. Execute the `ng serve` command in the root of the client application folder (`OneDayTrip/src/client`) to build and run the frontend project.

3.5.2 Running the backend project

Before running the backend project locally, the database must be set up according to the following steps.

1. Install PostgreSQL and pgAdmin 4 according to the instructions [54]. During the pgAdmin installation, ensure you check the PostGIS checkbox to install the PostGIS extension.
2. Verify that PostGIS is installed. If it is not installed, follow the instructions to install it [40].

After setting up the database, configure the backend project with the following steps:

1. Install Visual Studio with .NET 8 included [55].
2. Set the database connection string in the `appsettings.Development.json` file located in the `OneDayTrip/src/server/OneDayTrip` folder, within the `ConnectionString` section under the `OneDayTrip` field.
3. In the same file, specify the path to the folder where the images will be uploaded in the `TripPlanImagesDirectory` field.
4. Open the project in Visual Studio by double-clicking the `OneDayTrip.sln` file in the root of the backend folder (`OneDayTrip/src/server/OneDayTrip`).
5. Open the Package Manager Console by clicking on the search button and finding it.
6. In the Package Manager Console, execute the command: `Update-Database -Project OneDayTrip.DataAccess -StartupProject OneDayTrip` to run database migrations and build the relevant database schema.
7. Run the project by clicking on Debug and selecting Start Without Debugging.

By following these steps, you will be able to configure and run the application on your local computer.

Chapter 4

Testing

The purpose of testing in software development is to identify and fix bugs, ensure the software meets requirements, and verify that it functions correctly under various conditions. This chapter will describe the project testing, which includes static testing, manual testing, unit testing, usability testing, and acceptance testing.

4.1 Static testing

Static testing is a method of software testing that analyzes code and identifies issues without executing the program [56]. Unlike dynamic testing, which focuses on the behavior of the code during runtime, static testing examines the code itself. It involves reviewing the source code, analyzing its structure, and identifying potential defects, vulnerabilities, and compliance issues.

For static testing of the project, I used ReSharper for Visual Studio [57]. It identifies various types of problems, including unused variables, unreachable code, potential bugs, logical errors, incorrect or missing nullability annotations and more. ReSharper ensures adherence to coding standards, flags inefficient code patterns that could impact performance and provides recommendations for code refactoring, helping to maintain clean, efficient, and maintainable code.

4.2 Manual testing

Manual testing is performed by the tester who carries out all the actions on the tested application manually, step by step and indicates whether a particular step was accomplished successfully or whether it failed [58]. It is especially useful in the initial phase of software development, when the software and its user interface are not stable enough, and beginning the automation does not make sense.

The list of test cases for manual testing was created based on functional requirements. These requirements were grouped by functionality modules and ordered to allow for sequential execution. The list of test cases performed by the testers is provided in Appendix C.

For manual testing, a group of five people was assembled. Each tester was provided with an identical list of test cases to complete. Testing was conducted by different individuals at various stages of project development. Upon completion of the test cases, feedback was collected and identified issues were resolved.

The collected information helped to identify and fix bugs. Additionally, it provided valuable feedback for the project, which affected the development process.

4.3 Unit testing

Unit Testing focuses on testing individual components or units of the code to ensure each part functions correctly. Individual functions or class methods, classes themselves,

class interactions, small libraries, or parts of an application may be tested [59]. These tests are automated and help detect errors early in the development process.

Several unit tests were created for the project where appropriate. These unit tests ensure the consistency of trip plans and their associated images. Given that images are saved on disk while trip plans are stored in a database, it is essential to handle cases where saving images or trip plans fails. Trip plans and their images should be saved together, ensuring that if one fails, neither is saved. Since storing images on disk is under consideration and may be improved by utilizing a CDN in the future, only the most critical cases were tested. Other unit tests were deemed unnecessary because they would involve testing trivial database operations.

4.4 Usability testing and comparison with TripAdvisor

Usability testing evaluates whether the end user understands how to use the product and how much they enjoy using it [59]. This process involves observing users as they complete specific tasks, identifying any usability issues, and gathering qualitative and quantitative data to improve the overall user experience.

To determine if the application offers any advantages over TripAdvisor, a group of users was asked to find the same functionality on both this bachelor project application and the TripAdvisor application. They were instructed to locate a trip plan containing a list of places to visit, their descriptions, and a map showing the locations. The results indicated that all users found this functionality more quickly on the bachelor project application than on TripAdvisor, confirming the project's value and motivation.

Tester	Bachelor project application	TripAdvisor
1	46 seconds	2 minutes 30 seconds
2	15 seconds	1 minute 10 seconds
3	10 seconds	1 minute 40 seconds
4	52 seconds	4 minutes
5	30 seconds	2 minutes 40 seconds

Table 4.1. Time spent to find trip plan on bachelor project application and the TripAdvisor application.

Testers were also asked to provide their usability feedback on test cases from Appendix C. Additionally, half of the users were asked to evaluate the same test cases using a mobile application simulator. Feedback was collected and a list of suggestions for improvements was compiled:

- Ensure button colors are consistent. The delete buttons should be red. Currently, the delete trip plan button on the Bookmarks page is green. Similarly, the cancel button should be grey, not green. The text in the delete and edit buttons may be replaced with icons.
- Increase the margins between information blocks on the trip plan page.
- Add a “view trip plan” button to the trip plan cards, such as those presented on the search results page.
- Change the color of stars indicating the rating to yellow. Currently, they have the primary color, which is green.

Based on general user feedback, the application was rated as sufficiently user-friendly and intuitive. The testing was successful, indicating that the application meets the necessary usability standards.

4.5 Acceptance testing

Acceptance testing is conducted to verify that the application meets the defined business requirements and the users' needs [59]. This type of testing ensures that the system behaves as expected from an end-user perspective.

Informal acceptance testing was conducted with a selected group of users who tested the essential and most important functionalities of the application described in test cases in Appendix C. During this testing phase, the users provided a list of suggested improvements:

- Improve the primary image selection process on the create and edit trip plan pages. Currently, each uploaded image has a star button in the corner. When clicked, the image is marked as primary and will be presented on the search results page. However, this star button is not intuitive, even with the existing hint, and can be mistaken for a rating button, which is usually represented by stars. Most testers were confused about this button. The suggested solution is to add a new section to the create and edit trip plan pages where users can select a primary image from all uploaded images. Additionally, the primary image functionality could be implemented for the trip plan block (POI) of the trip plan.
- Add support for users to post multiple comments on a single trip plan. Implement thread functionality to enable users to engage in discussions and answer each other's questions.
- Replace Google Maps tags with manually added tags or support editing them. Frequently, place type tags retrieved from the Google Maps API are not human-readable and can be confusing. Ensure that place type tags are more user-friendly.
- Add text formatting support for trip plan and trip plan block descriptions and comments. This will enhance the text visual appeal.
- Show users' private trip plans in search results if they are the author.
- When redirecting from POI bookmark to the trip plan page, ensure that the trip plan page scrolls to the specific POI rather than starting from the beginning of the page.

The feedback received was invaluable and has been taken into account. This testing was a crucial part of the software development lifecycle, as it confirmed that the application meets the needs and expectations of its end-users.

Chapter 5

Conclusion

This section outlines the experience of designing and developing this bachelor project, highlighting key aspects of the process. The project's goals included researching publicly available travel data sources, proposing architecture, and implementing a web application structured around blog functionality, enabling travelers to compose and share posts (trip plans) detailing their journeys.

The work began with an analysis phase. Application requirements were defined based on an analysis of existing solutions and the preferences of the target audience. A Use Cases diagram was created to capture system requirements and visually represent possible interactions between the system and external entities. A business domain model was developed to structure the data and visualize the entities, representing the conceptual view of the system. Additionally, a low fidelity GUI prototype was designed.

During the architecture proposal phase, it was decided to adopt a client-server architecture due to the application's small size and the low complexity of the business logic. To enhance the separation of concerns, the project adopted a Three-Tier architecture comprising the Presentation Layer, Business Logic Layer, and Data Access Layer.

C# and the .NET platform were selected as the backend technologies, primarily due to personal experience in developing applications with these tools. For the frontend implementation, the Angular framework was chosen, again due to prior experience with this technology.

Due to personal interest and a desire to deepen my understanding, an analysis and comparison of popular database solutions were conducted. Based on personal experience and industry statistics, Oracle and Microsoft SQL Server are among the most popular database engines for enterprise projects. While these engines are highly reliable, there is a noticeable trend of migrating projects to PostgreSQL, which is frequently discussed at various conferences. As a result of this investigation, it became clear that the primary reason for this trend is the cost of the license, as PostgreSQL is available free of charge.

The decisive factor in choosing the PostgreSQL database for the project is its ability to be extended with the PostGIS extension, which offers robust geospatial data management and a variety of functions to enable GIS processing within the database itself.

One of the most critical aspects of the project was selecting the appropriate geographical data provider. The most suitable options were Google Maps and Mapy.cz. Considering the popularity of the service, the richness of its functionality, and the potential for future project growth, Google Maps was chosen as the primary geographical data provider.

During the implementation phase, a component diagram was created to visually represent the project's architecture. Additionally, a sequence diagram was developed to illustrate the primary functionality of creating a trip plan, which utilizes both the Google Maps Platform and the backend.

The frontend implementation started with the selection of a component library. Initially, I opted to utilize the Angular Material component library due to its simplicity and official endorsement by the Angular Team at Google. However, in the middle of the project implementation, I encountered limitations within the library that were critical for the application's needs. The library lacked essential components such as an image carousel and a file upload button. After careful consideration, I opted for the PrimeNG components library and transitioned the project to it. Working with PrimeNG proved to be an exceptional experience, thanks to its comprehensive suite of components and extensive configurability. This experience taught me the importance of thoroughly reviewing a library's components before integration. I also realized that such decisions should be made during the analysis phase rather than the implementation phase.

All interactions with the Google Maps Platform within the project take place on the frontend. This includes embedded map rendering, autocomplete place search functionality, and place details requests. Research was conducted to compare different versions of Google Maps APIs and their billing structures. Google's billing is determined by the data fields queried, making it essential to request only the necessary data to avoid unnecessary costs. Another cost-saving factor for requests is the utilization of sessions.

The .NET backend solution was structured as a Three-Tier application, consisting of three distinct projects, each representing layers of the Three-Tier architecture, which made it well-encapsulated. The Data Access Layer (DAL) project consists of Entities and Repositories, facilitating access to the database and file system. This project utilized the Entity Framework ORM and follows a code-first approach, wherein the Entity Framework generates the database schema based on the Entities and DB Context configuration. The Business Logic Layer project contains models and services, utilizing methods from the DAL. The Presentation Layer project functions as the main executable, acting as a monolithic project that connects all components and manages dependency injection. This layer includes Controllers and DTOs, providing Swagger documentation in accordance with the OpenAPI specification.

Furthermore, the backend implementation chapter describes the details of implementing the dependency injection software design pattern, along with experiences of utilizing PostGIS using the NetTopologySuite library in conjunction with the Entity Framework ORM. It also describes solutions for storing uploaded files on the backend, data mapping, security, and the usage of the Unit of Work design pattern. Additionally, it describes the experience of generating TypeScript contracts from C# DTOs with the assistance of the Reinforced.Typings library.

The final presents an overview of the project's testing phase, which was carried out consistently and included a range of methods such as static, manual, unit, usability, and acceptance testing. The feedback obtained was instrumental in identifying bugs and improving the project. The testing was a crucial part of the software development lifecycle, as it confirmed that the application meets the needs and expectations of its end-users and possesses potential for further growth and improvement.

Considering all these factors, it can be concluded that the bachelor project was successfully completed, and the project goals are considered to be achieved.

5.1 Further prospects of the project

Based on user suggestions gathered during testing and my personal experience and knowledge about the project, I have compiled a list of factors that may enhance the project to production quality:

- Deploy the project. Consider using cloud services such as Azure or Amazon. Compare their free tiers to determine which is more beneficial.
- Investigate using a Content Delivery Network (CDN) for POI image storage. This may provide better data consistency, distribution, and availability.
- Improve logging to make it more meaningful.
- Assess whether using containers for this project could be beneficial.
- Debug database queries generated by Entity Framework to improve performance.
- Add caching for frequently used data such as POI types.
- Implement administrative functionality such as blocking users and hiding trip plans from the public. Consider using AI for content validation, censorship, and ensuring trip plans do not contain inappropriate or harmful content.
- Enhance UI theme customization to give the project a polished and professional appearance.

I plan to continue working on the project as it is an excellent opportunity to expand my knowledge and gain new experience.

