



EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPT. OF MEDIA AND EDUCATIONAL TECHNOLOGY

Map Management Platform for Sharing Travel Experiences

Supervisor:

Horváth Győző

Associate professor, PhD

Author:

Ivan Pylypenko

Computer Science BSc

Budapest, 2025

Contents

1	Introduction	3
2	User documentation	4
2.1	App Usage	4
2.1.1	System Requirements	4
2.1.2	Running the app	4
2.2	User Interface Overview	5
2.2.1	Navigation Bar	5
2.2.2	Signup page	6
2.2.3	Login page	7
2.2.4	Profile page	8
2.2.5	Edit Profile Page	9
2.2.6	Explore page	10
2.2.7	Create page	11
2.2.8	User Page	13
2.2.9	Place page	13
2.2.10	Map page	14
2.3	Error Handling and Validation	14
2.3.1	Signup Page	14
2.3.2	Login Page	16
2.3.3	Create Page	17
3	Developer documentation	19
3.1	Functional requirements	19
3.2	Non-functional requirements	21
3.3	Windows navigation	22
3.4	Functionality	23
3.4.1	Unauthorized user	23

3.4.2	Authorized user	24
3.5	Technologies	25
3.6	Installation	26
3.6.1	Prerequisites	26
3.6.2	Environment Variables	26
3.6.3	Frontend Installation	27
3.6.4	Backend Installation	27
3.7	Architecture	28
3.7.1	Frontend Layer	29
3.7.2	Backend Layer	33
3.7.3	Database Layer	40
3.8	Testing	43
4	Conclusion	48
	Acknowledgements	49
	Bibliography	49
	List of Figures	53
	List of Tables	55

Chapter 1

Introduction

Planning a trip usually starts with asking friends for advice and scrolling social media for inspiration [1]. All those links and tips go into a note-taking app [2]. There is still no solution where people can save their spots, see them on a map, and share them with others in one click. The thesis is dedicated to my solution to this problem. The thesis presents a web application that lets users build and share maps containing their favorite places, such as restaurants, museums, parks, and more.

The idea originated from my own travel habits. Before every journey, I opened Apple Notes, pasted all the information and links to the places I planned to visit, and tried to make it understandable for others. However, the process quickly becomes messy: duplicate entries, broken links, and no visual representation of where these locations are. I wanted a more convenient solution - an application where I could keep places I visited or want to visit, clearly organized, see them spotted on a map, and add a title with a description for context. Map Management Platform for Sharing Travel Experiences provides all of these features [3].

Chapter 2

User documentation

This chapter introduces the end user to the Map Management Platform. It lists the system requirements, provides a manual on how to start using the application, and walks through each screen the user will interact with.

2.1 App Usage

2.1.1 System Requirements

To access and use the Map Management Platform, users must satisfy the following requirements:

- **Internet Connection:** A stable internet connection is required for loading maps, enabling autocomplete for places, and fetching data from the database.
- **Supported Browsers:**
 - Google Chrome (latest version recommended)
 - Microsoft Edge (latest version recommended)
- **Screen Resolution:** The platform is optimized for resolution of 1280x720 pixels and higher.

2.1.2 Running the app

Install the project as described in the Installation Section 3.6 in Developer documentation, then open the URL: `http://localhost:5173` in a supported browser.

2.2 User Interface Overview

The Map Management Platform provides a user-friendly interface [4]. The interface consists of a navigation, authentication pages, profile management page, exploration, and content creation.

2.2.1 Navigation Bar

The navigation bar is located at the top and provides quick access to the main sections of the application. Its content dynamically adjust based on the user's authentication status:

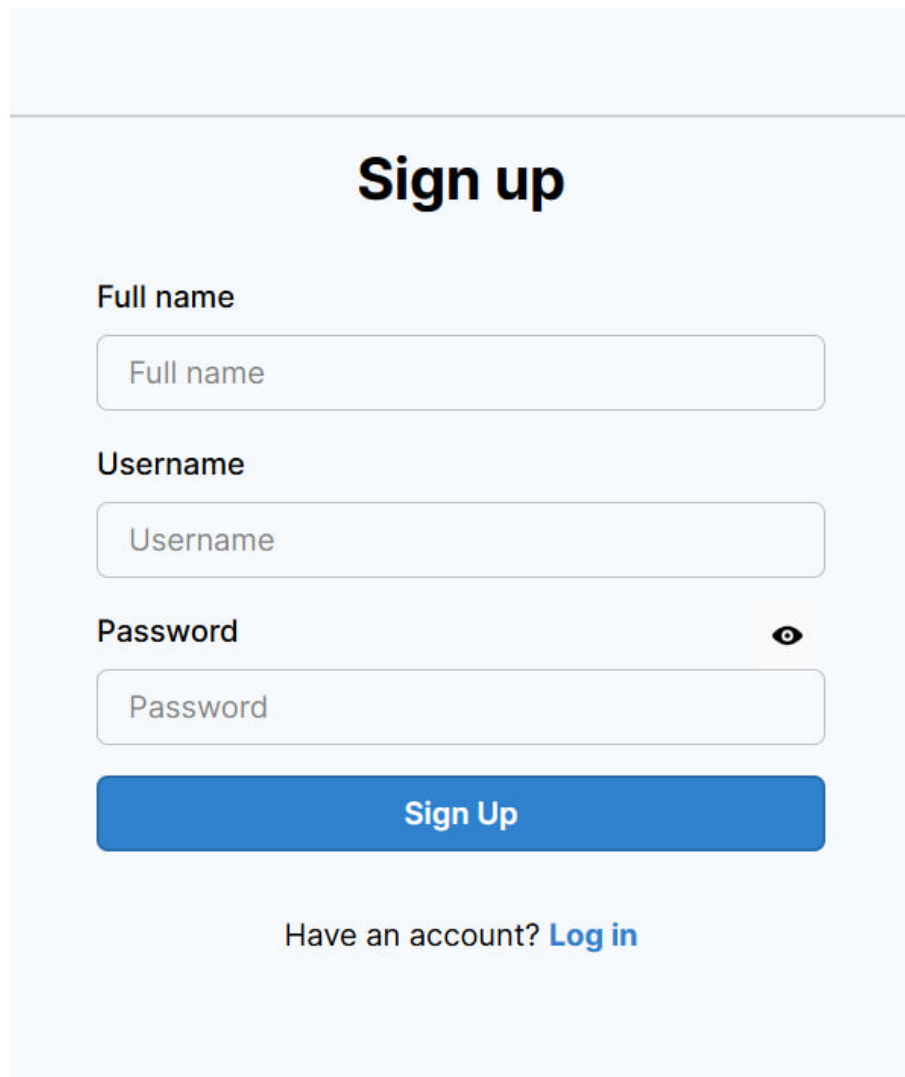
- **Unauthenticated Users:**

- Login/Signup Pages: only the “Explore” button is displayed.
- Explore Page: both “Login” and “Signup” buttons are displayed.

- **Authenticated Users:**

- “Profile,” “Explore,” and “Create” buttons are displayed. The active page button is highlighted.

2.2.2 Signup page




Sign up

Full name

Full name

Username

Username

Password 

Password

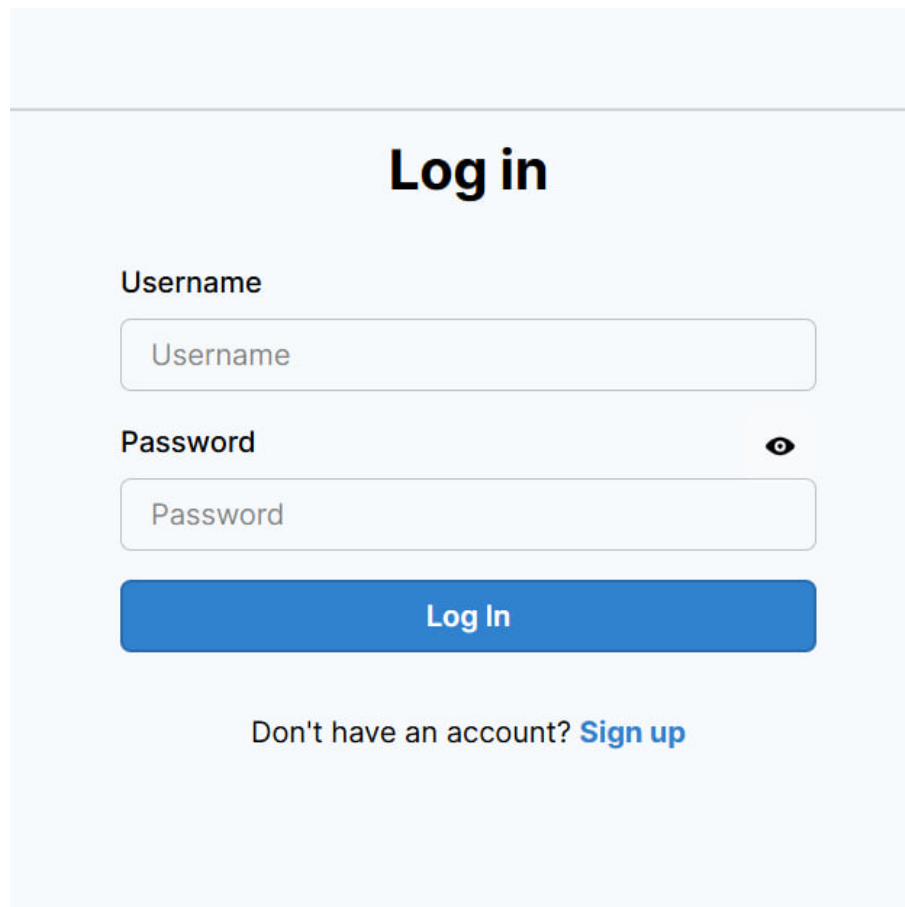
Sign Up

Have an account? [Log in](#)

Figure 2.1: Signup page

New users can register by entering a full name, username, and password. After successful registration, users are automatically redirected to their personal profile page. If fields are missing or invalid, appropriate error messages are displayed.

2.2.3 Login page



The image shows a login page with a light blue background. At the top, the text "Log in" is displayed in a large, bold, black font. Below this, there are two input fields. The first is labeled "Username" in a bold black font, and the input field itself contains the placeholder text "Username". The second is labeled "Password" in a bold black font, and the input field contains the placeholder text "Password". To the right of the password field is a small black eye icon, indicating a toggle for password visibility. Below the input fields is a blue button with the text "Log In" in white. At the bottom, there is a link that says "Don't have an account? Sign up", where "Sign up" is in blue and the rest is in black.

Figure 2.2: Login page

Existing users can log in by entering their username and password. After successful authentication, users are automatically redirected to their profile page. If fields are missing or invalid, appropriate error messages are displayed.