Bibliography

- [1] Tripadvisor. *[online]*.
<https://www.tripadvisor.com>. c2023. Accessed: 2023-12-16.
- [2] Wanderlog. *travel itinerary, vacation & road trip planner [online]*.
<https://wanderlog.com>. c2023. Accessed: 2023-10-15.
- [3] Plan Your Trip. *[online]*.
<https://planyourtrip.com>. c2023. Accessed: 2023-12-16.
- [4] RoutePerfect. *Plan & Book A Perfect Trip With Our Itinerary Planner [online]*.
<https://www.routeperfect.com>. c2014-2023. Accessed: 2023-12-16.
- [5] Coara. *Business Requirements vs Functional Requirements [online]*.
<https://coara.co/blog/business-requirements-vs-functional-requirements>. 2020. Accessed: 2023-12-16.
- [6] Sparx Systems. *Use Case Diagram - UML 2 Tutorial [online]*.
<https://sparxsystems.com/resources/tutorials/uml2/use-case-diagram.html>. c2000-2023. Accessed: 2023-12-17.
- [7] Santosh Kumar. A Review on Client-Server based applications and research opportunity. *International Journal of Recent Scientific Research*. 2019, 10 (7), 33857–3386.
- [8] Appsierra. *Software Architecture: N Tier, 3 Tier, 1 Tier, 2 Tier Architecture [online]*.
<https://www.appsierra.com/blog/tiers-in-software-architecture>. 2023. Accessed: 2023-12-17.
- [9] Guru99. *N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE [online]*.
<https://www.guru99.com/n-tier-architecture-system-concepts-tips.html>. 2023. Accessed: 2023-12-17.
- [10] Microsoft. *C# | Modern, open-source programming language for .NET [online]*.
<https://dotnet.microsoft.com/en-us/languages/csharp>. c2023. Accessed: 2023-12-17.
- [11] Stack Overflow Insights. *Stack Overflow Developer Survey 2023 [online]*.
<https://survey.stackoverflow.co/2023/#technology-admired-and-desired>. 2023. Accessed: 2023-12-17.
- [12] Vadym Rudenko. *Bachelor thesis. Design and implementation of a cryptocurrency market notification application [online]*.
<https://dspace.cvut.cz/handle/10467/108764>. c2023. Accessed: 2023-11-12.
- [13] Microsoft Learn. *Lifecycle FAQ - .NET and .NET Core [online]*.
<https://learn.microsoft.com/en-us/lifecycle/faq/dotnet-core>. c2023. Accessed: 2023-12-17.
- [14] Marino Posadas. *Mastering C# and .NET Framework*. Packt Publishing Ltd, 2016.

- [15] Andrew W Troelsen, and Philip Japikse. *Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming*. Springer, 2022.
- [16] Shyam Seshadri. *Angular: Up and running: Learning angular, step by step.* ” O’Reilly Media, Inc.”, 2018.
- [17] Aristeidis Bampakos, and Pablo Deeleman. *Learning Angular: A no-nonsense guide to building web applications with Angular 15*. Packt Publishing Ltd, 2023.
- [18] DB-Engines - Knowledge Base of Relational, and NoSQL Database Management Systems. *Historical trend of relational DBMS popularity [online]*. https://db-engines.com/en/ranking_trend/relational+dbms. c2023. Accessed: 2023-12-17.
- [19] IBM. *PostgreSQL vs. MySQL: What’s the Difference? [online]*. <https://www.ibm.com/blog/postgresql-vs-mysql-whats-the-difference>. 2021. Accessed: 2023-12-17.
- [20] Amazon Web Services (AWS). *PostgreSQL vs MySQL - Difference Between Relational Database Management Systems (RDBMS) [online]*. <https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql>. c2023. Accessed: 2023-12-17.
- [21] Spatial Post. *11 Best Geospatial Database Systems: An In-Depth Comparison [online]*. <https://www.spatialpost.com/best-geospatial-database-systems>. 2023. Accessed: 2023-12-17.
- [22] Google Maps Platform. *Platform Pricing & API Costs - Google Maps Platform [online]*. <https://mapsplatform.google.com/pricing>. c2023. Accessed: 2023-12-17.
- [23] Developer Mapy.cz. *Pricing - Developer Mapy.cz [online]*. <https://developer.mapy.cz/en/pricing>. c1996–2023. Accessed: 2023-12-17.
- [24] Google Trends. *Google Maps, Mapy.cz - Explore - Google Trends [online]*. <https://trends.google.com/trends/explore?q=Google%20Maps,Mapy.cz&hl=en-US>. c2023. Accessed: 2023-12-17.
- [25] Angular. *Angular Material [online]*. <https://material.angular.io>. 2024. Accessed: 2024-05-12.
- [26] Angular. *Google Maps component [online]*. <https://github.com/angular/components/blob/main/src/google-maps/README.md>. 2023. Accessed: 2024-05-12.
- [27] PrimeNG. *Angular UI Component Library [online]*. <https://primeng.org>. 2024. Accessed: 2024-05-12.
- [28] AGM. *Angular Google Maps [online]*. <https://angular-maps.com>. 2018. Accessed: 2024-05-12.
- [29] Google Maps Platform Documentation. *Maps JavaScript API [online]*. <https://developers.google.com/maps/documentation/javascript>. 2024. Accessed: 2024-05-12.
- [30] Google Maps Platform Documentation. *Maps JavaScript API Usage and Billing [online]*. <https://developers.google.com/maps/documentation/javascript/usage-and-billing>. 2024. Accessed: 2024-05-12.

- [31] Google Maps Platform Documentation. *The Maps Embed API overview [online]*. <https://developers.google.com/maps/documentation/embed/get-started>. 2024. Accessed: 2024-05-12.
- [32] Google Maps Platform Documentation. *Maps Embed API Usage and Billing [online]*. <https://developers.google.com/maps/documentation/embed/usage-and-billing>. 2024. Accessed: 2024-05-12.
- [33] Google Maps Platform Documentation. *Places API [online]*. <https://developers.google.com/maps/documentation/places/web-service>. 2024. Accessed: 2024-05-12.
- [34] Google Maps Platform Documentation. *Choose your API version - Places API [online]*. <https://developers.google.com/maps/documentation/places/web-service/choose-api>. 2024. Accessed: 2024-05-12.
- [35] Google Maps Platform Documentation. *Session tokens - Places API [online]*. <https://developers.google.com/maps/documentation/places/web-service/place-session-tokens>. 2024. Accessed: 2024-05-12.
- [36] Google Maps Platform Documentation. *Autocomplete (New) and session pricing [online]*. <https://developers.google.com/maps/documentation/places/web-service/session-pricing>. 2024. Accessed: 2024-05-12.
- [37] Microsoft Learn. *.NET dependency injection [online]*. <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>. 2024. Accessed: 2024-05-12.
- [38] Microsoft Learn. *Dependency injection guidelines [online]*. <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines>. 2024. Accessed: 2024-05-12.
- [39] PostGIS. *About PostGIS [online]*. <https://postgis.net>. 2023. Accessed: 2024-05-12.
- [40] PostGIS. *Getting Started [online]*. https://postgis.net/documentation/getting_started. 2023. Accessed: 2024-05-12.
- [41] PostGIS. *Geography [online]*. <https://postgis.net/workshops/postgis-intro/geography.html>. 2023. Accessed: 2024-05-12.
- [42] Cockroach Lab. *SRID 4326 - longitude and latitude [online]*. <https://www.cockroachlabs.com/docs/stable/srid-4326>. 2024. Accessed: 2024-05-12.
- [43] Npgsql - .NET Access to PostgreSQL. *Spatial Mapping with NetTopologySuite [online]*. <https://www.npgsql.org/efcore/mapping/nts.html?tabs=without-datasource>. 2023. Accessed: 2024-05-12.
- [44] Stack Overflow. *Storing Images in DB - Yea or Nay? [online]*. <https://stackoverflow.com/questions/3748/storing-images-in-db-yea-or-nay>. 2008. Accessed: 2024-05-12.

- [45] Amazon Web Services (AWS). *What is a CDN (Content Delivery Network)? [online]*.
<https://aws.amazon.com/what-is/cdn/>. 2024. Accessed: 2024-05-12.
- [46] AutoMapper. *[online]*.
<https://docs.automapper.org>. 2024. Accessed: 2024-05-12.
- [47] Reinforced.Typings. *Source code and documentation [online]*.
<https://github.com/reinforced/Reinforced.Typings>. 2024.
- [48] Alex Klaus. *6+ ways to marry C# with TypeScript [online]*.
<https://alex-klaus.com/marry-csharp-typescript>. 2020. Accessed: 2024-05-12.
- [49] Medium. *Implementing the Unit of Work Pattern in Clean architecture with .NET Core [online]*.
<https://medium.com/@edin.sahbaz/implementing-the-unit-of-work-pattern-in-clean-architecture-with-net-core-53efb7f9d4d>. 2024. Accessed: 2024-05-12.
- [50] Microsoft Learn. *Introduction to Identity on ASP.NET Core [online]*.
<https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio>. 2024. Accessed: 2024-05-12.
- [51] Node.js. *Download Node.js [online]*.
<https://nodejs.org/en/download/prebuilt-installer>. 2024. Accessed: 2024-05-12.
- [52] Angular. *Setting up the local environment and workspace [online]*.
<https://angular.io/guide/setup-local>. 2022. Accessed: 2024-05-12.
- [53] Google Maps Platform Documentation. *Use API Keys [online]*.
<https://developers.google.com/maps/documentation/embed/get-api-key>. 2024. Accessed: 2024-05-12.
- [54] W3Schools. *Install PostgreSQL [online]*.
https://www.w3schools.com/postgresql/postgresql_install.php. Accessed: 2024-05-12.
- [55] Microsoft. *Visual Studio [online]*.
<https://visualstudio.microsoft.com>. 2024. Accessed: 2024-05-12.
- [56] Medium. *Understanding Static Testing and Static Code Analysis Tools — SonarLint [online]*.
<https://recepemul.medium.com/understanding-static-testing-and-static-code-analysis-tools-sonarlint-23359a8756f3>. 2023. Accessed: 2024-05-12.
- [57] JetBrains ReSharper. *Code analysis [online]*.
https://www.jetbrains.com/help/resharper/Code_Analysis__Index.html. 2024. Accessed: 2024-05-12.
- [58] SmartBear Support. *Manual Testing [online]*.
<https://support.smartbear.com/testcomplete/docs/testing-with/deprecated/manual/index.html>. 2024. Accessed: 2024-05-12.
- [59] Svyatoslav Kulikov. *Software testing. 3rd edition [online]*. 2024. Accessed: 2024-05-12.

Appendix A

Low fidelity prototype of GUI

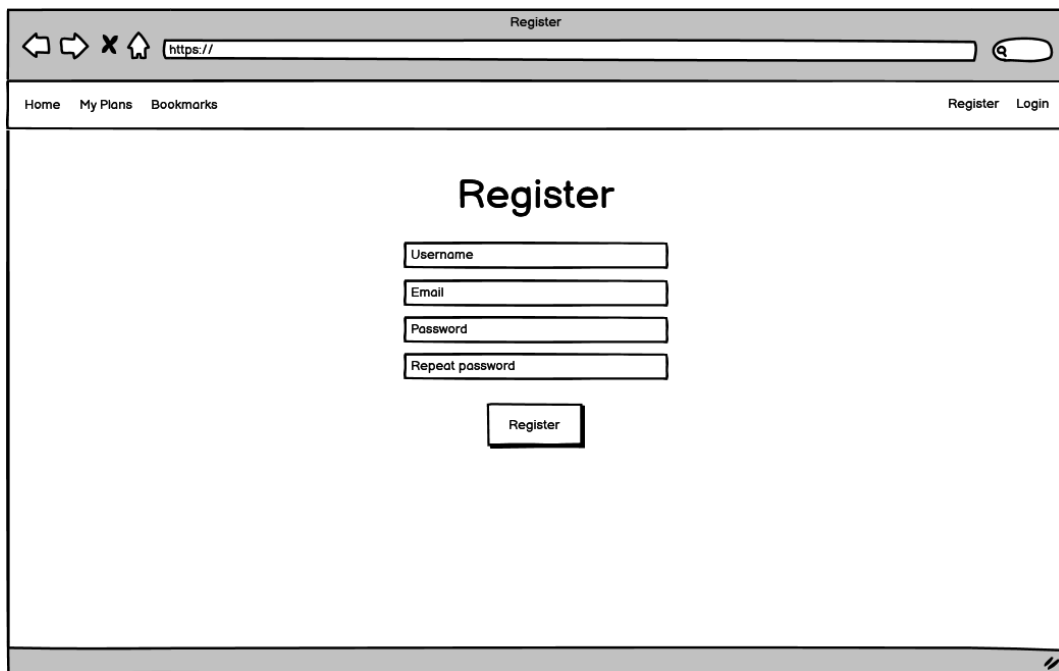


Figure A.1. Desktop registration page prototype.