2.2.4 Profile page

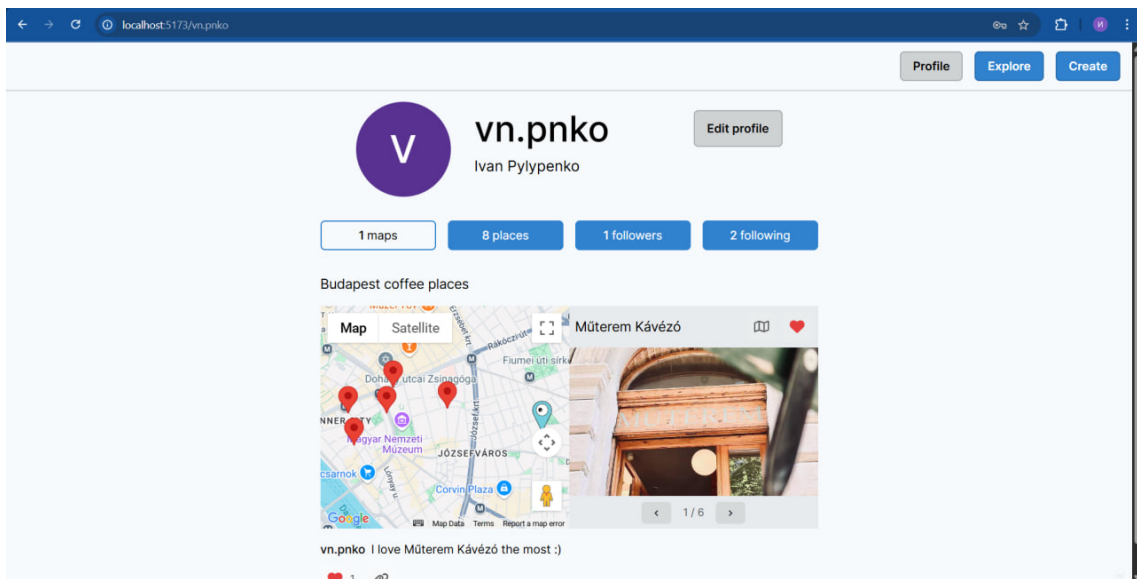


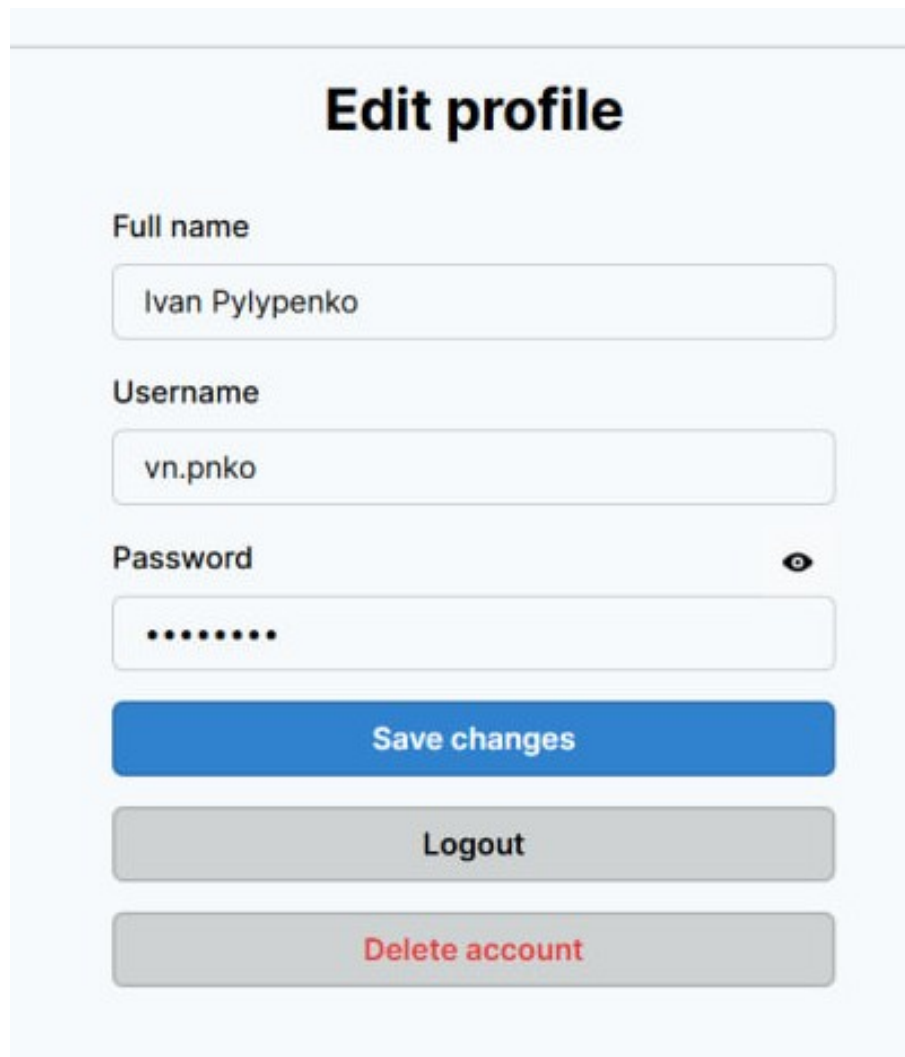
Figure 2.3: Profile page

Displays user data:

- Username and full name
- Liked maps and places
- Followers and followed users

Users can unlike maps, like/unlike places, and follow/unfollow their connections. Navigation to the Edit Profile Page is available from the Profile Page.

2.2.5 Edit Profile Page



The screenshot displays a web form titled "Edit profile" in a large, bold, black font. Below the title are three input fields. The first is labeled "Full name" and contains the text "Ivan Pylypenko". The second is labeled "Username" and contains the text "vn.pnko". The third is labeled "Password" and is filled with ten dots; a small eye icon is positioned to its right. Below these fields are three buttons: a blue button labeled "Save changes", a light gray button labeled "Logout", and another light gray button labeled "Delete account" in red text.

Figure 2.4: Edit profile page

The logged-in user can update personal data, logout, or delete the account. Form validation ensures that all required fields are filled.

2.2.6 Explore page

Users can explore other users, places, and maps. Logged-in users also receive personalized recommendations.

- Explore users

A search bar filters by username or full name. Clicking on a username redirects to that user's profile.

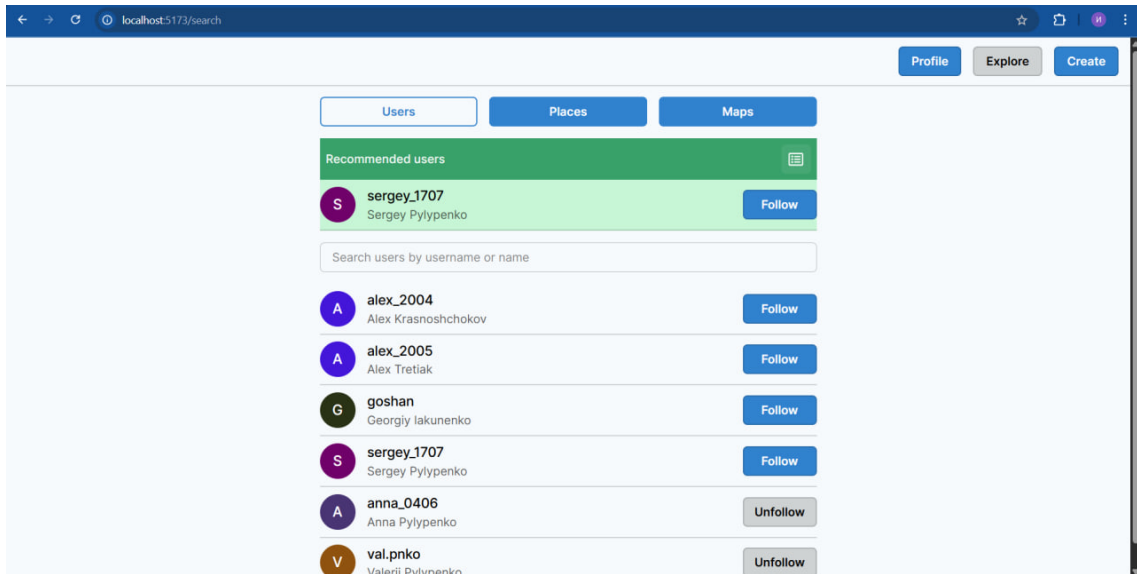


Figure 2.5: Explore page (explore users)

- Explore places

A search bar filters by place name. Clicking on a place name redirects to that place's profile. Users can like/unlike places.

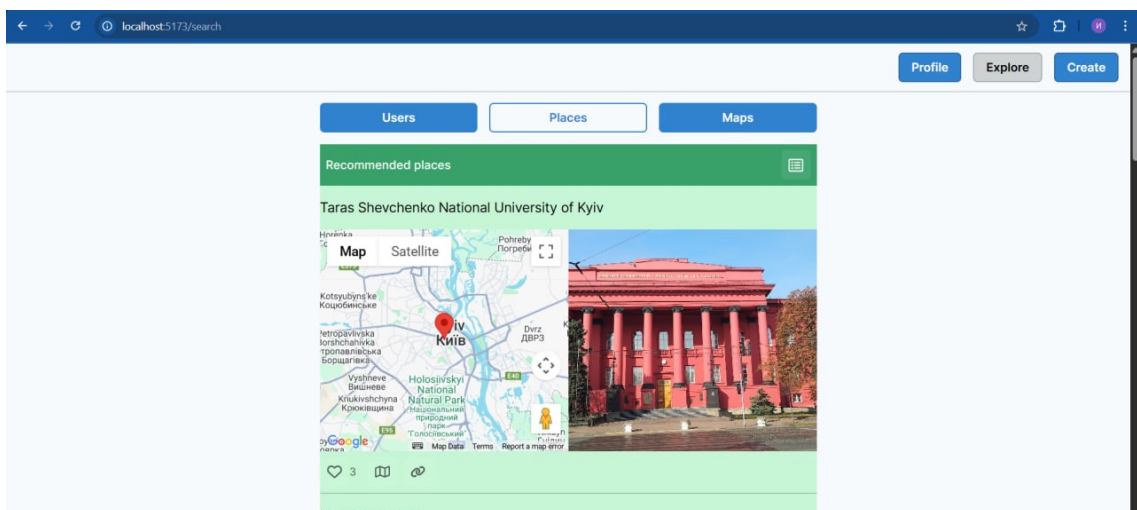


Figure 2.6: Explore page (explore places)

- Explore maps

A search bar filters by map name or its description. Clicking on a place name redirects to that map's profile. Users can like/unlike maps and their places. Clicking on a map-creator name redirects to that user's profile.

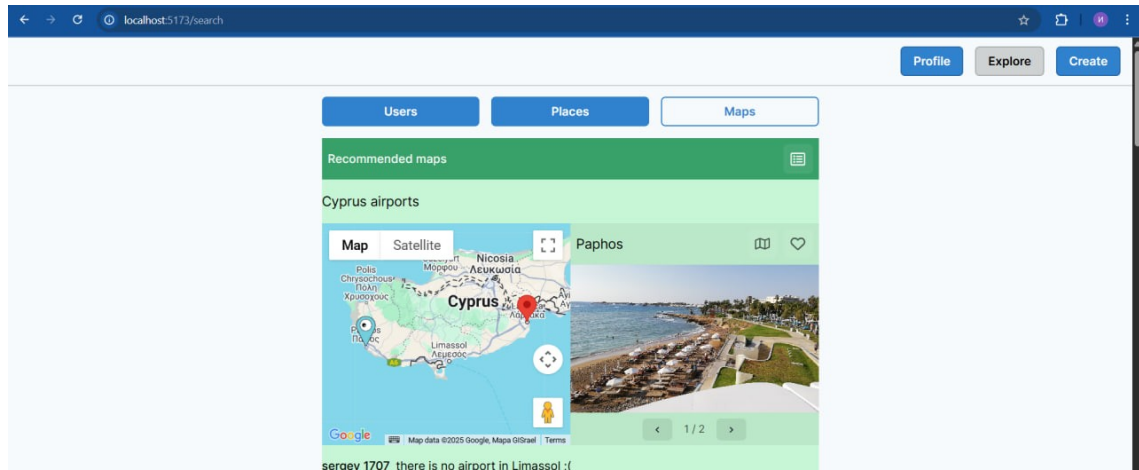


Figure 2.7: Explore page (explore maps)

2.2.7 Create page

Only authenticated users can create places and maps.

- Create place
 - Begin typing a place name in the search field.
 - Select a place from the autocomplete list.
 - Submit the form by clicking the "+" button.

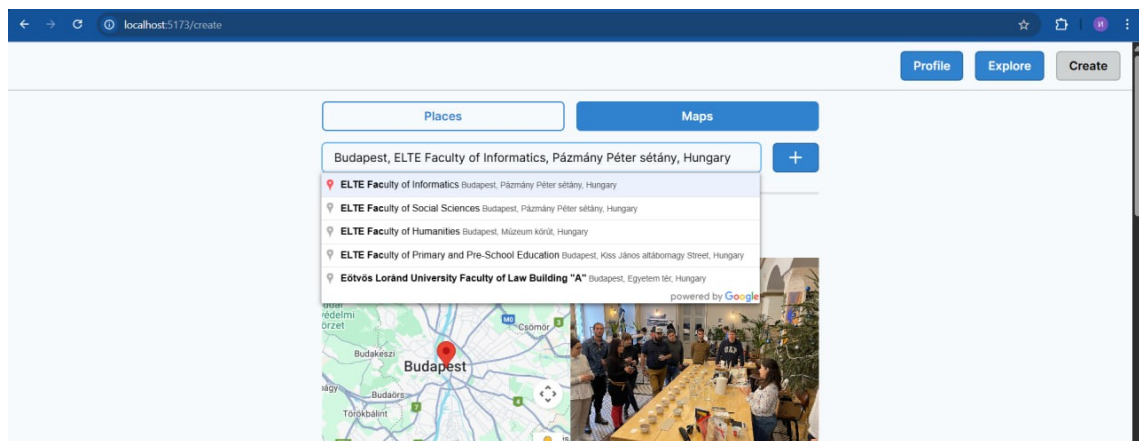


Figure 2.8: Create page (create place)

After submitting the form, the place is automatically liked by the user and appears at the top of their liked places.