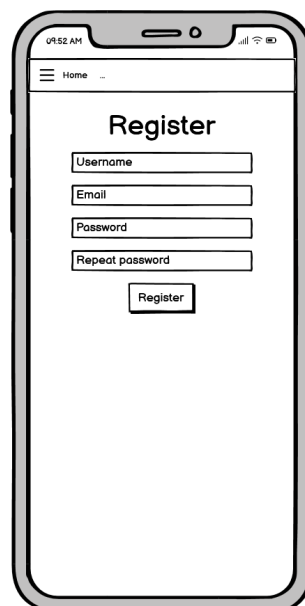


Figure A.2. Mobile registration page prototype.

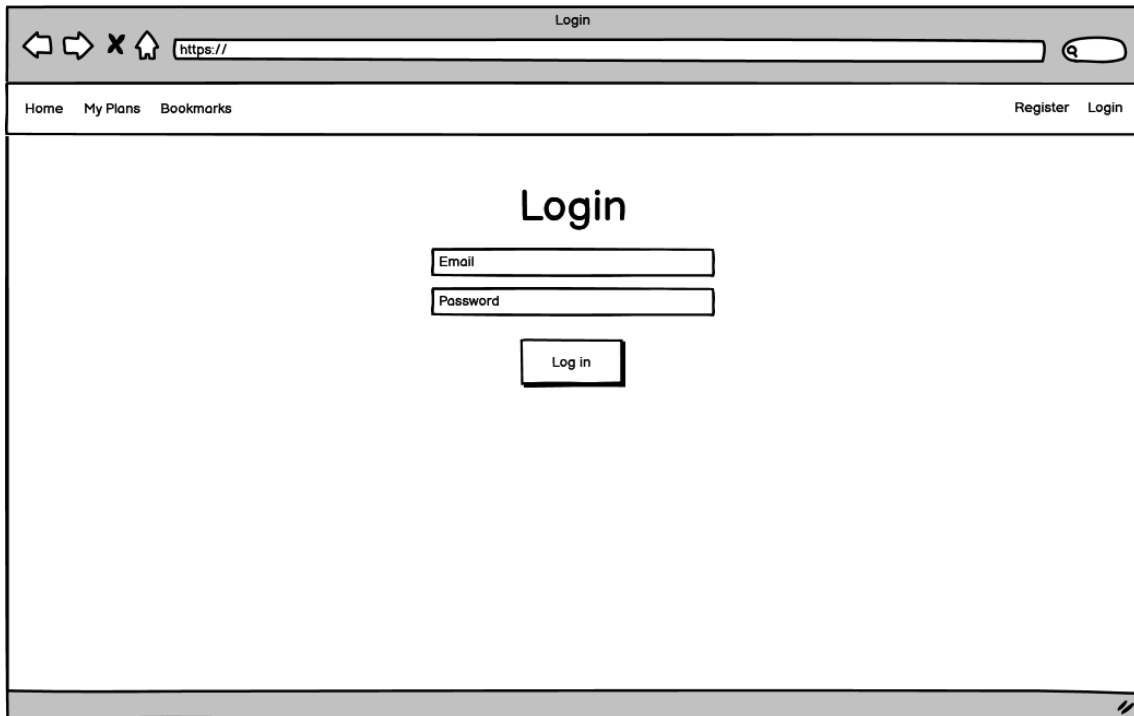


Figure A.3. Desktop login page prototype.

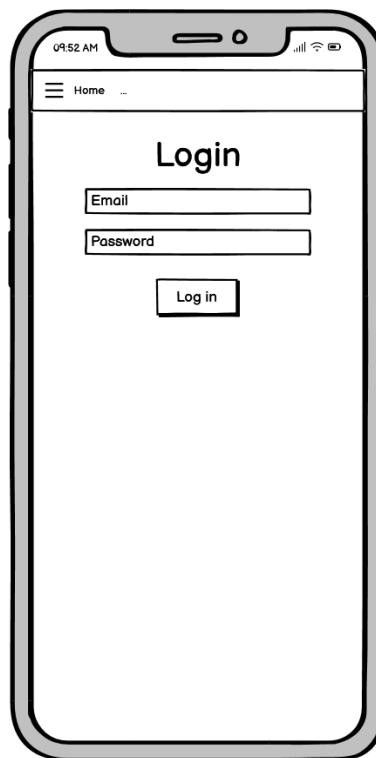


Figure A.4. Mobile login page prototype.

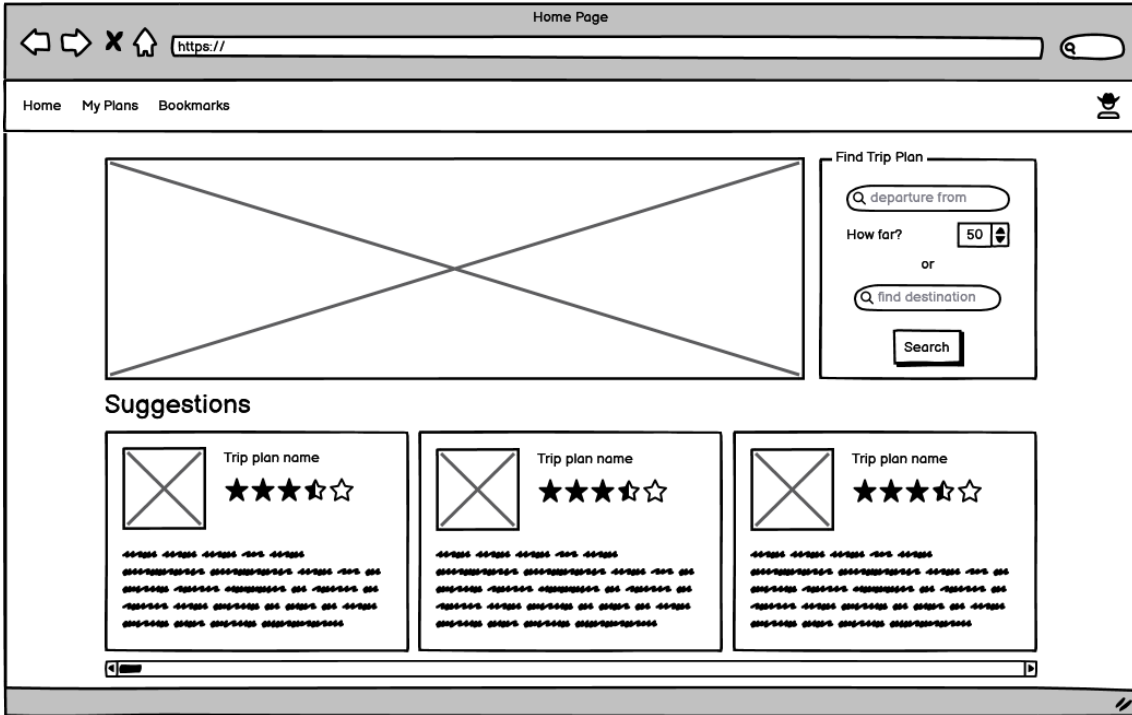


Figure A.5. Desktop home page prototype.



Figure A.6. Mobile home page prototype.

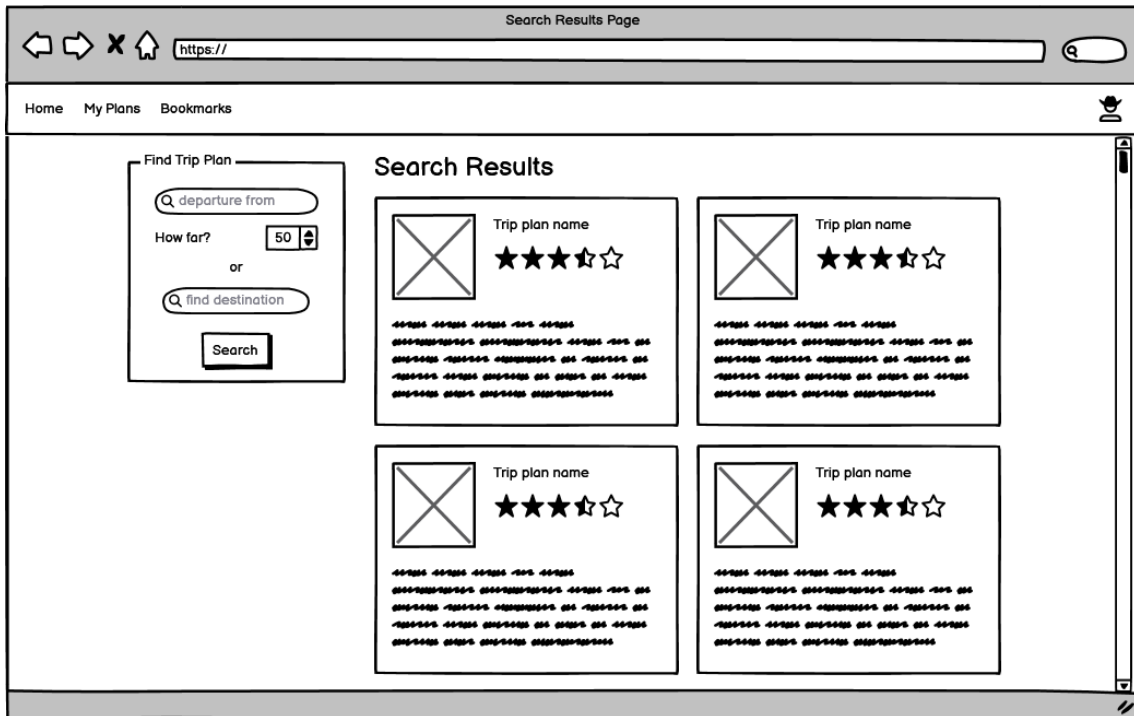


Figure A.7. Desktop search results page prototype.

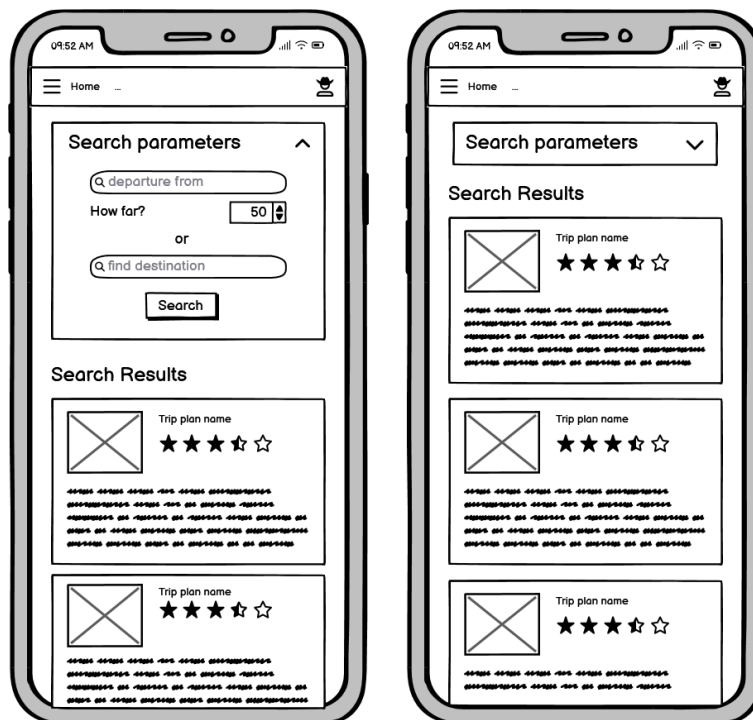


Figure A.8. Mobile search results page prototype.

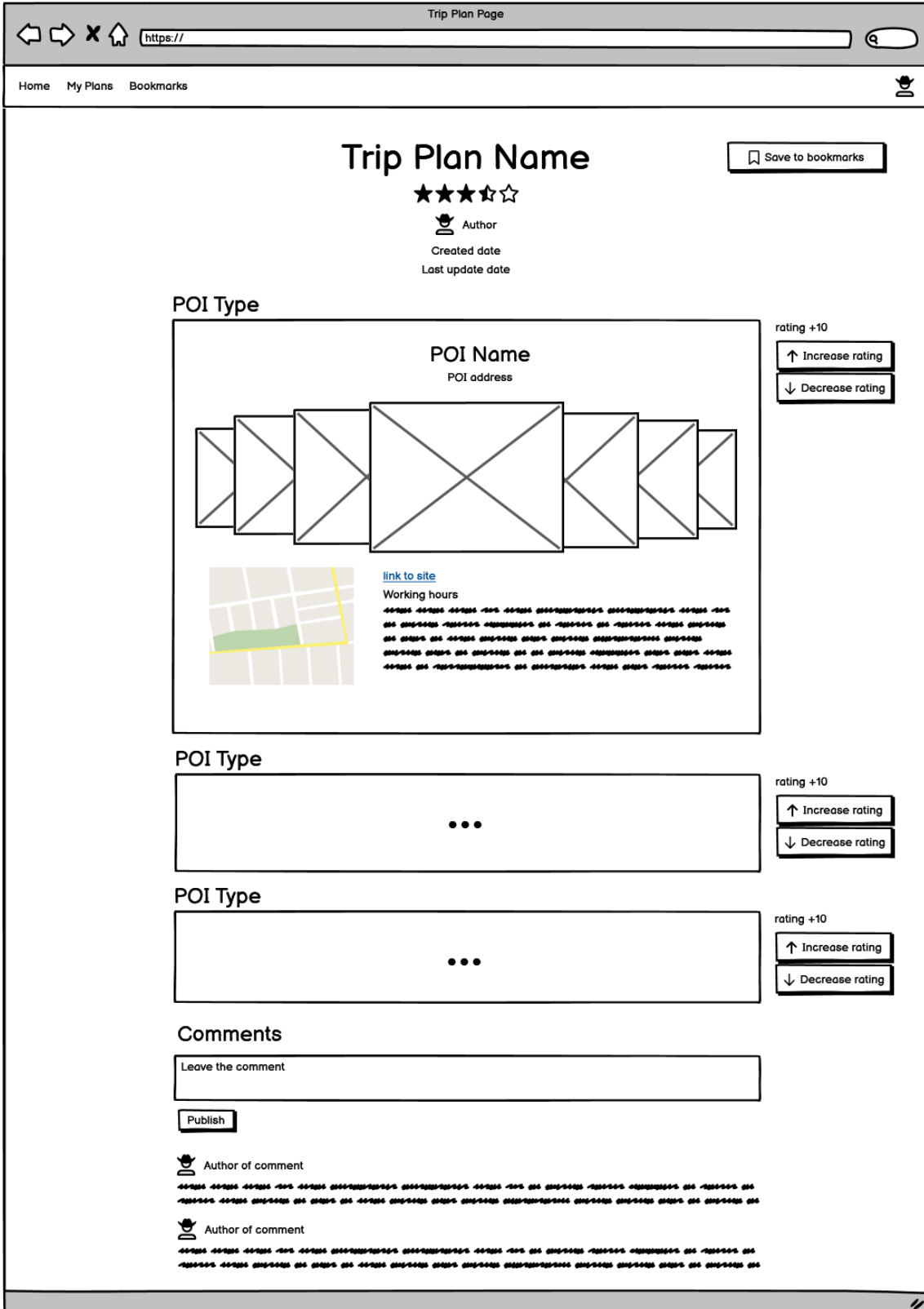


Figure A.9. Desktop trip plan page prototype.



Figure A.10. Mobile trip plan page prototype.

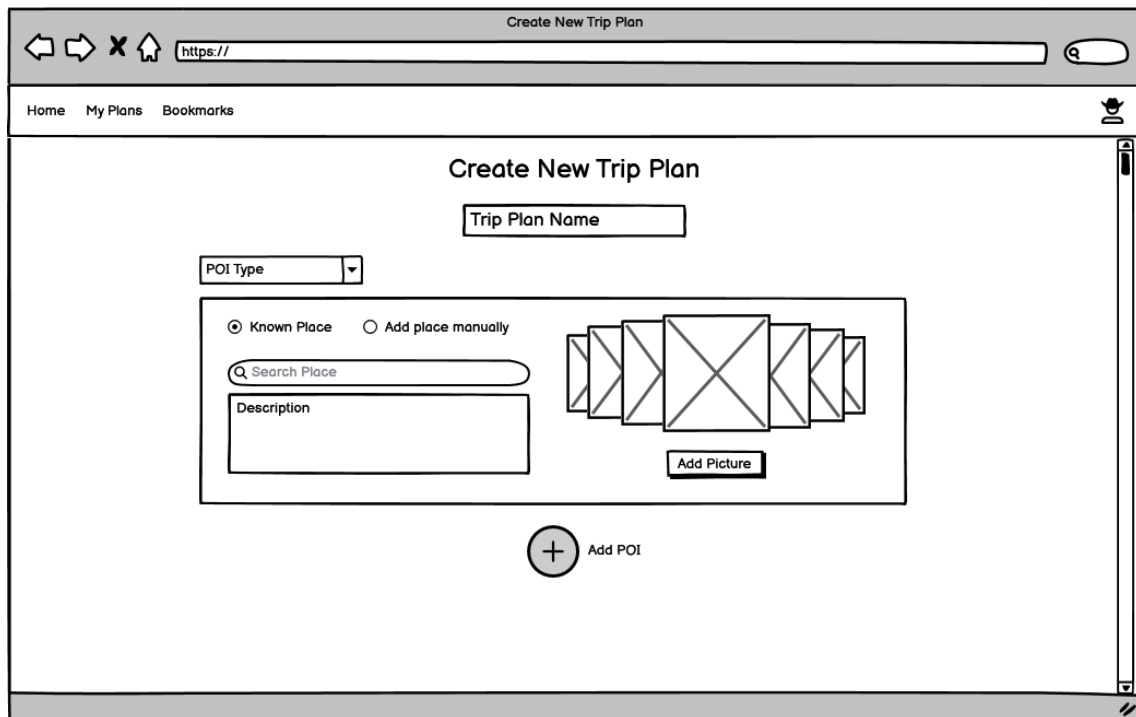


Figure A.11. Desktop create new trip plan page prototype.

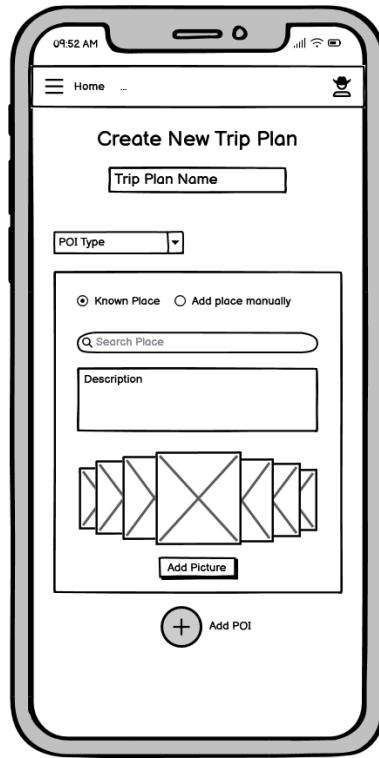


Figure A.12. Mobile create new trip plan page prototype.

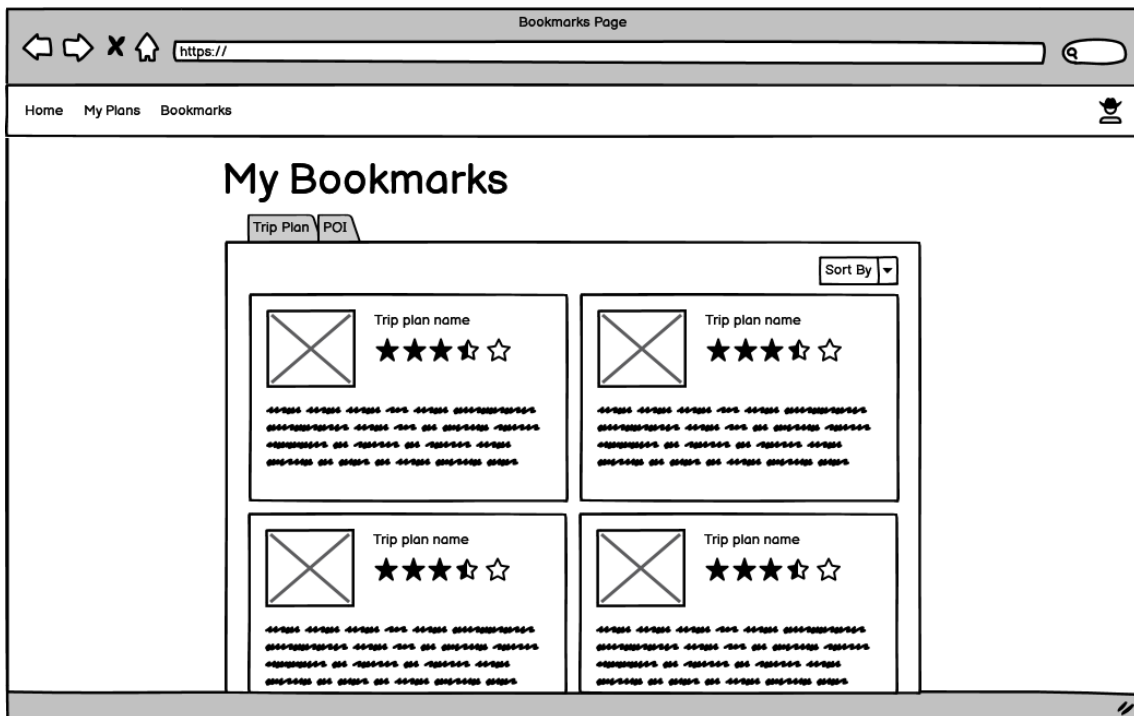


Figure A.13. Desktop user's bookmarks page prototype.

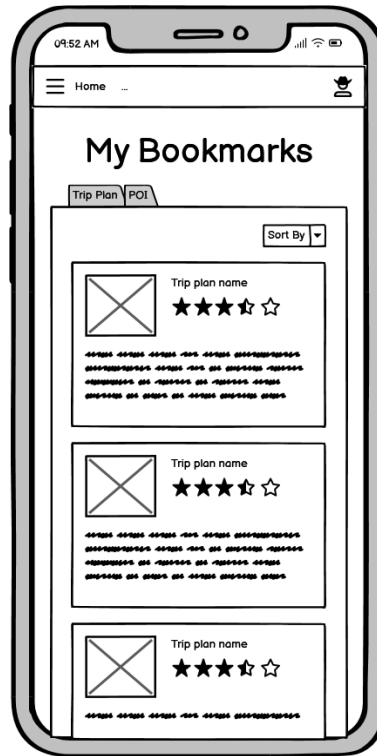


Figure A.14. Mobile user's bookmarks page prototype.

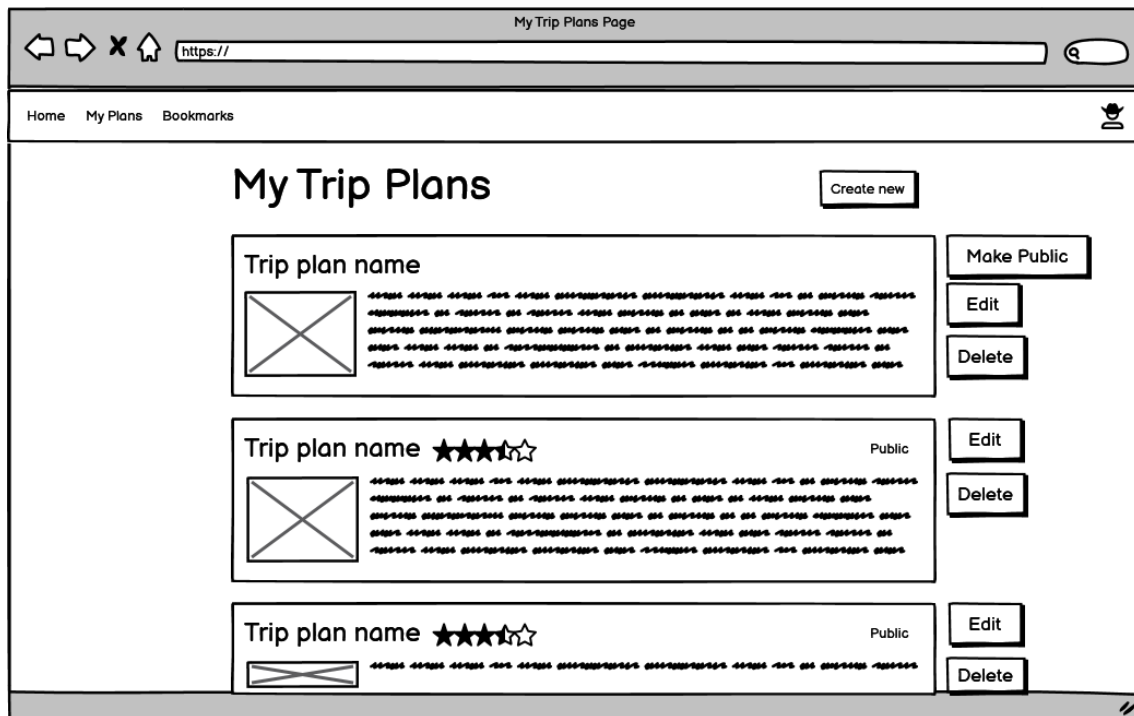


Figure A.15. Desktop user's trip plans page prototype.