- Create Map
 - Create a name and a description for the map.
 - Add at least two places to the map using the place creation form.
 - Submit the form by clicking the "Create map" button.

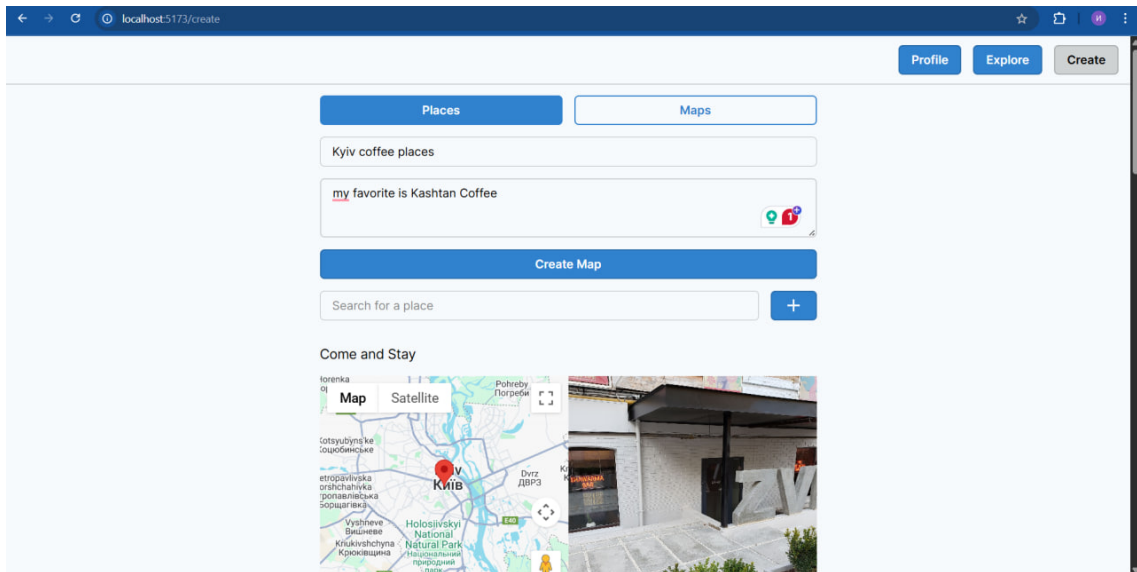


Figure 2.9: Create page (create maps)

After submitting the form, the map is automatically liked by the user and appears at the top of their liked maps.

2.2.8 User Page

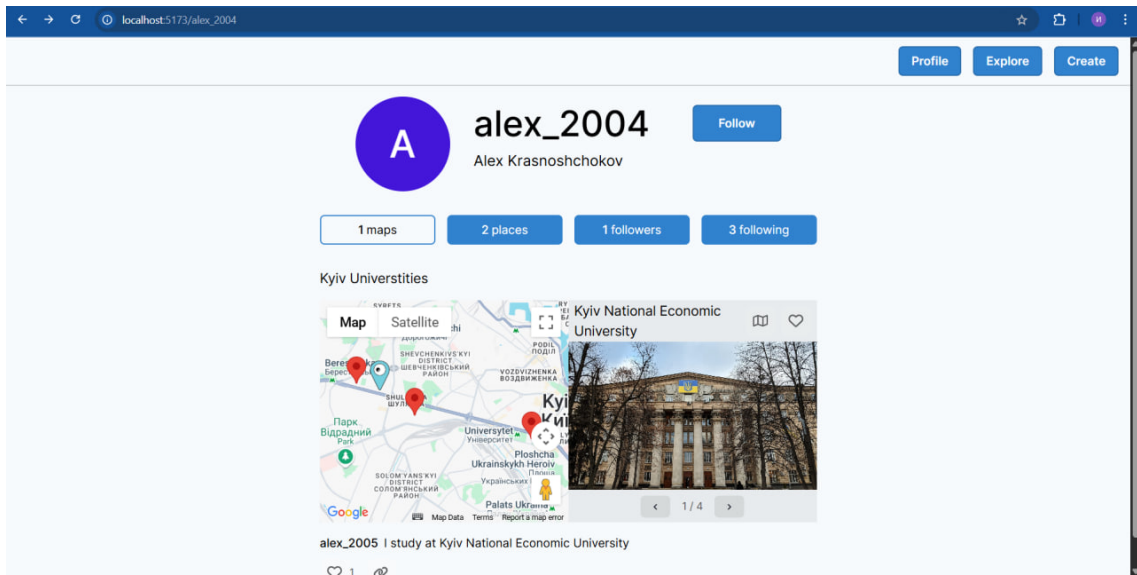


Figure 2.10: User page

The page shows detailed information about a specific user. Logged-in users can follow/unfollow the user and view their maps, places, followers, and followed users.

2.2.9 Place page

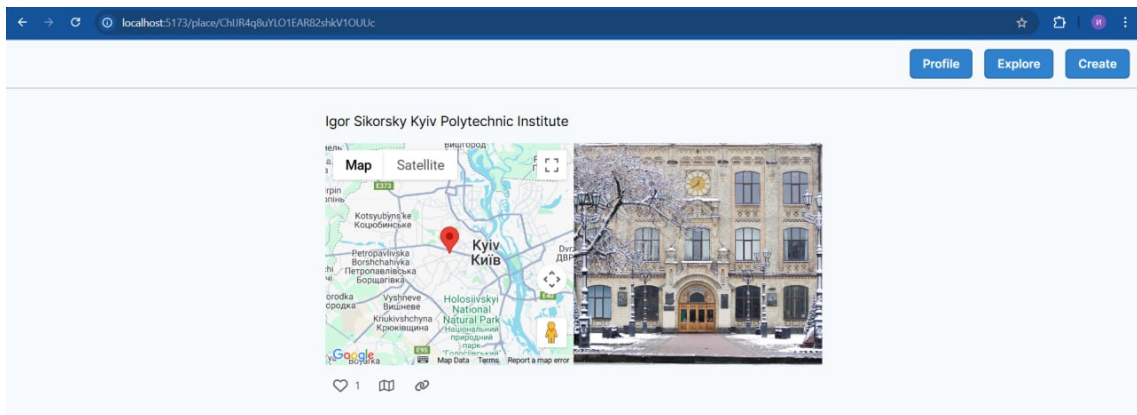


Figure 2.11: Place page

The page shows detailed information about a specific place. Logged-in users can:

- Like/unlike the place.
- Copy a link to the place's profile page.
- Navigate to the corresponding place in Google Maps.

2.2.10 Map page

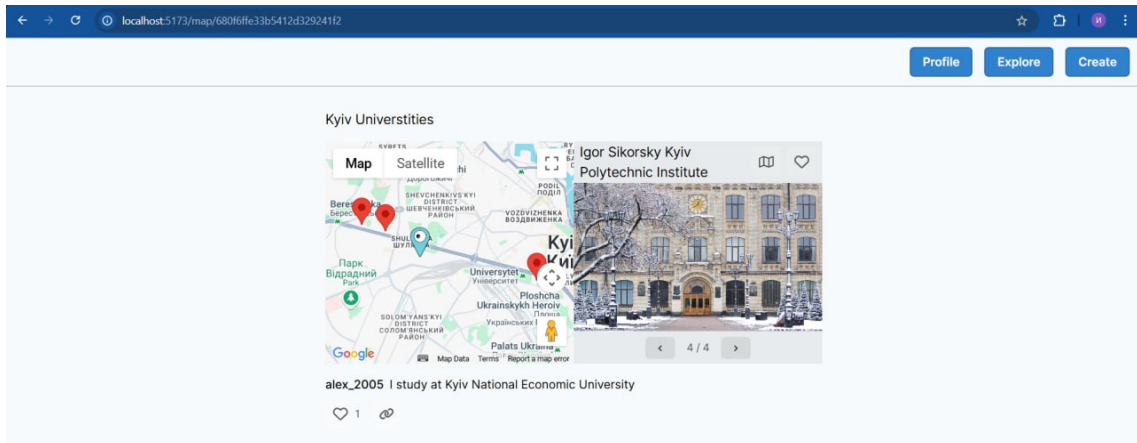


Figure 2.12: Map page

The page shows detailed information about a specific map. Logged-in users can:

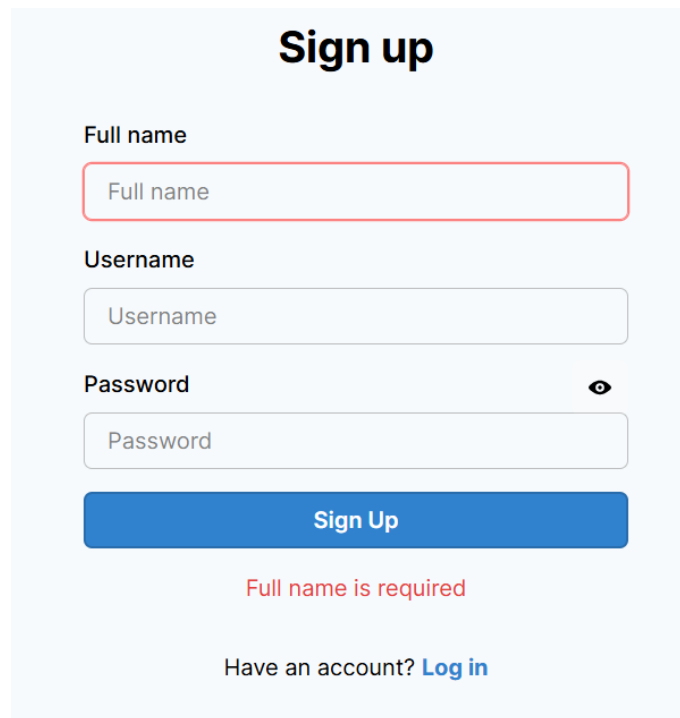
- Like/unlike the map.
- Like/unlike the places of the map.
- Copy the link to the map's profile page.
- Navigate to the profile pages of the places of the map.

The map highlights the selected place to simplify the exploration process.

2.3 Error Handling and Validation

2.3.1 Signup Page

- **Empty Field:** The empty input is highlighted and the appropriate error message appears.



The image shows a 'Sign up' form with three input fields: 'Full name', 'Username', and 'Password'. The 'Full name' field is highlighted with a red border, indicating an error. Below the fields is a blue 'Sign Up' button. A red error message 'Full name is required' is displayed below the button. At the bottom, there is a link 'Have an account? Log in'.

Sign up

Full name

Full name

Username

Username

Password

Password

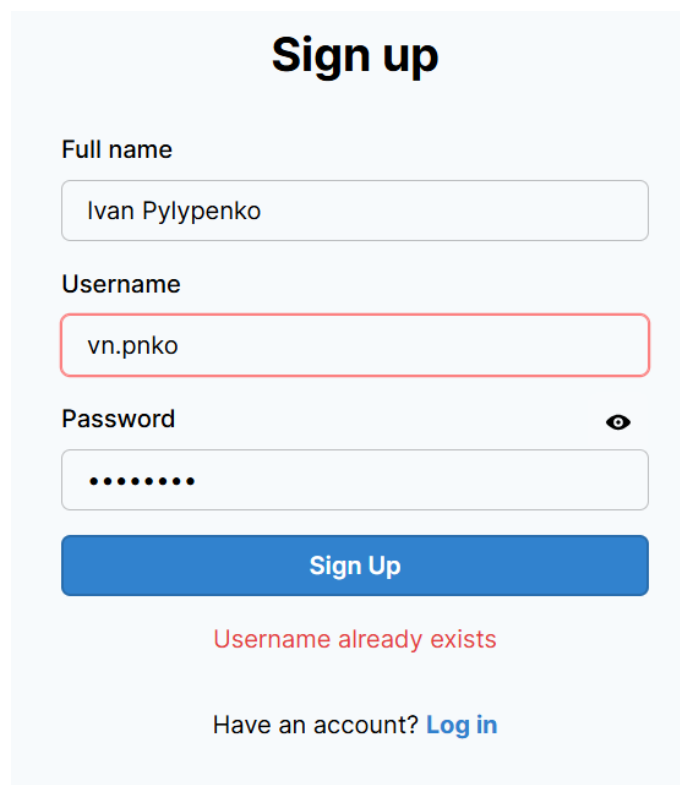
Sign Up

Full name is required

Have an account? [Log in](#)

Figure 2.13: Signup error (empty field)

- **Username Uniqueness:** If the chosen username is already taken, the username field is highlighted and the appropriate error message appears.



The image shows a 'Sign up' form with three input fields: 'Full name', 'Username', and 'Password'. The 'Username' field is highlighted with a red border, indicating an error. Below the fields is a blue 'Sign Up' button. A red error message 'Username already exists' is displayed below the button. At the bottom, there is a link 'Have an account? Log in'.

Sign up

Full name

Ivan Pylypenko

Username

vn.pnko

Password

.....

Sign Up

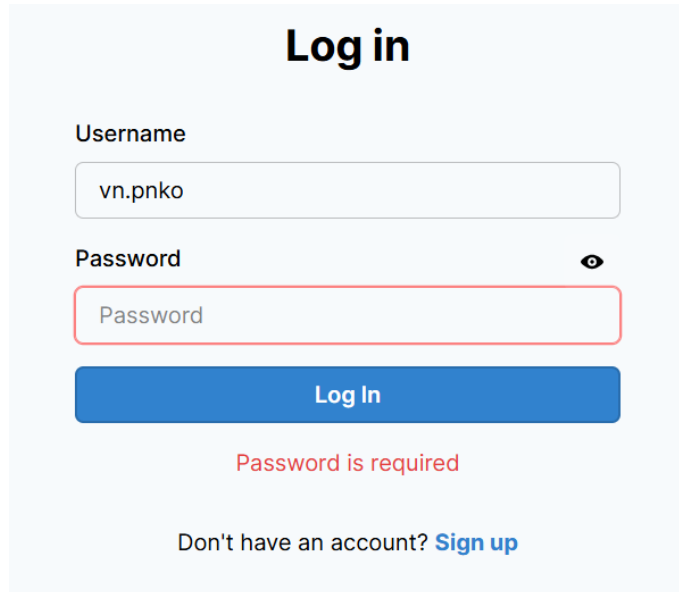
Username already exists

Have an account? [Log in](#)

Figure 2.14: Signup error (username uniqueness)

2.3.2 Login Page

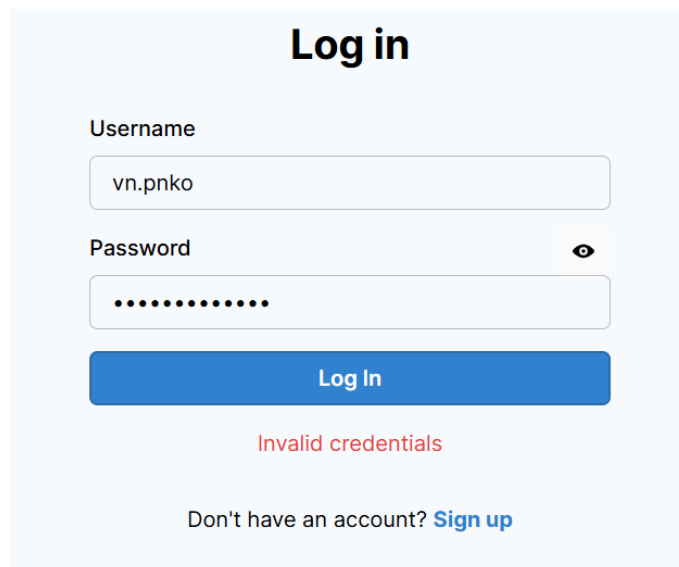
- **Empty Field:** The empty input is highlighted and the appropriate error message appears.



The image shows a login form titled "Log in". It has two input fields: "Username" and "Password". The "Username" field contains the text "vn.pnko". The "Password" field is empty and has a red border, indicating an error. Below the password field is a blue "Log In" button. Below the button is a red error message "Password is required". At the bottom, there is a link "Don't have an account? Sign up".

Figure 2.15: Login error (empty field)

- **Invalid Credentials:** If the provided username or password is incorrect, the appropriate error message appears.



The image shows a login form titled "Log in". It has two input fields: "Username" and "Password". The "Username" field contains the text "vn.pnko". The "Password" field contains a series of dots, indicating that the password is masked. Below the password field is a blue "Log In" button. Below the button is a red error message "Invalid credentials". At the bottom, there is a link "Don't have an account? Sign up".

Figure 2.16: Login error (invalid credentials)

2.3.3 Create Page

Create place

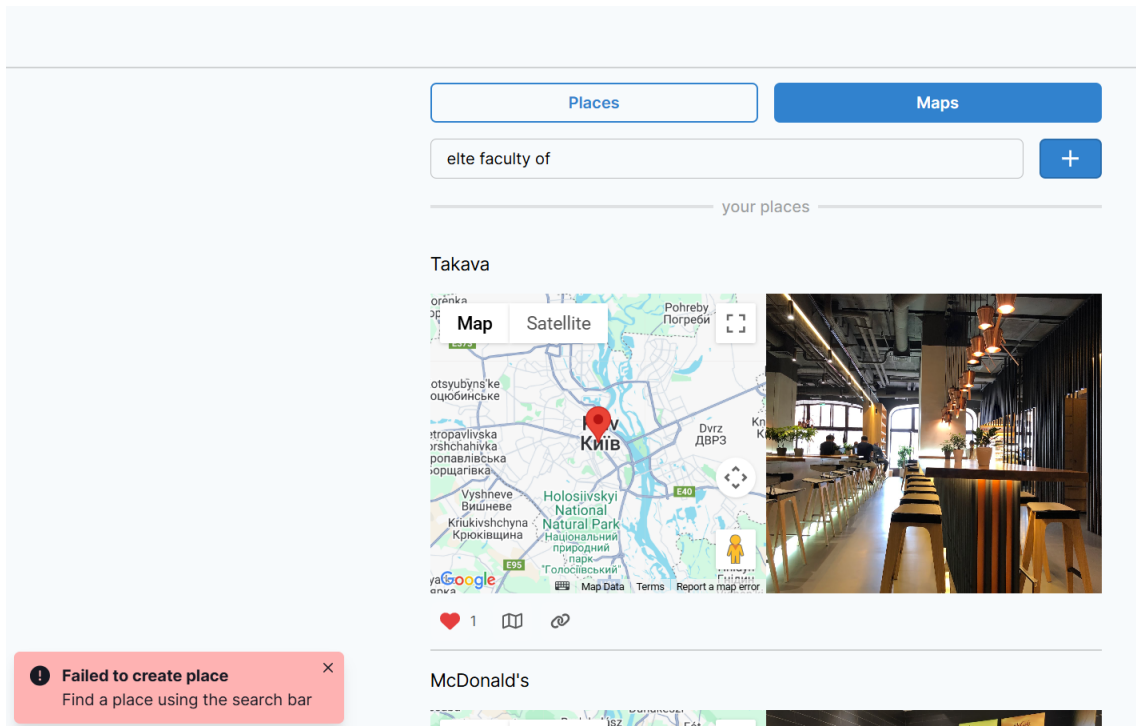


Figure 2.17: Create place error

- **No Place Selected:** If a user tries to create a place without selecting a place from the autocomplete list, the appropriate error message appears.
- **Empty Field:** If a user tries to create a place without searching for it, the appropriate error message appears.

Create Map

- **Empty Name or Description:** If the map name or description is missing, the appropriate field is highlighted.

The screenshot shows a web interface for creating a map. At the top, there are two tabs: 'Places' (active) and 'Maps'. Below the tabs is a text input field containing 'Summer 2025'. Underneath that is a larger text area for 'Map Description', which is highlighted with a red border, indicating it is required. Below the description field is a blue 'Create Map' button. At the bottom, there is a search bar with the placeholder text 'Search for a place' and a blue button with a plus sign. Below the search bar is a horizontal line with the text 'your maps' in the center.

Figure 2.18: Create map error (empty field)

- **Lack of Places:** If less than two places are added to the map, the appropriate error message appears.

This screenshot shows the same 'Create Map' form as Figure 2.18, but with a different state. The 'Map Description' field now contains the text 'I plan to take a nap on the beach'. Below the 'Create Map' button, there is a search bar and a plus button. Below the search bar is a horizontal line with the text 'your maps'. At the bottom of the form, there is a map preview titled 'Kyiv Universities' showing a map of Kyiv with several red location pins. To the right of the map is a thumbnail image of a building, identified as 'Kyiv National Economic University'. In the bottom left corner, there is a red error message box that says 'Failed to create map' and 'Map must have at least 2 places'.

Figure 2.19: Create map error (lack of places)

Chapter 3

Developer documentation

3.1 Functional requirements

This chapter provides all the necessary information about the Map Management Platform to developers [5]. It lists functional and non-functional requirements, provides a manual on how to set up the project, and describes in detail the tasks solved.

1. Signup

- User Story: As a new user, I want to create an account.
- Functional Flow: The user has to provide a full name, a unique username, and a password. The system validates the inputs, checks for username uniqueness, and creates an account.

2. Login

- User Story: As a user, I want to log in to my account.
- Functional Flow: The user has to provide the correct username and password.

3. Edit profile

- User Story: As a user, I want to edit my profile information.
- Functional Flow: After logging in, users can access the Edit Profile page and change their full name, username, or password. The system validates the changes and updates the profile information.

4. Logout

- User Story: As a user, I want to log out.
- Functional Flow: After a successful logout, the user is redirected to the Login page.

5. Delete account

- User Story: As a user, I want to delete my account.
- Functional Flow: After a successful deletion, the user is redirected to the Login page.

6. Explore users

- As a user, I want to explore users of the platform.
- Functional Flow: Users can view all users of the platform on the Explore page (Users section); search users by full name or username. The system provides a list of recommended users based on the user's preferences.

7. Explore places

- User Story: As a user, I want to explore places added to the platform.
- Functional Flow: Users can view all places added to the platform on the Explore page (Places section); search places by place name. The system provides a list of recommended places based on the user's preferences.

8. Explore maps

- User Story: As a user, I want to explore maps created on the platform.
- Functional Flow: Users can view all maps created on the platform on the Explore page (Maps section); search maps by map name or its description. The system provides a list of recommended maps based on the user's preferences.

9. Create place

- User Story: As a user, I want to add a new place to the platform and like it.
- Functional Flow: The user searches for a place by typing its name in the input field. The user has to select a place from the autocomplete list and submit the form.

10. Create map

- User Story: As a user, I want to create a map.
- Functional Flow: The user has to provides a map name, description, select at least two places, and submit the form.

3.2 Non-functional requirements

1. Performance

- All pages must load within 2 seconds under standard network conditions.
- Autocomplete for places must respond within 2 seconds after user input.
- Place card should be rendered within 2 seconds after submitting the create place form.

2. Usability

- The interface must be intuitive [6].
- Users should be able to navigate to the main features typically within three clicks.

3. Robustness

- Input validation should happen on both the frontend and the backend to ensure data security.
- Most errors should be handled, providing informative feedback to users.

4. Readability

- The source code should be well organised and consistently formatted.
- The codebase should have clear naming.