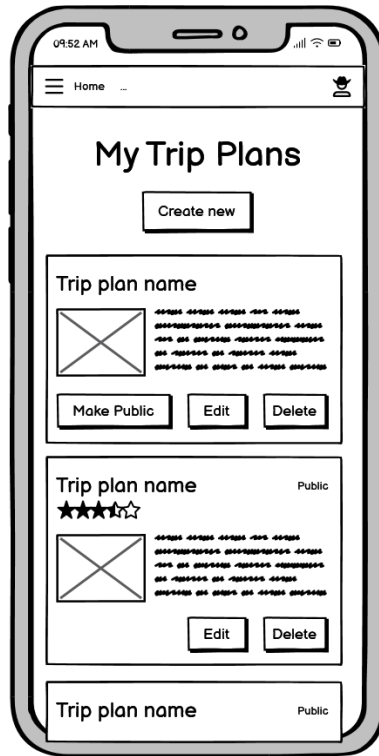


Figure A.16. Mobile user's trip plans page prototype.

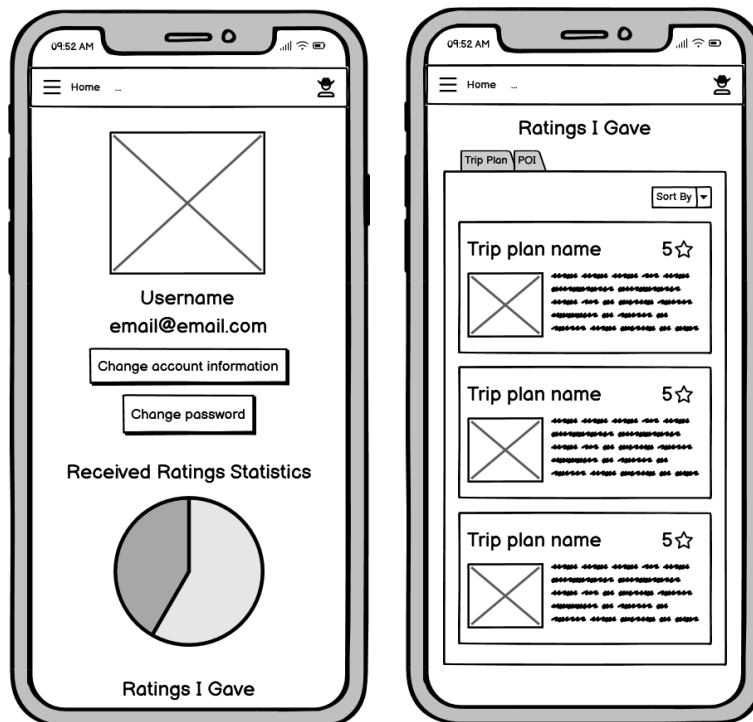


Figure A.17. Mobile user's account page prototype.

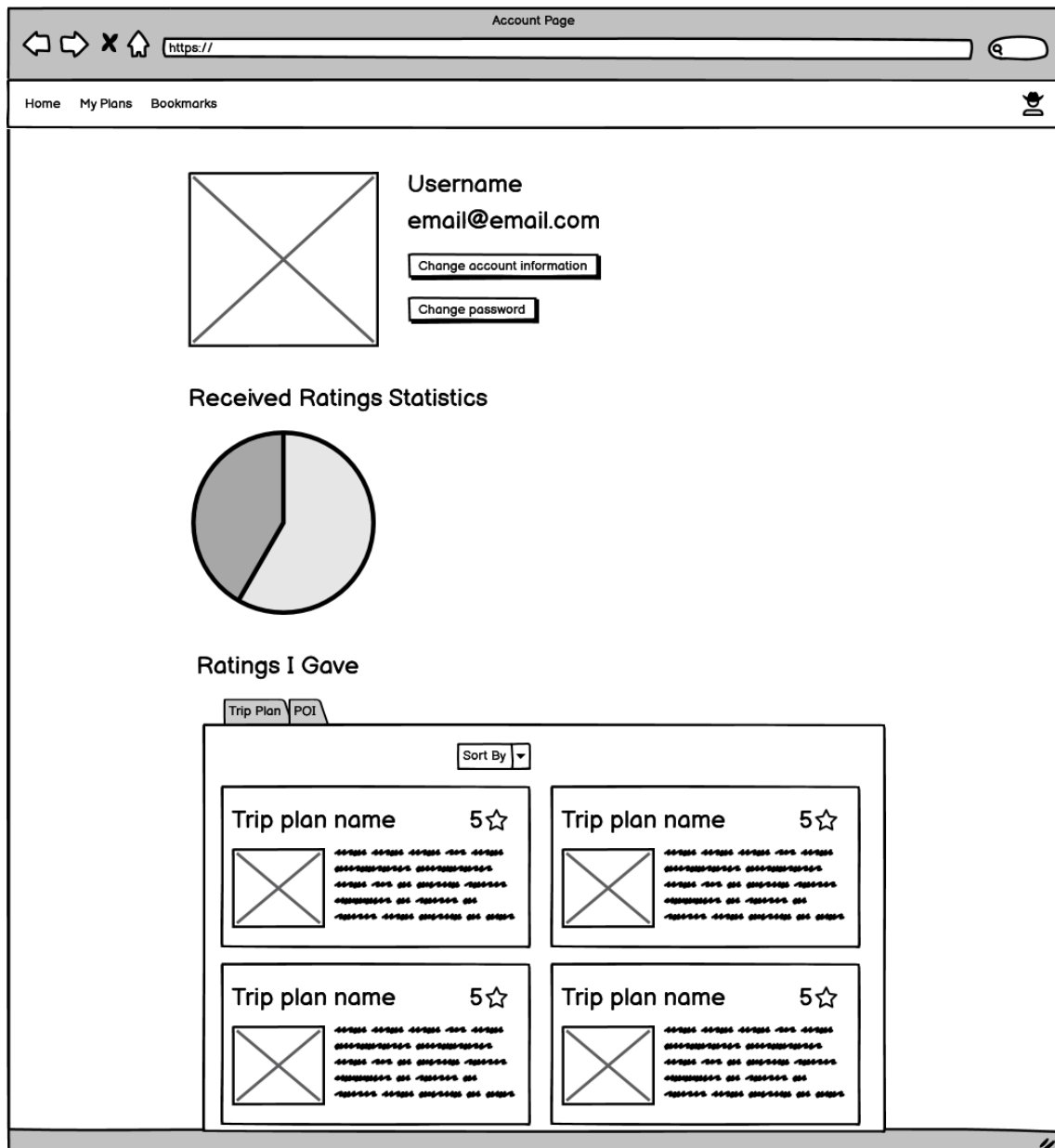


Figure A.18. Desktop user's account page prototype.

Appendix B

GUI

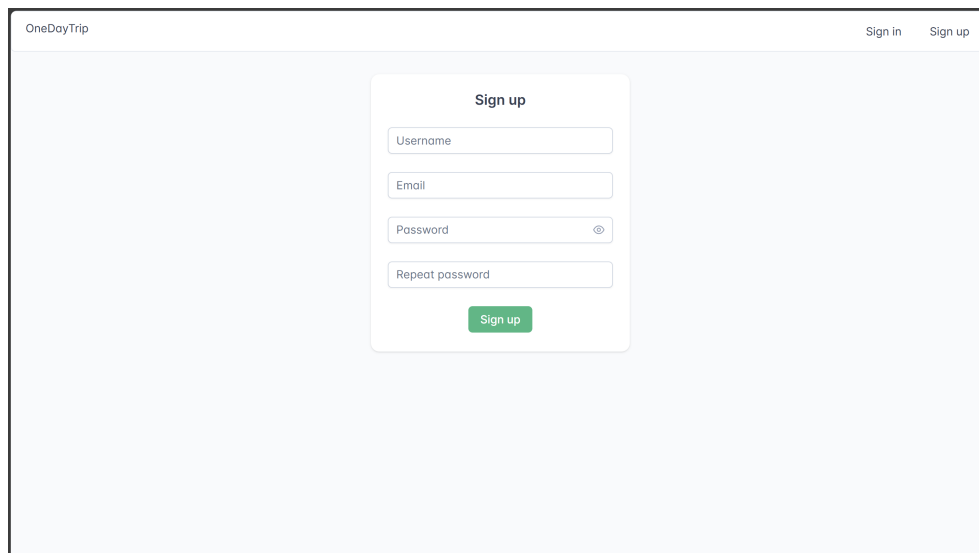


Figure B.19. Desktop registration page.

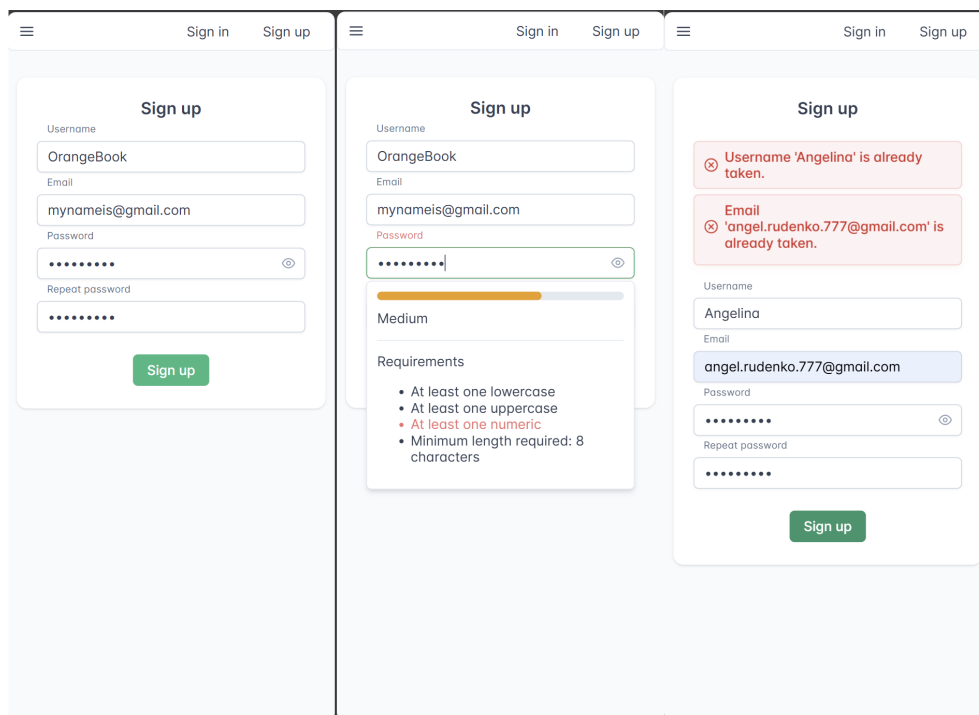


Figure B.20. Mobile registration page.

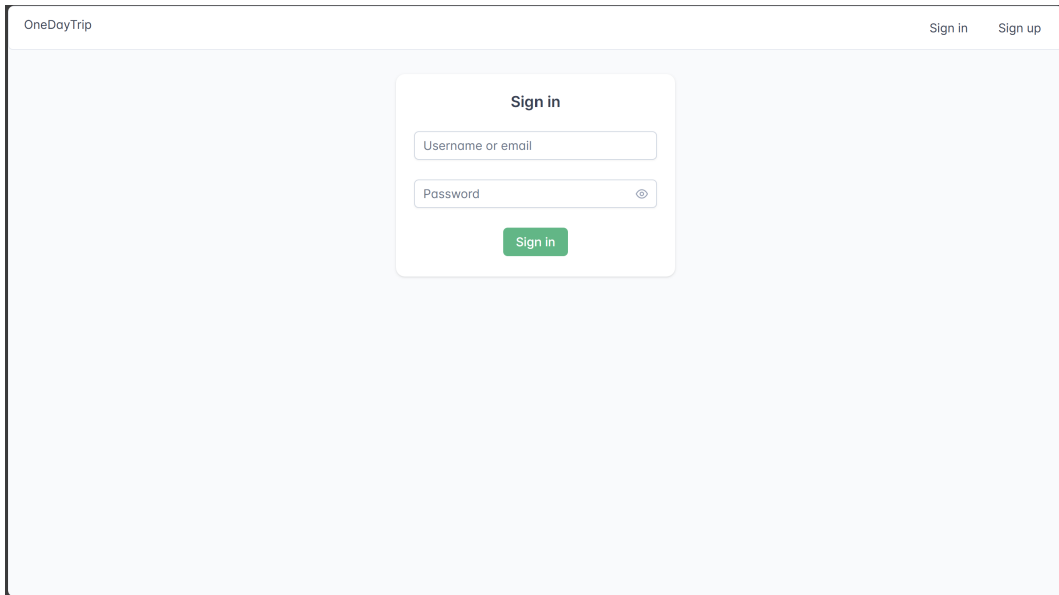


Figure B.21. Desktop login page.