3.3 Windows navigation

The diagram shows every window of the application and how a user can navigate between them. Arrow colors correspond to the window category:

- **red:** authentication forms and header
- **lavender:** profile data lists
- **blue:** explore lists
- **green:** create forms
- **yellow:** place profile
- **orange:** map profile
- **black:** profile profile

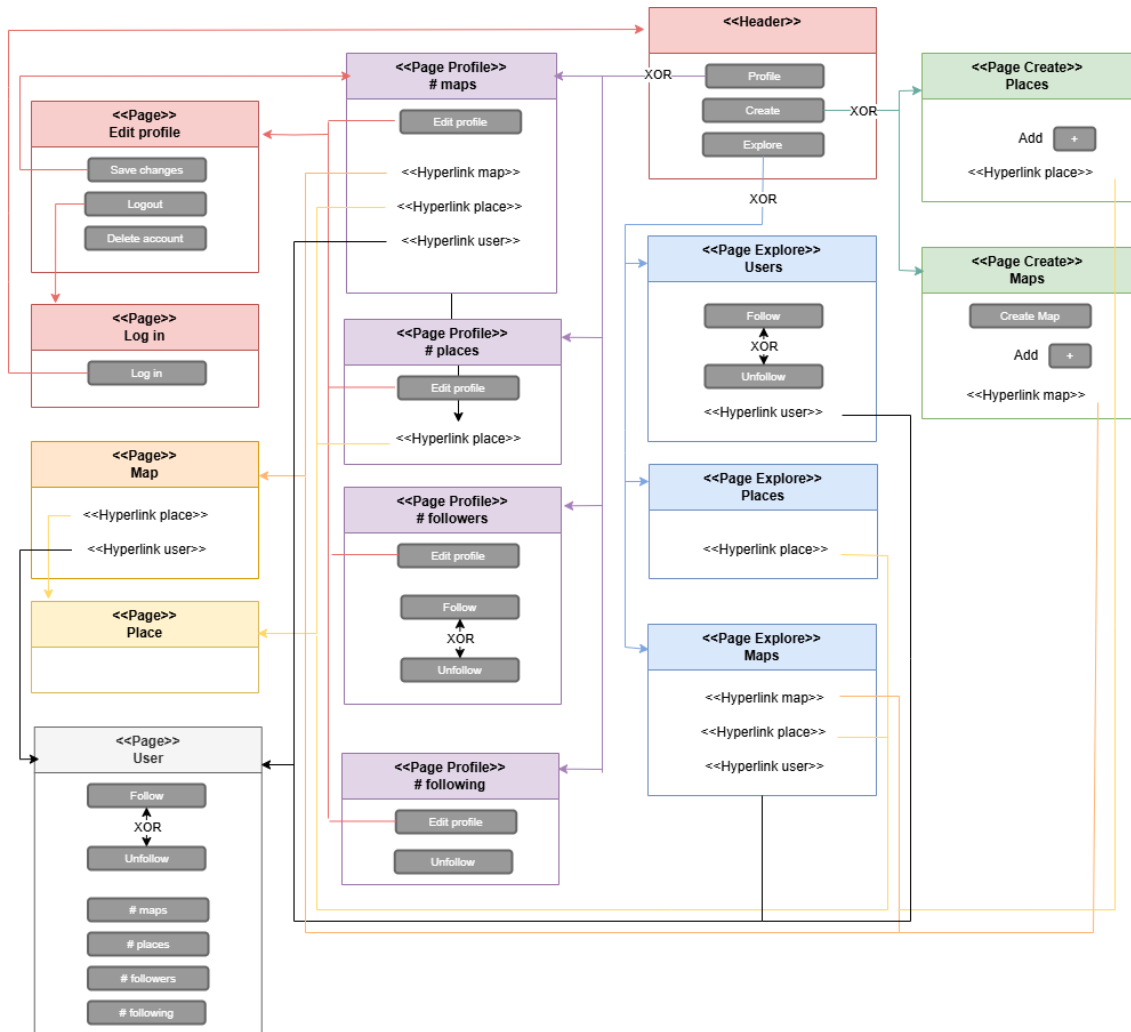


Figure 3.1: Windows navigation diagram

3.4 Functionality

3.4.1 Unauthorized user

The diagram illustrates the use cases available to a user who is not logged into the system. An unauthorized user has limited access to features, they can only explore public content.

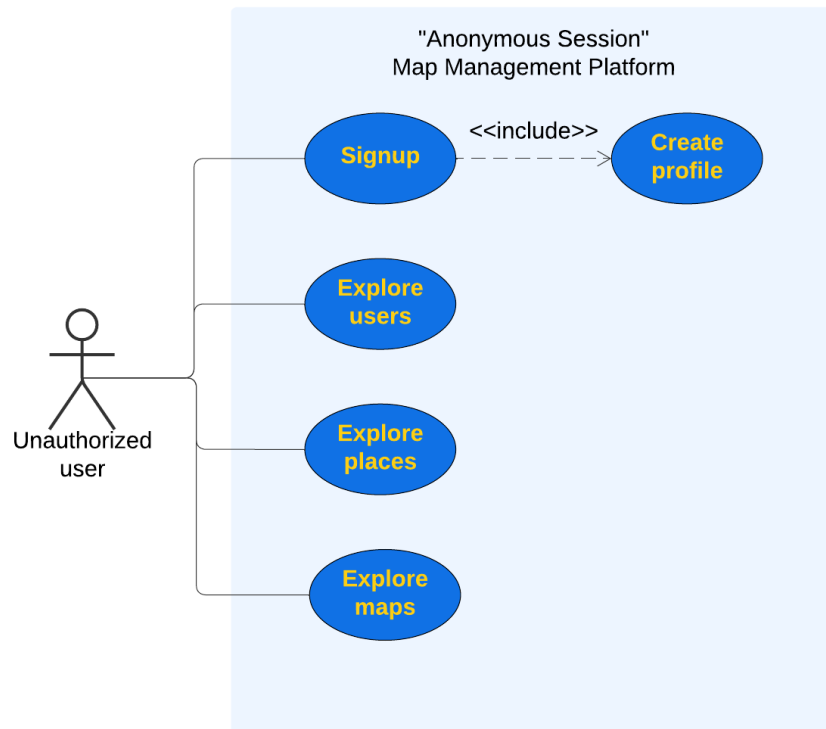


Figure 3.2: Use case diagram (unauthenticated user)

Available pages

- **Signup** – create a new account.
- **Explore** – search for users, maps, or places.
- **User** – view public profile data.
- **Map** – view map details and copy its link.
- **Place** – view place details, open it in Google Maps, or copy its link.

3.4.2 Authorized user

The diagram illustrates the use cases available to a user after successful authentication. An authorized user can update their profile data. They also can create maps and places.

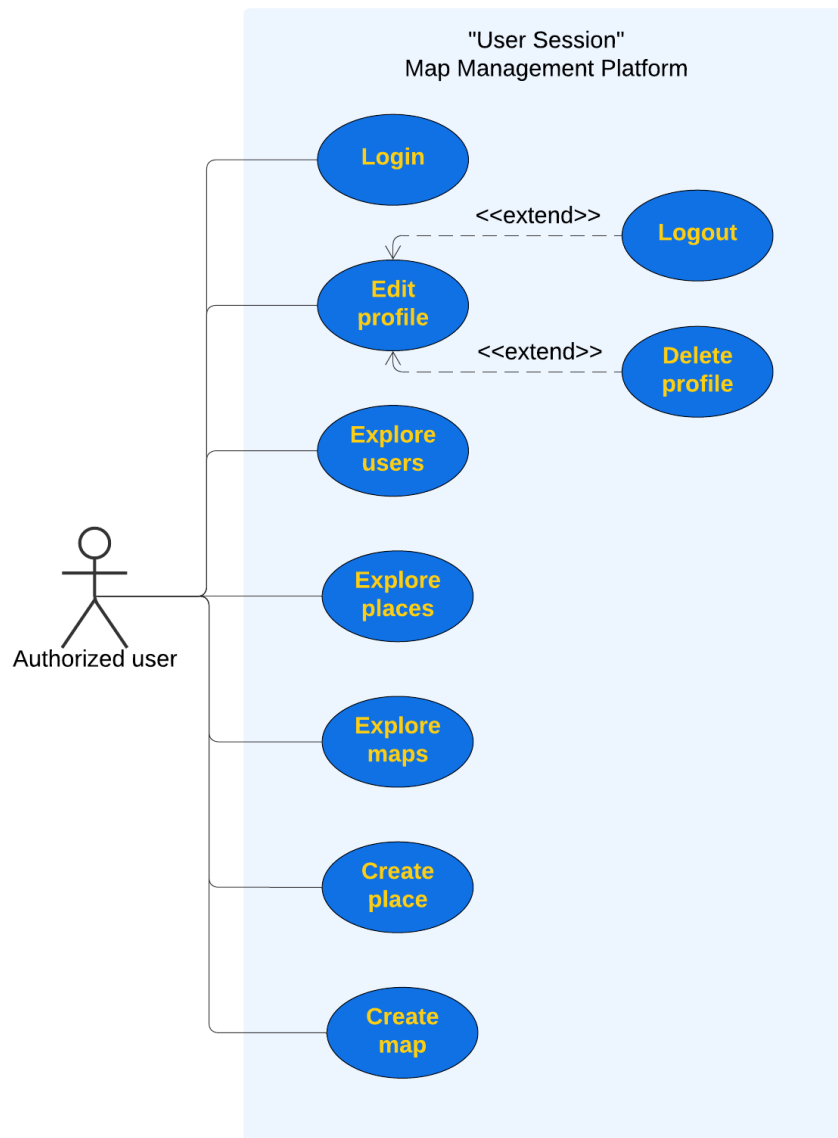


Figure 3.3: Use case diagram (authenticated user)

Available pages

- **Login** – log in to the system.
- **Edit Profile** – update personal data, log out, or delete the account.
- **Create** – create new places or maps.
- **Explore** – search for users, places or maps.
- **User** – view public profile data, follow/unfollow the user.
- **Map** – view map details and copy its link, like/unlike the map.
- **Place** – view place details, open it in Google Maps, or copy its link, like/unlike the place.

3.5 Technologies

Go is a statically typed, compiled programming language. It is used to implement the server-side business logic and APIs [7, 8].

Fiber is a lightweight web framework for Go, inspired by Express.js. It was used to build RESTful APIs [9, 10].

MongoDB is a NoSQL, document-oriented database used to store user profiles, places, and maps. Chosen for flexible schemas [11, 12].

React.js is a JavaScript library for building user interfaces [13, 14, 15, 16].

TypeScript is a typed superset of JavaScript. It adds static typing to React components and other front-end logic [17].

Chakra UI is a modular component library for React applications. Provided reusable components speed up interface development [18].

Google Maps Platform is a set of APIs that allow to retrieve data from Google Maps [19].

Zustand minimal state-management library for React.

Vite is a front-end build tool and development server. It offers instant hot module replacement and optimized build outputs.

3.6 Installation

This section describes the steps required to set up the development environment for the platform. The root directory is `Map-Management-Platform-for-Sharing-Travel-Experiences` and is split into a frontend (`client/`) and backend (`server/`).

3.6.1 Prerequisites

Node.js (version 18 or higher)

Go (version 1.21 or higher)

MongoDB (local instance or remote cluster)

3.6.2 Environment Variables

Copy the sample values below into a `.env` file in both subdirectories. The keys are ready to use for local testing, but for public deployment you should replace them with your own.

`client/.env`

- `VITE_GOOGLE_MAPS_API_KEY=AIzaSyCv320DiXyFsfhwz35paoAZRZFIUh4DCv4`
- `REACT_APP_BASE_URL=http://localhost:5000/api`

`server/.env`

- `PORT=5000`
- `MONGODB_URI=mongodb+srv://vnpnko:Qle4RC7Aone1ZVKY@cluster0.dnngs.mongodb.net/golang_db?retryWrites=true&w=majority&appName=Cluster0`

3.6.3 Frontend Installation

To set up and run the frontend:

1. Open a terminal window and navigate to the `client/` directory
 - `cd client`
2. Install the required dependencies
 - `npm install`
3. Start the development server
 - `npm run dev`
4. The frontend will be available at
 - `http://localhost:5173/`

3.6.4 Backend Installation

To set up and run the backend:

1. Open a new terminal window and navigate to the `server/` directory
 - `cd server`
2. Start the backend server
 - `go run main.go`
3. The backend server will listen on
 - `http://localhost:5000/api`

3.7 Architecture

The Map Management Platform consists of the Frontend, Backend, and Database layers. The system integrates with Google Maps API interacting through two paths:

- **Frontend** communicates directly with Google Maps for map rendering and place search [20].
- **Backend** acts as a proxy for fetching signed photo URLs and returning them to the client.