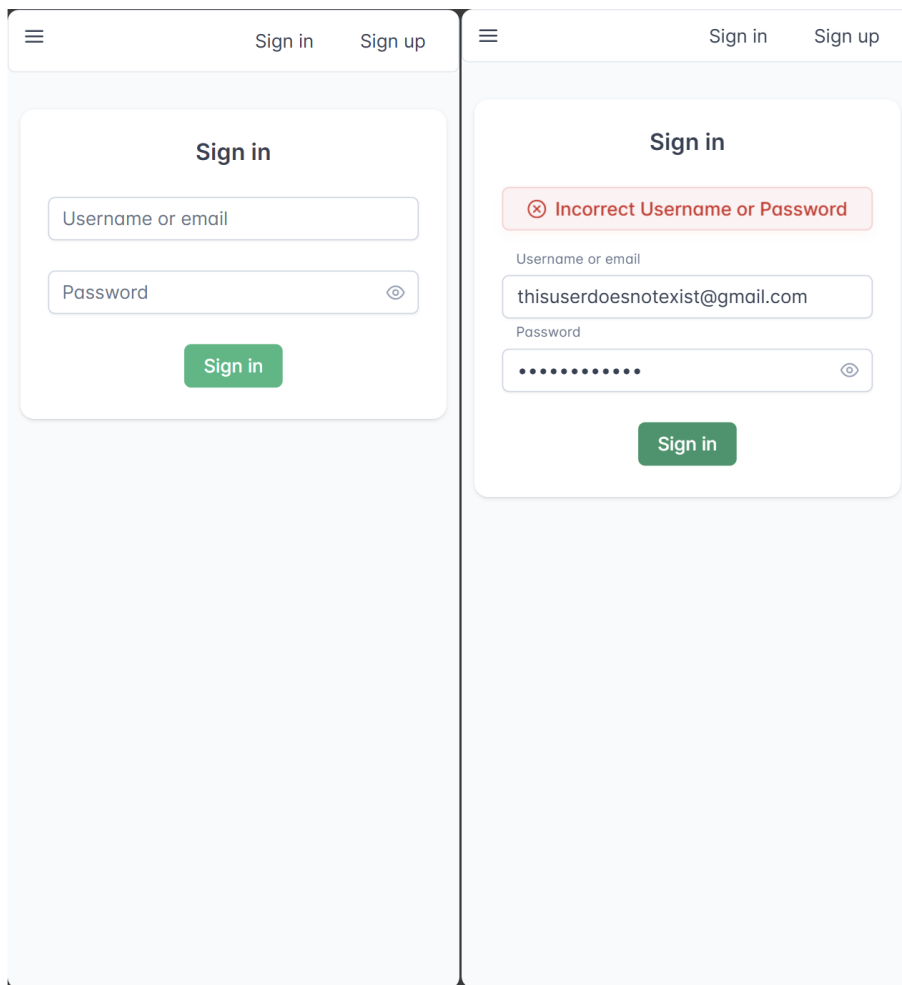


Figure B.22. Mobile login page.

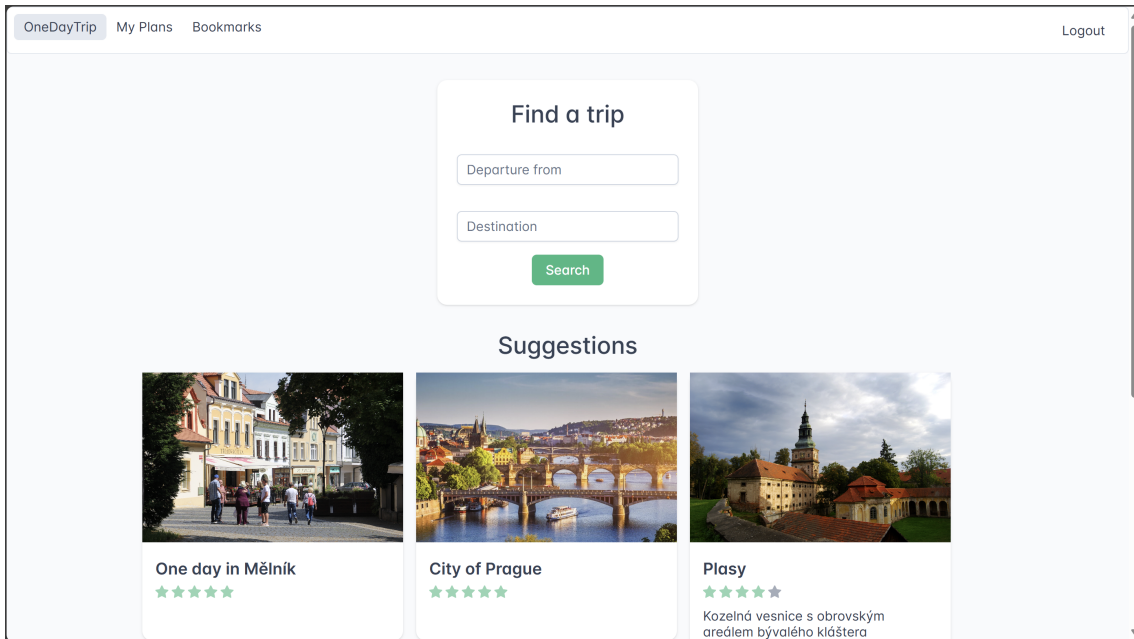


Figure B.23. Desktop home page.

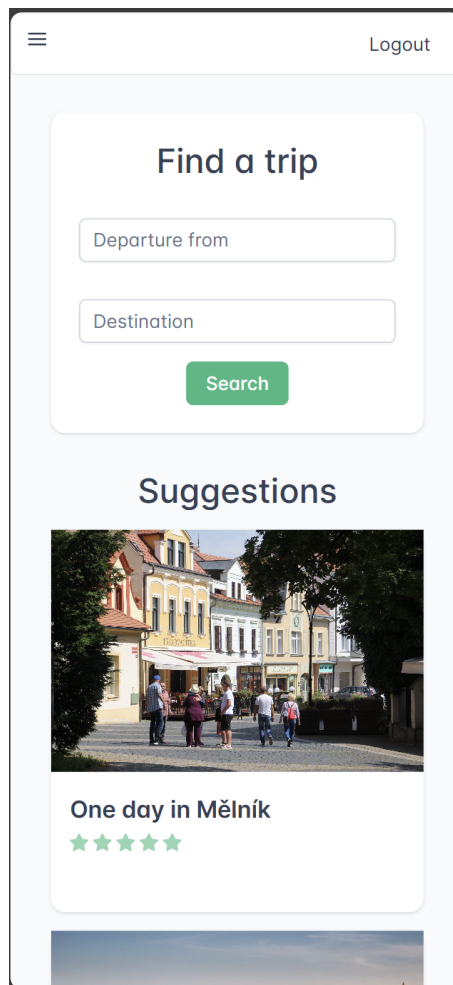


Figure B.24. Mobile home page.

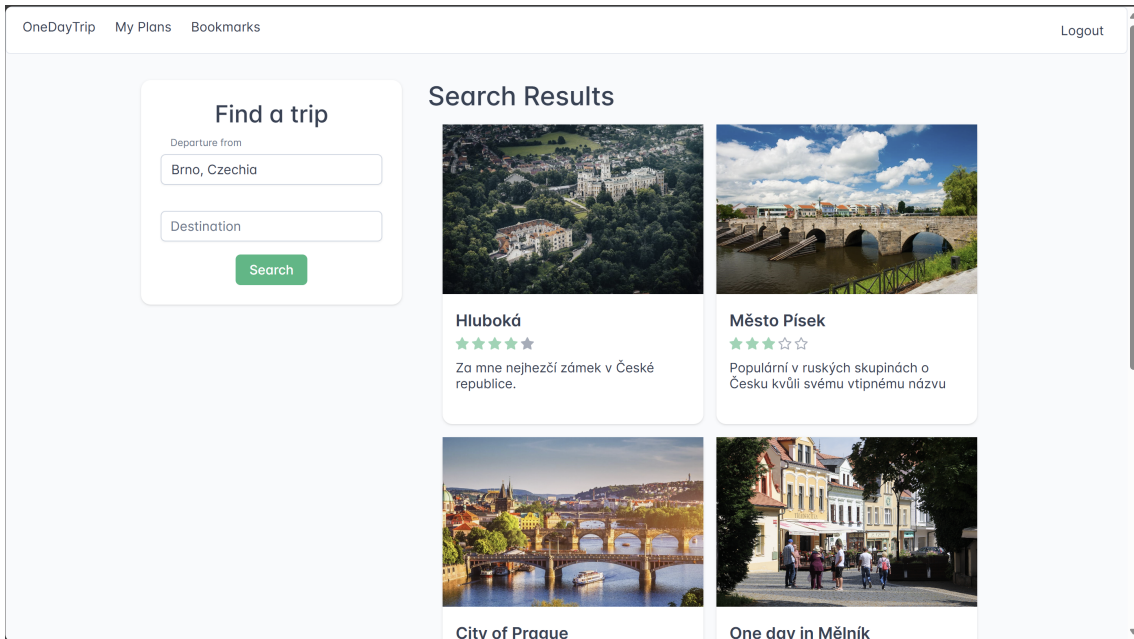


Figure B.25. Desktop search results page.

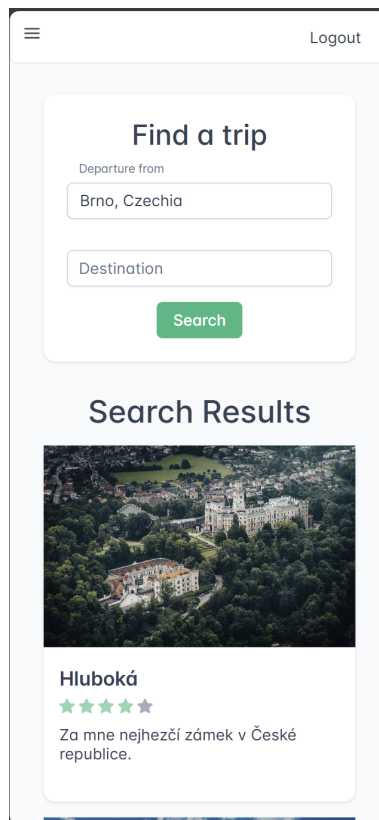


Figure B.26. Mobile search results page.

OneDayTrip My Plans Bookmarks
Logout

Hluboká

📍
🌟🌟🌟🌟🌟
 Total rating 4.5
 Author Ladislav
 Created 2024-05-08 12:04
 Last update 2024-05-08 12:38
 Za mne nejhezčí zámek v České republice.

premise

Hluboká Castle

📍 🗺️
 Státní zámek Hluboká, Zámek 142, 373 41 Hluboká nad Vítavou, Czechia

Dost vysoké vstupné. Neberou ITIC!

...Some more POIs...

point_of_interest
establishment
zoo
tourist_attraction
hiking_area
park

Zoo Hluboká

📍 🗺️
 Ohrada 417, 373 41 Hluboká nad Vítavou, Czechia

Mají hezké papoušky.

Comments

(You) Angelina ✎ ✕

★★★★★

I would like to visit it!

2024/05/20 09:51

Ladislav

★★★★☆

Už nemám chybu.

2024/05/08 12:40

Ladislav2

★★★★☆

Máš tam hroznou chybu!

2024/05/08 12:19

Figure B.27. Desktop trip plan page.

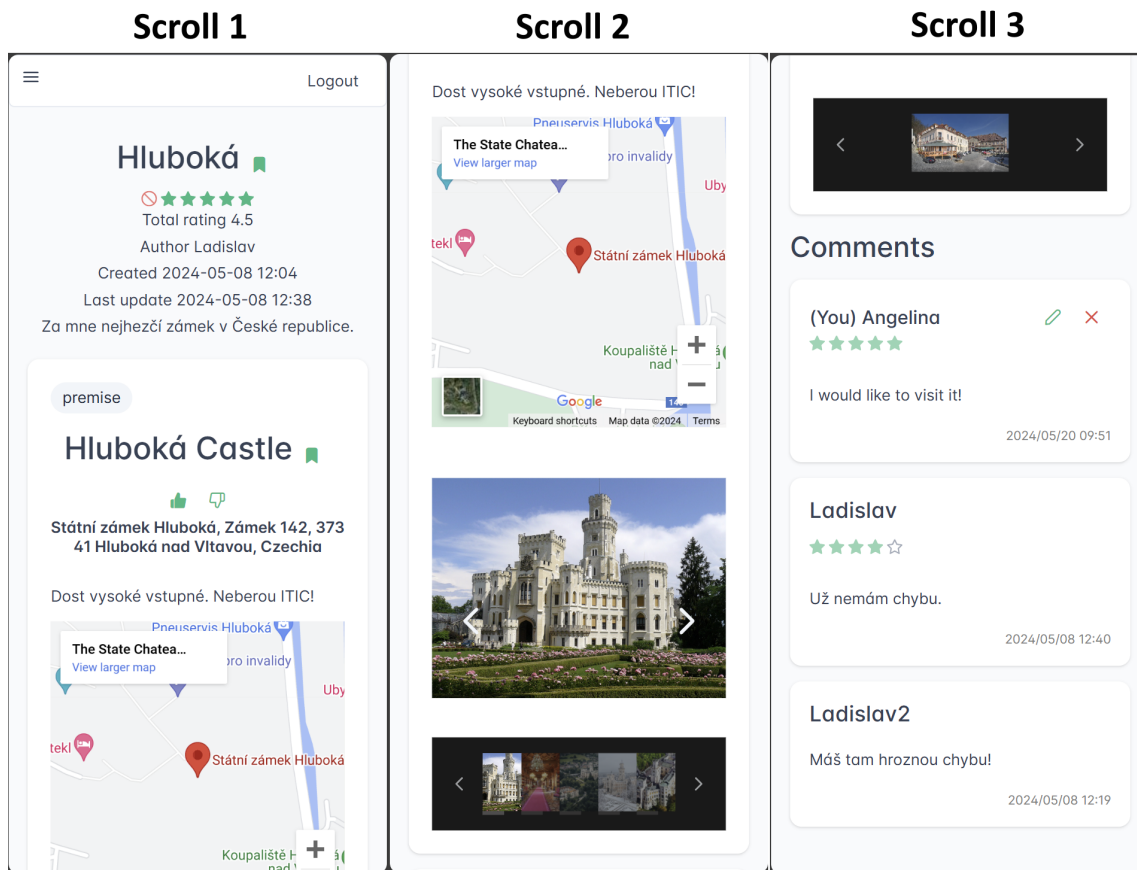


Figure B.28. Mobile trip plan page.

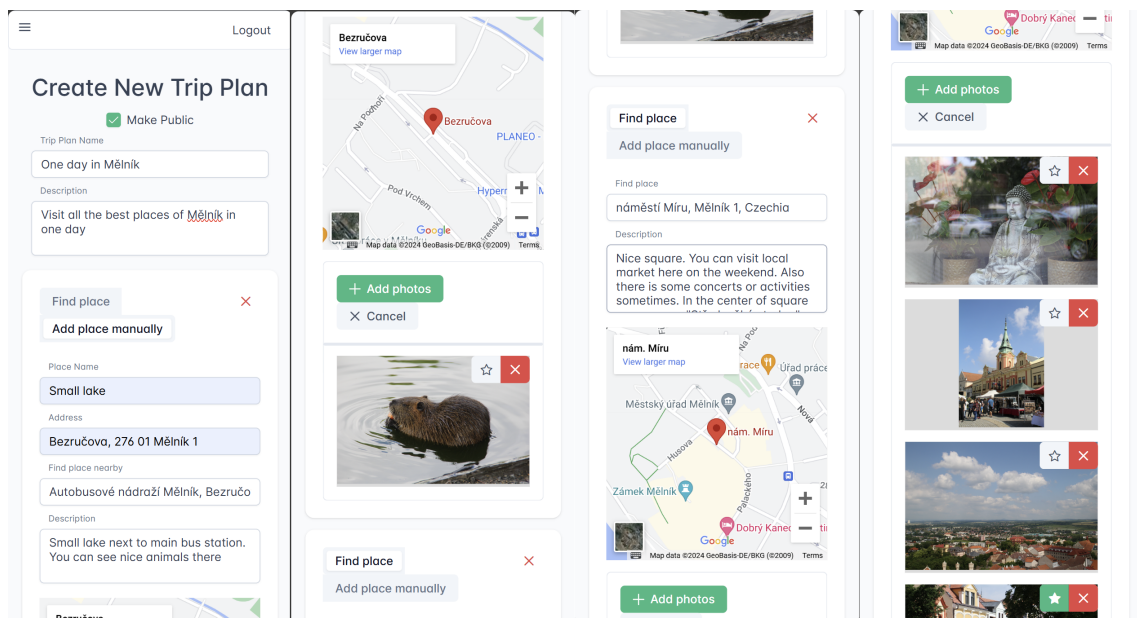


Figure B.29. Mobile create new trip plan page.

OneDayTrip My Plans Bookmarks Logout

Create New Trip Plan

Make Public

Trip Plan Name

Description

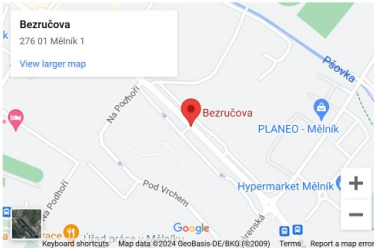
Find place Add place manually

Place Name

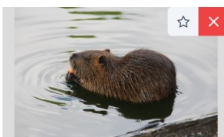
Address

Find place nearby

Description



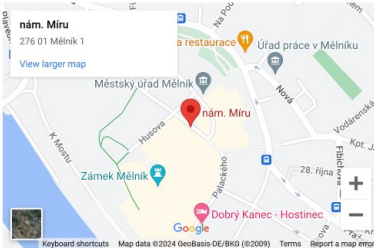
+ Add photos
✕ Cancel




Find place Add place manually


Find place

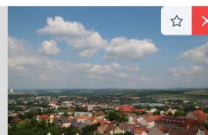
Description





+ Add photos
✕ Cancel











Add place
Save

Figure B.30. Desktop create new trip plan page.

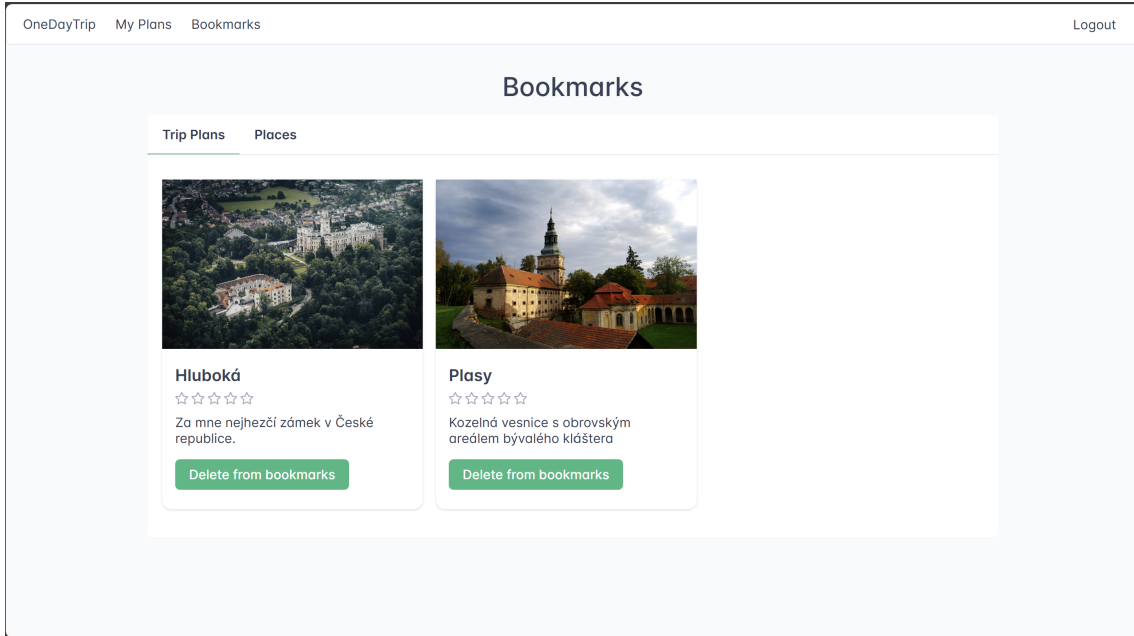


Figure B.31. Desktop user's bookmarks page.

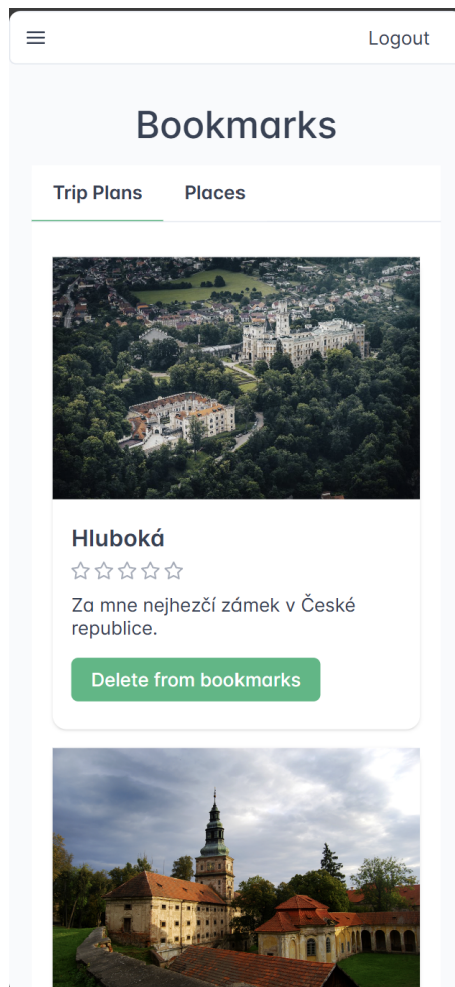


Figure B.32. Mobile user's bookmarks page.

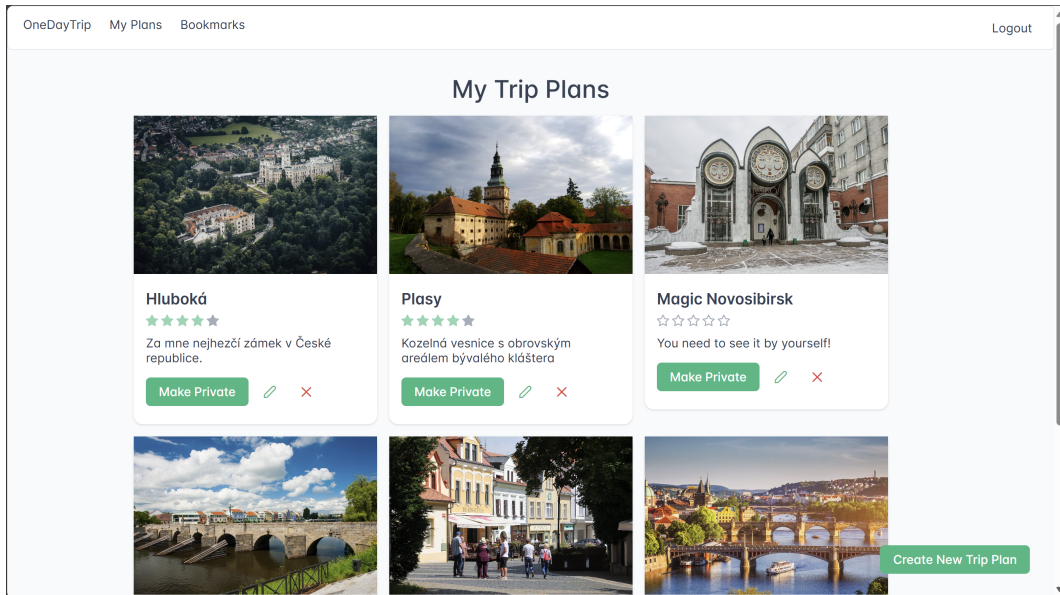


Figure B.33. Desktop user's trip plans page.

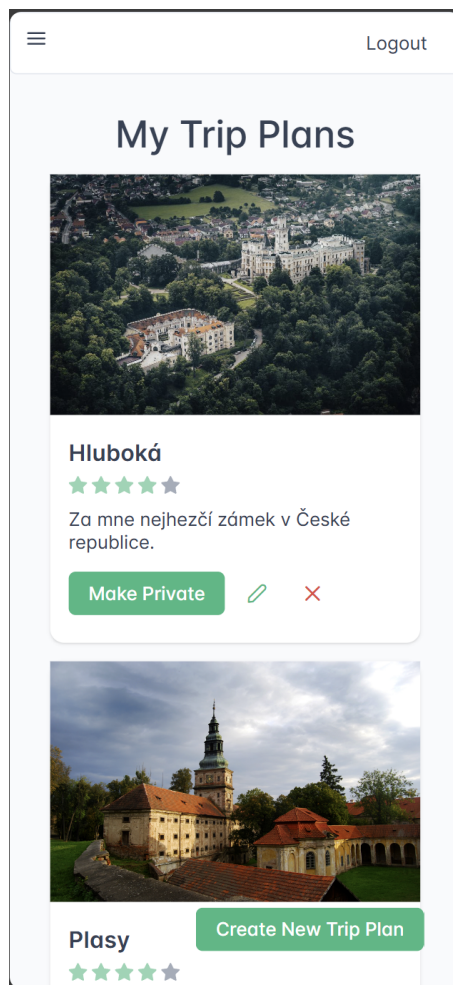


Figure B.34. Mobile user's trip plans page.