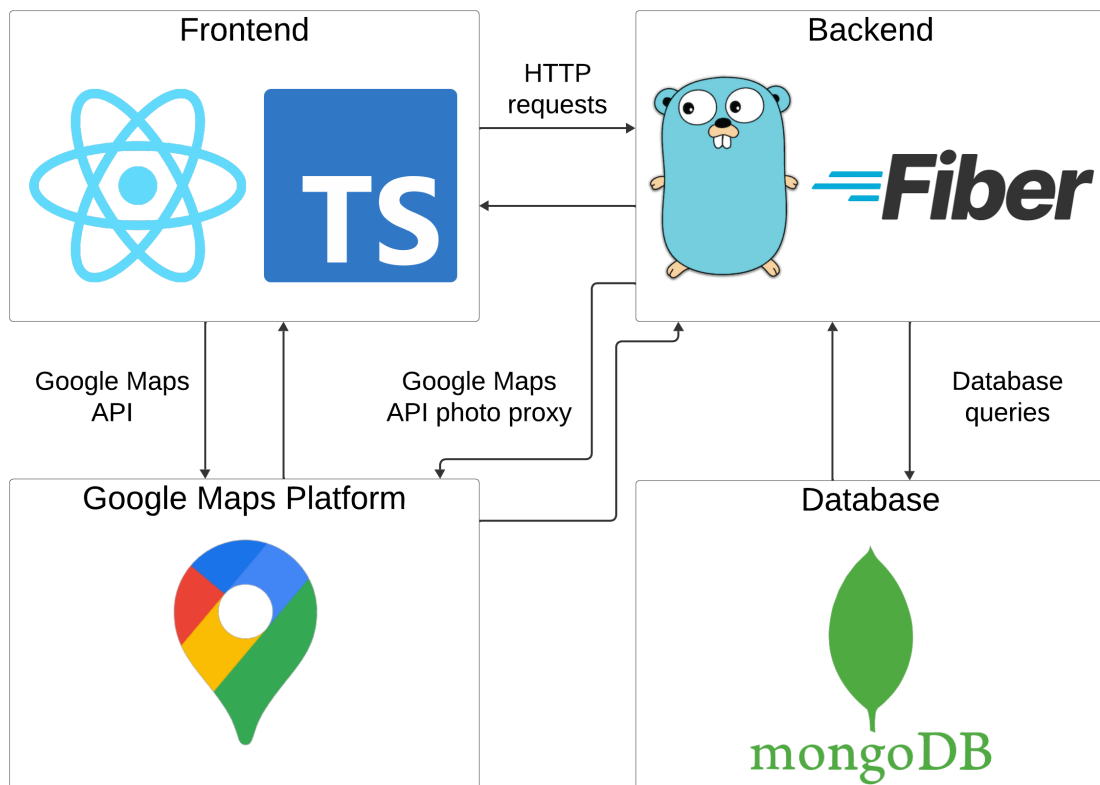


Figure 3.4: Architecture diagram

3.7.1 Frontend Layer

Quality Description

- Routing

- Navigation is handled by React Router.
- Based on the URL, a specific page component is loaded dynamically inside a shared Layout component.
- Pages that require Google Maps functionality are wrapped inside the GoogleMapsLoader component to ensure Maps APIs are properly initialized [21].

```
4
5 export const GOOGLE_MAPS_LIBRARIES: ("places" | "maps")[] = ["places", "maps"];
6
7 interface GoogleMapsLoaderProps {
8   children: React.ReactNode;
9 }
10
11 const GoogleMapsLoader: React.FC<GoogleMapsLoaderProps> = ({ children }) => {
12   const { isLoading, loadError } = useJsApiLoader({
13     googleMapsApiKey: import.meta.env.VITE_GOOGLE_MAPS_API_KEY,
14     libraries: GOOGLE_MAPS_LIBRARIES,
15   });
16
17   if (loadError) {
18     return <Text color="red.500">Error loading Google Maps API.</Text>;
19   }
20
21   if (!isLoading) {
22     return (
23       <Flex align="center" justify="center" minH="200px">
24         <Spinner size="lg" />
25       </Flex>
26     );
27   }
28
29   return <>{children}</>;
30 };
31
32 export default GoogleMapsLoader;
```

Figure 3.5: GoogleMapsLoader.tsx

- **Global State Management**

- Zustand is used for lightweight, reactive state management.
- `loggedInUserStore` and `mapDraftStore` store session data across page reloads.
- These stores are automatically persisted using `zustand/middleware` to local storage.

```
client > src > store > TS loggedInUserStore.ts > ...
1  import { create } from "zustand";
2  import { persist } from "zustand/middleware";
3  import { User } from "../models/User";
4  import { LoggedInUserStore } from "../models/LoggedInUserStore.ts";
5
6  export const loggedInUserStore = create<LoggedInUserStore>()(
7    persist(
8      (set) => ({
9        loggedInUser: null,
10       setLoggedInUser: (loggedInUser: User | null) => {
11         set({ loggedInUser });
12       },
13     }),
14     {
15       name: "loggedInUser",
16     },
17   ),
18 );
```

Figure 3.6: `LoggedInUserStore.ts`

- **Data Fetching and Mutations**

- API requests are performed using React Query, which handles loading states, error handling, and caching automatically.
- All API endpoints use the base URL `http://localhost:5000/api/` defined in a centralized location.

```

client > src > common > Place > hooks > TS useFetchPlace.ts > default
1  import { useQuery } from "@tanstack/react-query";
2  import { BASE_URL } from "../../App.tsx";
3  import { Place } from "../../models/Place.ts";
4
5  interface FetchPlacePayload {
6    placeId: string;
7  }
8
9  const useFetchPlace = ({ placeId }: FetchPlacePayload) => {
10    const { data, isLoading, error } = useQuery<Place>({
11      queryKey: ["place", placeId],
12      queryFn: async () => {
13        const response = await fetch(`${BASE_URL}/places/${placeId}`);
14        if (response.status === 404) {
15          return null;
16        }
17        if (!response.ok) {
18          const errorData = await response.json();
19          throw new Error(errorData.error || "Failed to fetch place");
20        }
21        return response.json();
22      },
23      enabled: !!placeId,
24      retry: false,
25      refetchOnWindowFocus: false,
26    });
27
28    return { place: data, isFetchingPlace: isLoading, placeError: error };
29  };
30
31  export default useFetchPlace;
32

```

Figure 3.7: useFetchPlace.ts

Quantity Description

Below is the description of every top-level directory in `client/`

- `assets/`
 - **logo:** logo icon of the project
- `common/`
 - **components:** generic components
 - **hooks:** generic hooks and custom toast
 - **layout:** layout and header components
 - **ui:** custom styled components

- **User:** user related components and hooks
- **Map:** map related components and hooks
- **Place:** place related and hooks
- **models/**
 - **LoggedInUserStore:** type describing a logged-in user.
 - **MapDraftStore:** type describing a map draft during its creation.
 - **User:** interface for user entity
 - **Map:** interface for map entity
 - **Place:** interface for place entity
 - **Location:** interface for location used by place interface
- **pages/**
 - **Login:** page components and hook for login
 - **Signup:** page components and hook for signup
 - **Profile:** page components and hooks that simplify the UI rendering of the page
 - **EditProfile:** page component and hooks for updating user data and deleting account
 - **Explore:** page components and hooks for recommendations
 - **Create:** page components and hooks for creation
 - **Place:** page component
 - **Map:** page component
- **store/**
 - **loggedInUserStore:** stores authenticated user data globally
 - **mapDraftStore:** stores current map draft data on the Create page

3.7.2 Backend Layer

Quality Description

- **Routing:** The router matches the request to the appropriate controller based on the URL and HTTP method.
- **Controller Execution:** The matched controller function processes the request, handling validation, business rules, and database interaction.
- **Database interaction:** The controller uses the MongoDB client to query or update database collections as needed.
- **Response formatting:** The final output is formatted into a standard JSON structure and returned to the client.

Quantity Description

Login Sequence Diagram illustrates the high-level login process.

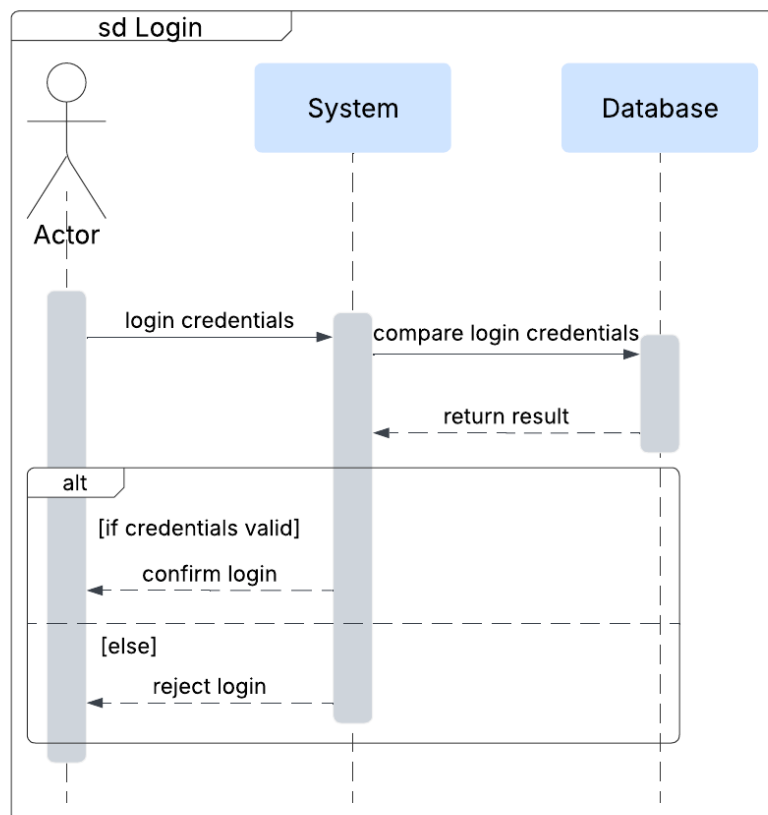


Figure 3.8: Login sequence diagram

1. The process begins when a user submits their login credentials (username and password) to the System.
2. The System forwards these credentials to the Database to verify whether they are correct.
3. The Database responds with a result indicating whether the credentials are valid or not.
4. The System then evaluates the result in **alt** (alternative) block:
 - **If the credentials are correct:**, the system confirms the login.
 - **Else:** the system rejects the login and returns an error message.

Create Place Sequence Diagram illustrates the full lifecycle of creating a place.

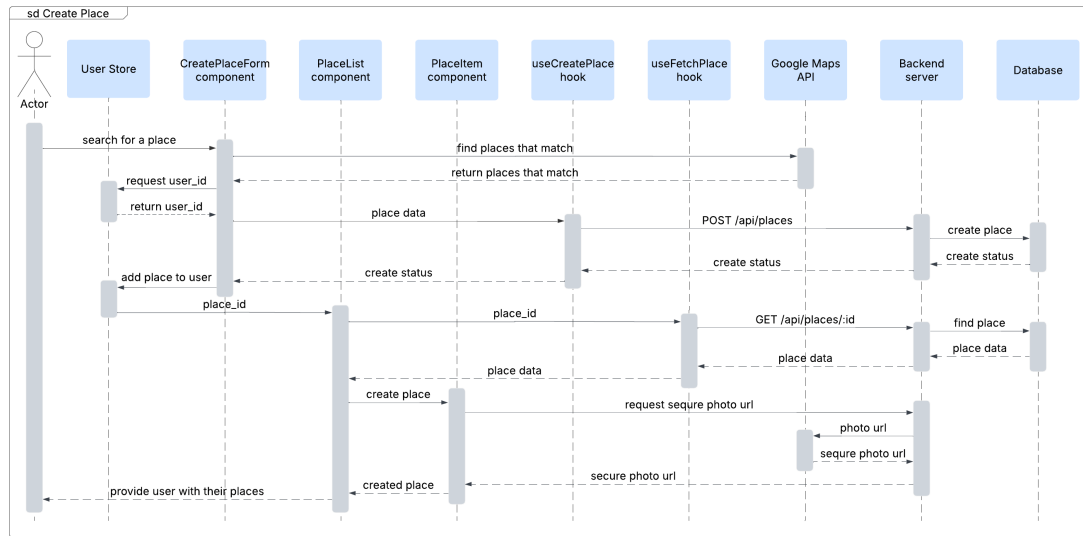


Figure 3.9: Create Place sequence diagram

1. The user initiates the process by typing a place name into the **CreatePlaceForm** component.
2. As the user types, the form sends requests to the Google Maps API to retrieve matching suggestions based on the input [22].
3. The API responds with a list of matching place results.
4. The user selects a place from the list.