Appendix C

Test cases

Id	2
Related requirements	FRQ-011
Title	Successful registration
Steps	<p>Navigate to the “Sign up” page. Perform each of the following validation tests one by one.</p> <ol style="list-style-type: none"> 1. Enter a valid username. Username must be between 5 and 20 characters. 2. Enter a valid email address. 3. Enter a valid password. Password must contain at least one lowercase letter, at least one uppercase letter and at least one numeric digit. The password minimum length is 8 characters. 4. Repeat the entered password. Ensure the repeated password matches the initial password. 5. Click on the Sign up button. 6. If error message that user with same username or email already exists, repeat the attempt with other data.
Expected result	<p>User account successfully created, and user is logged in.</p> <p>Verify successful login by checking existence of “Logout” button on the navigation bar.</p>

Table C.2. Test case: Successful registration.

Id	5
Related requirements	FRQ-013
Title	Logout
Steps	<p>To execute this test user must be logged in first.</p> <ol style="list-style-type: none"> 1. Click on “Logout” button on the navigation bar.
Expected result	<p>User is logged out. Verify successful logout by checking that “Logout” button is not presented on the navigation bar.</p>

Table C.5. Test case: Logout.

Id	1
Related requirements	FRQ-011
Title	Registration form validation
Steps	<p>Navigate to the “Sign up” page. Perform each of the following validation tests one by one.</p> <ol style="list-style-type: none"> 1. Open the sign-up page. 2. Leave the username field empty and attempt to submit the form. 3. Enter a username shorter than 5 characters (e.g., <code>abc</code>) and submit the form. 4. Enter a username longer than 20 characters (e.g. <code>thisisaverylongusernameexceedingthelimit</code>) and submit the form. 5. Leave the email field empty and attempt to submit the form. 6. Enter an invalid email format (e.g. <code>invalidemail@</code>) and submit the form. 7. Leave the password field empty and attempt to submit the form. 8. Enter a password shorter than 8 characters (e.g., <code>Abc123</code>) and submit the form. 9. Enter a password without any uppercase letters (e.g., <code>abc12345</code>) and submit the form. 10. Enter a password without any lowercase letters (e.g., <code>ABC12345</code>) and submit the form. 11. Enter a password without any numeric digits (e.g., <code>Abcdefgh</code>) and submit the form. 12. Leave the repeat password field empty and attempt to submit the form. 13. Enter a password to the Repeat password” field other than password in “Password” field.
Expected result	<ol style="list-style-type: none"> 1. Sign-up page is shown. 2. Error message <code>Username is required</code> is displayed. 3. Error message <code>Minimum length required: 5 characters</code> is displayed. 4. Error message <code>Maximum length allowed: 20 characters</code> is displayed. 5. Error message <code>Email is required</code> is displayed. 6. Error message <code>Invalid email format</code> is displayed. 7. Error message <code>Password is required</code> is displayed. 8. Error message <code>Minimum length required: 8 characters</code> is displayed. 9. Error message <code>At least one uppercase</code> is displayed. 10. Error message <code>At least one lowercase</code> is displayed. 11. Error message <code>At least one numeric</code> is displayed. 12. Error message <code>Please repeat the password</code> is displayed. 13. Error message <code>Passwords do not match</code> is displayed.

Table C.1. Test case: Registration form validation.

Id	3
Related requirements	FRQ-011
Title	Registration failed because user with same email or username already exists
Steps	<p>Navigate to the “Sign up” page. Perform each of the following validation tests one by one.</p> <ol style="list-style-type: none"> 1. Enter the username of the existing user. 2. Enter a valid email address. 3. Enter a valid password. Password must contain at least one lowercase letter, at least one uppercase letter and at least one numeric digit. The password minimum length is 8 characters. 4. Repeat the entered password. Ensure the repeated password matches the initial password. 5. Click on the Sign up button. 6. Enter a new valid username. Username must be between 5 and 20 characters. 7. Enter the email address of the existing user. 8. Click on the Sign up button.
Expected result	After the steps 5 and 8 validation message saying that user with same username or email already exists must be shown.

Table C.3. Test case: Registration failed because user with same email or username already exists.

Id	4
Related requirements	FRQ-012
Title	Successful login
Steps	<p>Navigate to the “Sign in” page. Perform each of the following validation tests one by one.</p> <ol style="list-style-type: none"> 1. Enter the username of the existing user. 2. Enter the correct password associated with the username. 3. Click “Sign in” button. 4. Logout by clicking “Logout” button, 5. Navigate to the “Sign in” page. 6. Enter the email of the existing user. 7. Enter the correct password associated with the email. 8. Click “Sign in” button.
Expected result	After steps 3 and 8 the user must be logged in. Verify successful login by checking existence of “Logout” button on the navigation bar.

Table C.4. Test case: Successful login.

Id	6
Related requirements	FRQ-0201, FRQ-0220
Title	Search for a place to visit.
Steps	Navigate to the main page. 1. Enter departure place to filter. 2. Enter destination place to filter. 3. Click on “Search” button.
Expected result	Trip plans displayed in the search results are sorted by their distance from the departure and destination points. Trip plans with locations closest to either the departure or destination points will appear at the top of the search results, while those farthest away will be listed at the bottom.

Table C.6. Test case: Search for a place to visit.

Id	7
Related requirements	FRQ-0202
Title	View trip plan
Steps	Navigate to the main page, fill in the searching filter and click on “Search” button. 1. Click on any found trip plan.
Expected result	Trip plan page is opened, and trip plan details are shown.

Table C.7. Test case: View trip plan.

Id	8
Related requirements	FRQ-0203, FRQ-0204, FRQ-0205, FRQ-0206
Title	Bookmarks management
Steps	<p>Open any trip plan page.</p> <ol style="list-style-type: none"> 1. Add trip plan to bookmarks by clicking the bookmark button near the trip plan name. 2. Add trip plan block (POI) to bookmarks by clicking the bookmark button near the POI name. 3. Delete trip plan from bookmarks by clicking the bookmark button near the trip plan name. 4. Delete trip plan block (POI) from bookmarks by clicking the bookmark button near the POI name. 5. Repeat steps 1 and 2 to add the trip plan and POI to bookmarks again. 6. Open the bookmarks page by clicking on “Bookmarks” button on navigation bar. 7. Delete trip plan from bookmarks by clicking the “Delete” button on the trip plan card. 8. Delete trip plan block (POI) from bookmarks by clicking the “Delete” button on the POI card.
Expected result	<p>Open the bookmarks page by clicking on “Bookmarks” button on navigation bar. After steps 1 and 2 check that trip plan bookmark and trip plan block (POI) bookmark are presented in the list of bookmarks. After steps 3 and 4 check that the trip plan bookmark and trip plan block (POI) bookmark are not presented in the list of bookmarks. After steps 7 and 8 check that trip plan bookmark and trip plan block (POI) bookmark are not presented in the list of bookmarks</p>

Table C.8. Test case: Bookmarks management.

Id	9
Related requirements	FRQ-0209
Title	Comment the trip plan
Steps	Open any trip plan page and scroll to the Comment section. <ol style="list-style-type: none"> 1. Write a comment and save it. 2. Edit the comment and save the changes. 3. Delete the comment.
Expected result	<ol style="list-style-type: none"> 1. The comment must be presented in the list of comments. 2. The updated comment must be presented in the list of comments 3. Comment must be deleted from the list of comments.

Table C.9. Test case: Comment the trip plan.

Id	10
Related requirements	FRQ-0210, FRQ-0211
Title	Trip plan and trip plan block rating
Steps	Open any trip plan page. <ol style="list-style-type: none"> 1. Rate the trip plan by clicking the rating button near the trip plan name. 2. Click “Like” button near the POI name. 3. Click “Like” button near the POI name again. 4. Click “Dislike” button near the POI name. 5. Click “Dislike” button near the POI name again.
Expected result	<ol style="list-style-type: none"> 1. The given rating must be shown for the trip plan. 2. Like must be shown near the POI name. 3. No rating must be shown for the POI. 4. Dislike must be shown near the POI name. 5. No rating must be shown for the POI.

Table C.10. Test case: Trip plan and trip plan block rating.

Id	11
Related requirements	FRQ-0213, FRQ-0214, FRQ-0215, FRQ-0216, FRQ-0217, FRQ-0218
Title	Create a trip plan
Steps	<p>Open “My Trip Plans” page.</p> <ol style="list-style-type: none"> 1. Click on “Create” button. 2. Fill the trip plan information. 3. Click the publicity checkbox to make trip plan public. 4. Add POI to trip plan by clicking on add POI button. 5. Fill the POI information. Find the place. 6. Add images to POI in multiple steps. Add images, delete some of them, add some more. 7. Repeat steps 4, 5, 6 multiple times. 8. Click “Save” button
Expected result	<ol style="list-style-type: none"> 1. Trip plan creation page must be shown. 4. New trip plan block must be added. 5. After a place was found with a help of autocomplete, it must be shown on map. 6. Uploaded images must be shown on UI. 8. Trip plan must be created and redirection to trip plan page must be done. Verify that all filled information is presented on the trip plan page. Check the post publicity by searching the trip plan via trip plans search.

Table C.11. Test case: Create a trip plan.

Id	12
Related requirements	FRQ-0207
Title	Change the trip plan publicity
Steps	<p>Open “My Trip Plans” page. To execute the test trip plan must be created.</p> <ol style="list-style-type: none"> 1. Click on “Make public” button on the trip plan card. 2. Click on “Make private” button on the trip plan card.
Expected result	<p>Check the post publicity by finding the trip plan via trip plans search.</p> <ol style="list-style-type: none"> 1. Trip plan must be presented in trip plan search results. 2. Trip plan should not be presented in trip plan search results.

Table C.12. Test case: Change the trip plan publicity.

Id	13
Related requirements	FRQ-0213, FRQ-0214, FRQ-0215, FRQ-0216, FRQ-0217, FRQ-0218
Title	Update the trip plan
Steps	<p>Open “My Trip Plans” page. For this test trip plan with at least 2 POIs must exist.</p> <ol style="list-style-type: none"> 1. Click on “Edit” button. 2. Change the trip plan information. 3. Change the trip plan publicity by clicking on the publicity checkbox. 4. Delete existing POI block. 5. Change existing POI information. 6. Change existing POI images. This step includes deleting some existing images and adding additional images. 7. Add POI to trip plan by clicking on add POI button. 8. Fill the POI information. Find the place. 9. Add images to POI in multiple steps. Add images, delete some of them, add some more. 10. Repeat steps 7, 8, 9 multiple times. 11. Click “Save” button
Expected result	<ol style="list-style-type: none"> 1. Trip plan editing page must be shown. 4. Place block must be deleted. 6. Uploaded images must be shown on UI. Deleted images should not be presented. 7. New trip plan block must be added. 8. After a place was found with a help of autocomplete, it must be shown on map. 9. Uploaded images must be shown on UI. 11. Trip plan must be edited and redirection to trip plan page must be done. Verify that all changes were applied correctly and presented on the trip plan page. Check the post publicity by searching the trip plan via trip plans search.

Table C.13. Test case: Update the trip plan.

Id	14
Related requirements	FRQ-0208
Title	Delete the trip plan
Steps	Open “My Trip Plans” page. To execute the test trip plan must be created. 1. Click on “Delete” button on the trip plan card.
Expected result	Trip plan must be deleted. Verify this by attempting to access the trip plan page of the deleted plan.

Table C.14. Test case: Delete the trip plan.

Appendix D

Acronyms

- ACID ■ Atomicity, Consistency, Isolation, Durability principles that ensure the reliability and integrity of transactions in a database system.
- API ■ Application Programming Interface.
- BG ■ Business Goal.
- BLL ■ Business Logic Layer.
- CDN ■ Content Delivery Network.
- CRUD ■ Create, Read, Update, Delete operations.
- DAL ■ Data Access Layer.
- DB ■ Database.
- DI ■ Dependency Injection.
- DTO ■ Data Transfer Objects.
- GIS ■ Geographic Information Systems.
- GPS ■ Global Positioning System.
- GUI ■ Graphical user interface.
- IoC ■ Inversion of Control.
- JWT ■ JSON Web Token.
- ORM ■ Object–relational mapping.
- OS ■ Operating System.
- POI ■ Point of Interest, place of interest.
- SPA ■ Single-Page Application.
- SRID ■ Spatial Reference System ID.
- UI ■ User Interface.
- USD ■ United States Dollar.