5. The component retrieves `user_id` from the `User Store`.
6. The selected place data with the `user_id`, is passed to the `useCreatePlace` hook.
7. The hook sends a `POST` request to the backend.
8. The backend parses the request and forwards the data to the Database.
9. After successful insertion into the database, the backend responds with a status confirmation.
10. The hook receives the response and triggers a success callback.
11. `place_id` is added to the user's liked places.
12. This update in `User Store` triggers an automatic re-render of the `PlaceList` (displays the user's liked places) with appended new `PlaceItem`.
13. Before `PlaceItem` component is rendered, it makes a request to the backend to fetch a secure image URL via the Google Maps Photo Proxy. After the image URL is returned, the `PlaceItem` is rendered.

Create Map Sequence Diagram illustrates the full lifecycle of creating a map.

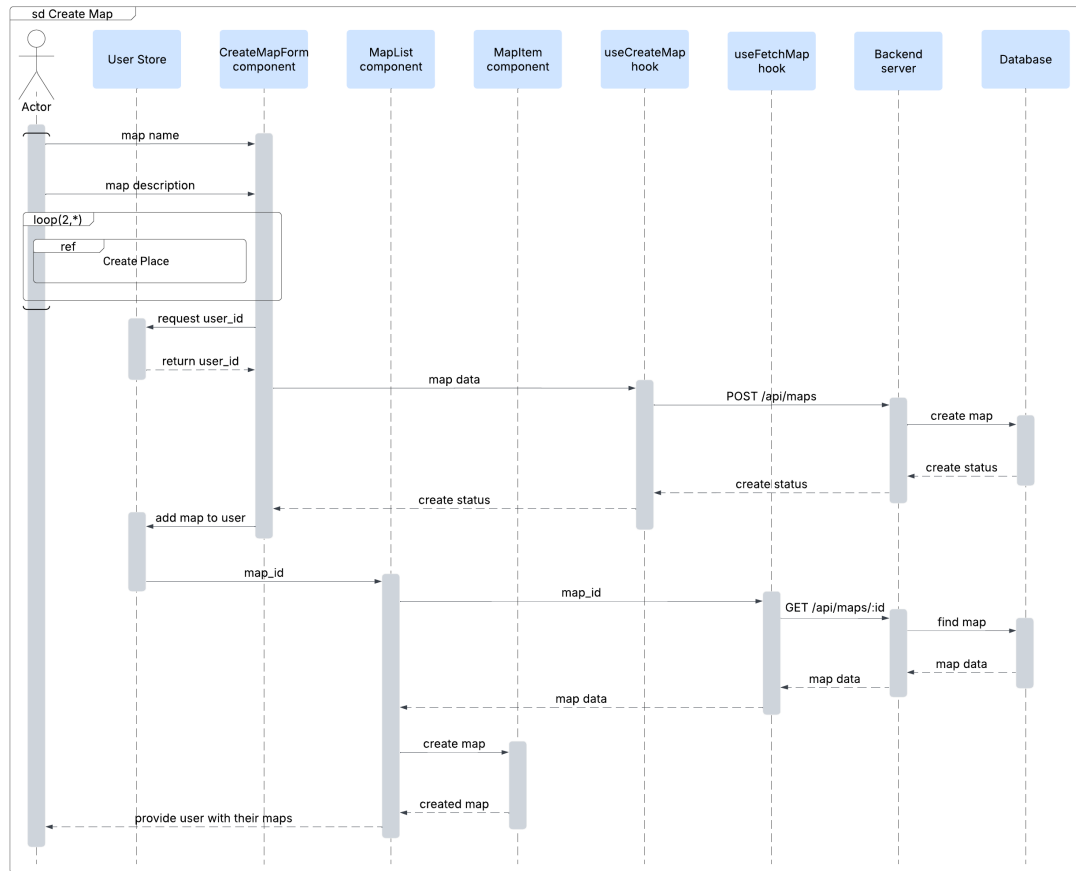


Figure 3.10: Create Map sequence diagram

1. A user begins by entering a `map name`, `description`, and adding at least two places (loop referencing the "Create Place" process) to the `CreateMapForm`.
2. The component retrieves the `user_id` from the `User Store`.
3. The map data with the `user_id` is passed to the `useCreateMap` hook.
4. The hook sends a `POST` request to the backend.
5. The backend parses the request and forwards the data to the `Database`.
6. After successful insertion into the database, the backend responds with a status confirmation.
7. The hook receives the response and triggers a success callback.
8. `map_id` is added to the user's liked maps.

9. This update in **User Store** triggers an automatic re-render of the **MapList** (displays the user's liked maps) with appended new **MapItem**.

Recommendation engine

The recommendation engine applies a lightweight, graph-based collaborative filtering approach to suggest new users, places, and maps to a viewer. For each candidate, it estimates the conditional probability that the viewer will be interested in that candidate, based on both social proximity and overall popularity.

The model is formulated as a probabilistic scoring function inspired by exponential-family distributions. Given a viewer v and a candidate c , the conditional probability $P(c | v)$ is computed as:

$$P(c | v) = \frac{\exp(\theta_1 \cdot \text{common_interest}(v, c) + \theta_2 \cdot \log(\text{global_interest}(c) + 1))}{\sum_{c' \in \text{Candidates}} \exp(\theta_1 \cdot \text{common_interest}(v, c') + \theta_2 \cdot \log(\text{global_interest}(c') + 1))} \quad (3.1)$$

- `common_interest(v, c)` indicates how many of the viewer's followed users also follow or like the candidate.
- `global_interest(c)` indicates the overall popularity of the candidate.
- θ_1 and θ_2 are weights that control the influence of social proximity and general popularity.
- The exponential function ensures that all scores are positive and enhances the differences between candidates before normalization.
- The denominator ensures normalization for a valid probability distribution.

Outline of the algorithm

1. Collect the set of users followed by the viewer.
2. Build a frequency map of secondary connections in one MongoDB aggregation.
3. For each candidate, compute `common_interest` and `global_interest`.
4. Apply the scoring formula and normalize across all candidates.
5. Select the top 3 candidates for each category (users, places, maps).

- **User recommendations**

- `common_interest`: number of the viewer's followed users who follow the candidate user.
- `global_interest`: total number of followers of the candidate user.

- **Place recommendations**

- `common_interest`: number of the viewer's followed users who liked the candidate place.
- `global_interest`: total number of likes of the candidate place.

- **Map recommendations**

- `common_interest`: number of the viewer's followed users who liked the candidate map.
- `global_interest`: total number of likes of the candidate map.

This probabilistic model provides a flexible and interpretable way to generate personalized recommendations by balancing social proximity and general popularity.

REST API overview

Every endpoint is rooted at `http://localhost:5000/api`

Authentication: `/auth`

Endpoint	Method	Description	Parameters	Returns
<code>/login</code>	Post	Login to the account	username, password	User object
<code>/signup</code>	Post	Register for an account	full name, username, password	User object

Table 3.1: Authentication endpoints

Users: /users

Endpoint	Method	Description	Params	Returns
/username/:username	Get	Get user by their username	-	User object
/id/:id	Get	Get user by their id	-	User object
/get/ids	Get	Get users by their full name or username	search query	Users ids
/recommended/:userId	Get	Get recommended users	-	Users ids
/:followerId/following/:followeeId	Post	Follow user	-	User object
/:followerId/following/:followeeId	Delete	Unfollow user	-	User object
/:id	Patch	Update user data	full name, username, password,	User object
/:id	Delete	Delete user	-	Success message

Table 3.2: User endpoints

Places: /places

Endpoint	Method	Description	Parameters	Returns
/id/:id	Get	Get places by their id	-	Place object
/get/ids	Get	Get places by their name	search query	Places ids
/recommended/:userId	Get	Get recommended places	-	Places ids
/	Post	Create place	name, url, likes, location, photoUrl	User object
/:id	Delete	Delete place	Success message	User object
/:placeId/likes/:userId	Post	Like place	-	User object
/:placeId/likes/:userId	Delete	Unlike place	-	User object

Table 3.3: Place endpoints

Maps: /maps

Endpoint	Method	Description	Parameters	Returns
/:id	Get	Get map by id	-	Map object
/get/ids	Get	Get maps by their name or description	search query	Maps ids
/recommended/:userId	Get	Get recommended maps	-	Maps ids
/	Post	Create map	name, description, places	Map object
/:id	Delete	Delete map	-	Success message
/:mapId/likes/:userId	Post	Like map	-	User object
/:mapId/likes/:userId	Delete	Unlike map	-	User object
/:mapId/places/:placeId	Post	Add place to map	-	Map object
/:mapId/places/:placeId	Delete	Remove place from map	-	Map object

Table 3.4: Map endpoints

3.7.3 Database Layer

Quality Description

- **Driver:** the server talks to MongoDB via the Go driver.
- **Collections:** there are three collections (`users`, `places`, `maps`). Each document keeps arrays of related document IDs, giving an efficient many-to-many model without join tables.
- **Consistency:** every action that touches more than one collection (e.g. like, follow, and map creation) opens a MongoDB session and executed inside one transaction. The entire batch is committed or rolled back together, so the data can never be left in a half-updated state.
- **Indexes:** `username`, `place.name`, `map.name`, and `map.description` (text) indexed to keep search queries fast.

- **Pagination:** endpoints that return lists range pagination, which scales linearly with collection size and avoids performance issues.

Quantity Description

Collection	Attributes	Relationships
User	<code>_id</code> , <code>fullname</code> , <code>username</code> , <code>password</code> , <code>followers</code> , <code>following</code> , <code>places</code> , <code>maps</code>	Links to Places and Maps. Connected to other users.
Map	<code>_id</code> , <code>name</code> , <code>description</code> , <code>places</code> , <code>likes</code> , <code>creatorUsername</code>	Linked from Users. Links to Places.
Place	<code>_id</code> , <code>name</code> , <code>url</code> , <code>likes</code> , <code>location</code> , <code>photoUrl</code>	Linked from Users and Maps.

Table 3.5: Collections

Entity Relationship Diagram

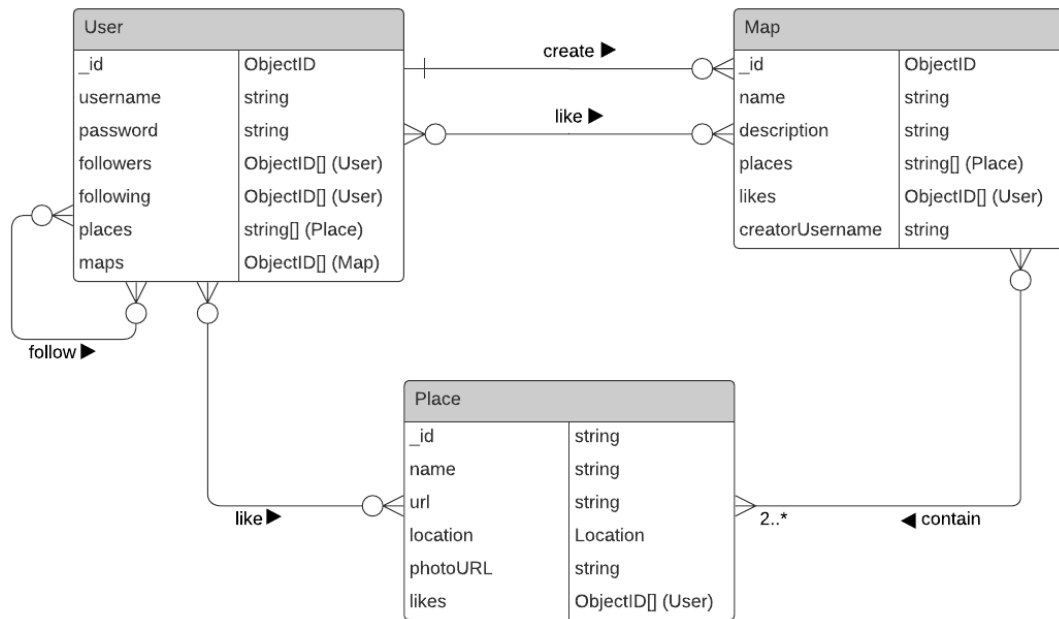


Figure 3.11: Entity relationship diagram

• User

- Represents a registered user on the platform.
- Can follow many users.
- Can be followed by many users.
- Can like many places.
- Can like many maps.
- Can create many maps.

• Map

- Represents a map created on the platform.
- Can contain at least two places.
- Can be created by one user.
- Can be liked by many users.

- **Place**

- Represents a place added to the platform.
- Can be liked by many users.
- Can be used in many maps.

3.8 Testing

This section is dedicated to manual testing of the Map Management Platform. These tests are designed to verify that all the features and functionalities of the project work as expected.

1. Test Case: Signup with valid credentials

- Test Steps:
 - (a) Navigate to the Signup Page.
 - (b) Enter valid full name, username, and password.
 - (c) Submit the form.
- Expected Results: The system creates a new user. User is redirected to the profile page.

2. Test Case: Signup with incomplete credentials

- Test Steps:
 - (a) Navigate to the Signup Page.
 - (b) Miss a full name, username, or password field empty.
 - (c) Submit the form.
- Expected Results: The missed field is highlighted, Message saying about the missed field is shown.

3. Test Case: Signup with a taken name

- Test Steps:
 - (a) Navigate to the Signup Page.
 - (b) Enter valid full name, and password, but use already taken username.
 - (c) Submit the form.
- Expected Results: The username field is highlighted. The message saying the username is taken is shown.

4. Test Case: Login with valid credentials

- Test Steps:
 - (a) Navigate to the Login Page.
 - (b) Enter valid username and password.
 - (c) Submit the form.
- Expected Results: User is redirected to the profile page.

5. Test Case: Login with incomplete credentials

- Test Steps:
 - (a) Navigate to the Login Page.
 - (b) Miss a username or password field empty.
 - (c) Submit the form.
- Expected Results: The missed field is highlighted, Message saying about the missed field is shown.

6. Test Case: Login with invalid credentials

- Test Steps:
 - (a) Navigate to the Login Page.
 - (b) Enter invalid username and/or password.
 - (c) Submit the form.
- Expected Results: Message saying about the invalid credentials is shown.

7. Test Case: Follow a user being logged-in

- Test Steps:
 - (a) Navigate to a user's profile page.
 - (b) Click the follow button.
- Expected Results: Follow button is changed to Unfollow button, profile's owner added to the followed users of the logged-in user. the logged-in user is added to the followers of the profile's owner.

8. Test Case: Follow a user being not logged-in

- Test Steps:
 - (a) Navigate to a user's profile page.
 - (b) Click the follow button.
- Expected Results: Message saying that the follow action was failed is shown.

9. Test Case: Like a place being logged-in

- Test Steps:
 - (a) Navigate to a place's profile page.
 - (b) Click the empty heart icon.
- Expected Results: Empty heart icon is changed to filled heart. The place is added to the places of the user. The user is added to the likes of the place.

10. Test Case: Like a place being not logged-in

- Test Steps:
 - (a) Navigate to a place's profile page.
 - (b) Click the empty heart icon.
- Expected Results: Message saying that the like action was failed is shown.

11. Test Case: Create a place with following the instructions

- Test Steps:
 - (a) Navigate to Create page (places).
 - (b) Start typing a name of the place to the search field.
 - (c) Select a recommendation from the autocomplete list.
 - (d) Click the "+" button.
- Expected Results: The place is added to the liked places of the user. The user is added to the likes of the place. The place appears on top of the liked places list.

12. Test Case: Create a place without selecting a place from the autocomplete list

- Test Steps:
 - (a) Navigate to Create page (places).
 - (b) Start typing a name of the place to the search field.
 - (c) Click the "+" button.
- Expected Results: Message saying that the place creation was failed is shown.

13. Test Case: Create a map with following the instructions

- Test Steps:
 - (a) Navigate to Create page (maps).
 - (b) Enter a name for the map.
 - (c) Enter a description for the map.
 - (d) Add at least two places using place creation form.
 - (e) Click the "Create Map" button.
- Expected Results: The map is created and added to the liked maps of the user. The user is added to the likes of the map. The map appears on top of the liked maps list.

14. Test Case: Create a map without providing a name to it

- Test Steps:
 - (a) Navigate to Create page (maps).
 - (b) Leave the name field empty.
 - (c) Enter a description for the map.
 - (d) Add at least two places using place creation form.
 - (e) Click the "Create Map" button.
- Expected Results: The name field is highlighted.

15. Test Case: Create a map without providing a description to it

- Test Steps:
 - (a) Navigate to Create page (maps).
 - (b) Enter a name for the map.
 - (c) Leave the description field empty.
 - (d) Add at least two places using place creation form.
 - (e) Click the "Create Map" button.
- Expected Results: The description field is highlighted.

16. Test Case: Create a map providing with less than two places

- Test Steps:
 - (a) Navigate to Create page (maps).
 - (b) Enter a name for the map.
 - (c) Enter a description for the map.
 - (d) Add less than two places using place creation form.
 - (e) Click the "Create Map" button.
- Expected Results: Message saying that the map creation was failed: map should contain at least two places is shown.

Chapter 4

Conclusion

Travellers now have one web application where they can keep every place they like, view them on an interactive map while planning, and share the finished map with others in one click. The Map Management Platform is a tool for storing, viewing, and sharing collections of favorite places. By combining interactive maps with social networking features, the platform offers a solution to the chaotic traditional note-taking practice.

The system was implemented using React.js with TypeScript and Chakra UI for the frontend, a RESTful API developed in Go using Fiber for the backend, and MongoDB for storing users, places, and maps. Google Maps Platform was integrated to support place search and map rendering.

Key features of the application include user profiles with public collections, the ability to create and share maps of favorite places. The recommendation engine uses a probabilistic model based on social proximity and global popularity to suggest relevant users, places, and maps.

In conclusion, the Map Management Platform successfully addresses the gap in existing travel planning tools by merging mapping functionality with social interactions. It offers a structured, shareable, and visually engaging way to document and explore travel experiences.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Horváth Győző, for his continuous support, expert guidance, and insightful feedback throughout the entire development of this thesis. His thoughtful suggestions and constructive criticism have significantly improved the quality of my work. I am truly thankful for his kind and encouraging attitude during every stage of the process.

I would also like to thank my family and friends for their support during this journey. Their encouragement helped me stay focused and motivated through the challenges.

Bibliography

- [1] M. Kang and M. A. Schuett. “Determinants of Sharing Travel Experiences in Social Media”. In: *Journal of Travel & Tourism Marketing* 30.1–2 (Jan. 2013), pp. 93–107. URL: <https://www.tandfonline.com/doi/abs/10.1080/10548408.2013.751237>.
- [2] C. Ip, H. A. Lee, and R. Law. “Profiling the Users of Travel Websites for Planning and Online Experience Sharing”. In: *Journal of Hospitality & Tourism Research* 36.3 (Nov. 2010), pp. 418–426. URL: <https://journals.sagepub.com/doi/10.1177/1096348010388663>.
- [3] V. Prasad et al. “Seamless Travel Planner: Turning Miles into Memories with Smart Navigation”. In: *International Journal for Research in Applied Science and Engineering Technology* 13.3 (Mar. 2025), pp. 1949–1952. URL: <https://www.ijraset.com/best-journal/seamless-travel-planner-turning-miles-into-memories-with-smart-navigation>.
- [4] R. E. Roth. “Interactivity and Cartography: A Contemporary Perspective on UI and UX Design from Geospatial Professionals”. In: *Cartographica* 50.2 (June 2015), pp. 94–115. URL: <https://utppublishing.com/doi/10.3138/cart.50.2.2427>.
- [5] C. Ravi and M. K. S. *Web Programming*. MTS Publications, 2023. URL: <https://www.magesticts.com/books/web-programming/>.
- [6] E. L.-C. Law, P. van Schaik, and V. Roto. “Attitudes Towards User Experience (UX) Measurement”. In: *International Journal of Human-Computer Studies* 72.6 (June 2014), pp. 526–541. URL: <https://www.sciencedirect.com/science/article/abs/pii/S1071581913001304?via%3Dihub>.
- [7] The Go Authors. *Go Documentation*. Google LLC, 2025. URL: <https://go.dev/doc>.

- [8] A. A. A. Donovan and B. W. Kernighan. *The Go Programming Language*. Boston, MA: Addison-Wesley, 2016. URL: <http://www.cs.uniroma2.it/upload/2017/TSC/The%20Go%20Programming%20Language.pdf>.
- [9] Fiber Contributors. *Fiber Documentation*. Gofiber, 2025. URL: <https://docs.gofiber.io>.
- [10] A. Neumann, N. Laranjeiro, and J. Bernardino. “An Analysis of Public REST Web Service APIs”. In: *IEEE Transactions on Services Computing* (2018), pp. 1–1. URL: <https://ieeexplore.ieee.org/document/8385157>.
- [11] MongoDB Inc. *MongoDB Documentation*. MongoDB Inc., 2025. URL: <https://mongodb.com/docs>.
- [12] R. Lawrence. “Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB”. In: *2014 International Conference on Computational Science and Computational Intelligence*. Mar. 2014. URL: <https://ieeexplore.ieee.org/document/6822123>.
- [13] Meta Open Source. *React: A JavaScript Library for Building UIs*. Meta Platforms, Inc., 2025. URL: <https://react.dev>.
- [14] A. Karić and N. Durmić. “Comparison of JavaScript Frontend Frameworks – Angular, React, and Vue”. In: *International Journal of Innovative Science and Research Technology* (July 2024), pp. 1383–1390. URL: <https://www.ijisrt.com/comparison-of-javascript-frontend-frameworks-angular-react-and-vue>.
- [15] A. Shukla. “Modern JavaScript Frameworks and JavaScript’s Future as a Full-Stack Programming Language”. In: *Journal of Artificial Intelligence & Cloud Computing* 2.4 (Dec. 2023), pp. 1–5. URL: <https://www.onlinescientificresearch.com/articles/modern-javascript-frameworks-and-javascripts-future-as-a-full-stack-programming-language.pdf>.
- [16] React Icons Community. *React Icons Documentation*. React Icons Project, 2025. URL: <https://react-icons.github.io/react-icons>.
- [17] TypeScript Team. *TypeScript Documentation*. 2025. URL: <https://www.typescriptlang.org>.

- [18] Chakra UI Team. *Chakra UI Documentation*. Chakra UI, 2025. URL: <https://v2.chakra-ui.com>.
- [19] Google Cloud Platform. *Google Maps Platform Documentation*. Google LLC, 2025. URL: <https://developers.google.com/maps/documentation>.
- [20] J. Xia et al. “Web-Based Mapping and Visualization Packages”. In: *Open GIS*. Springer, 2024, pp. 283–314. URL: https://link.springer.com/chapter/10.1007/978-3-031-41748-1_11.
- [21] S. Peltonen, L. Mezzalana, and D. Taibi. “Motivations, Benefits, and Issues for Adopting Micro-Frontends: A Multivocal Literature Review”. In: *Information and Software Technology* 136 (Aug. 2021), p. 106571. URL: <https://www.sciencedirect.com/science/article/pii/S0950584921000549?via%3Dihub>.
- [22] G. Svennerberg. *Beginning Google Maps API 3*. Berkeley, CA: Apress, 2010. URL: <https://link.springer.com/book/10.1007/978-1-4302-2803-5>.

List of Figures

2.1	Signup page	6
2.2	Login page	7
2.3	Profile page	8
2.4	Edit profile page	9
2.5	Explore page (explore users)	10
2.6	Explore page (explore places)	10
2.7	Explore page (explore maps)	11
2.8	Create page (create place)	11
2.9	Create page (create maps)	12
2.10	User page	13
2.11	Place page	13
2.12	Map page	14
2.13	Signup error (empty field)	15
2.14	Signup error (username uniqueness)	15
2.15	Login error (empty field)	16
2.16	Login error (invalid credentials)	16
2.17	Create place error	17
2.18	Create map error (empty field)	18
2.19	Create map error (lack of places)	18
3.1	Windows navigation diagram	22
3.2	Use case diagram (unauthenticated user)	23
3.3	Use case diagram (authenticated user)	24
3.4	Architecture diagram	28
3.5	GoogleMapsLoader.tsx	29
3.6	LoggedInUserStore.ts	30
3.7	useFetchPlace.ts	31
3.8	Login sequence diagram	33

3.9	Create Place sequence diagram	34
3.10	Create Map sequence diagram	36
3.11	Entity relationship diagram	42

List of Tables

3.1	Authentication endpoints	38
3.2	User endpoints	39
3.3	Place endpoints	39
3.4	Map endpoints	40
3.5	Collections	41