Design Document Scenic Route

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Student: Edward Omorusi
Student ID:C00258296
Supervisor: Paul Barry
Date: 14/02/2025

Table of Contents

Introduction

This design document aims to provide a detailed description of the features, technologies and overall objectives of the Scenic Route Application. The main purpose of this application is to offer users recommendations for scenic travel routes, allowing them to explore interesting paths as they journey from one location to another such as landscape, landmarks, monuments etc.

The objectives and goals of this project is to ensure a thorough understanding of the vision behind the application, this document will include a good understanding of the technologies being used and screens representing the overview design of the application.

Application Model

The Scenic Route Application called Scenic Path is a web-based navigation tool that helps users to find and visualise scenic routes between different locations. The web-navigation integrates OpenStreetMap (OSM) with Leaflet.js and OpenRouteService API to provide routes.

Figure 1: Application Logo [2]

Application Overview

Objective: This helps users in finding popular scenic routes between two locations.

Users: Travelers, cyclists, and drivers looking for detailed route guidance.

- **Key Features**:
    - Search for locations using **autocomplete** input fields.
    - Display multiple **possible routes** on the map.
    - Mark **towns along the route** with pins.
    - Fetch country lists dynamically for user convenience.
    - Provide interactive route options for better planning.

LOGIN / CREATE ACCOUNT

Step 1:

- The user logs into their account by entering their email and password on the **Login Screen**.

Step 2:

- If the user does not have an account, they can select the **Create Account** option.
- The app will prompt the user to fill in details such as their name, surname, address, and other relevant information.

NAVIGATION SEQUENCE

Step 1:

- After successfully logging in, the user is taken to the **Home Screen**, where a personalised greeting such as "Welcome Back, Ben!" is displayed.

Step 2:

- Users can navigate between different screens using the **navigation tab** located at the bottom of the application.

Step 3:

- When the user selects the **Map Screen**, a map showing their current location is displayed along with a **Search Destination** option on the screen.

MAPPING IN ACTION

Step 1:

- The user will type the destination they would like to travel using the search option.
- After selecting a destination, the app displays a route to the destination and a list of suggested Points of Interest (POIs).
- The application allows the user to set a maximum distance (in km) within which they want to see POIs. This will define how far away the recommended POIs will be based on the user's selected range.
- Once the user selects the main route, the app will check for POIs within the specified distance (e.g., 5 km). If there are no POIs within this range, the app will not display any results. However, as the user increases the range, the app will expand the search to include more POIs.
- The app searches for POIs based on predefined categories, such as **natural attractions**, **cultural sites**, **dining**, and more, ensuring relevant suggestions for the user along their route.
- With that information of the user the application will create a query based on the user preferences.
- The query is sent to the API to fetch information of POI data (location , name , type of activity etc) based near or destination route

- The application will search these POIs by using an external API (OpenStreetMap or Yelp API), and this will display information of the POIs available, such as the time that they close and photos of the place so the user can have a look at how the different POIs near their route look.
- The user can browse through these POIs and select the ones they are interested in visiting along the route.

Step 2:

- When a user selects one or more POIs, the app recalculates the route, updating the travel time based on the number and distance of the selected POIs from the main destination.
- The updated map is displayed for the user to review and confirm or make further adjustments.

- After that, the user will be able to see the updated route with their POIs of interest on the way to their main destination.

SAVE ROUTE

Step 1:

- After using a scenic route, the user can save it by pressing the **heart icon** displayed beside the route's details.

Step 2:

- Once the user saves the route, a confirmation message, such as "Route Saved," is displayed on the screen.

Step 3:

- Saved routes can be accessed by selecting the **Saved Routes tab** from the navigation bar.
- When the user opens the **Saved Routes Screen**, a list of all the previously saved routes is displayed

Technologies

Frontend

HTML

HTML (HyperText Markup Language)[5] will serve as the core structure of the web application. It is used for creating the web pages and defining the layout of the application, such as input fields, buttons, and other interactive elements.

CSS

CSS (Cascading Style Sheets) [6] will be used to style the application, making it visually appealing and user-friendly. It will manage the design aspects, such as colours, fonts, layout, and responsive behaviour.

JavaScript

JavaScript[7] will handle the dynamic and interactive aspects of the application, such as interacting with the map, fetching data from APIs, and updating the interface based on user input.

Leaflet.js

Leaflet [9] is an open-source JavaScript library for building interactive maps. It will be used for rendering maps and showing routes. The combination of Leaflet.js with OpenStreetMap provides an affordable and feature-rich solution for map rendering.

Mapping API

OpenStreetMap (OSM)

OSM [10] is a free, open-source mapping platform that provides detailed and community-driven geographic data. It is particularly suitable for scenic route applications because it offers detailed coverage of rural and remote areas that may not be well-documented by commercial APIs. Developers can leverage OSM’s API along with tools like Leaflet.js to render maps and build custom navigation features.

Backend

Flask

The Flask-based[8] backend for the Scenic Route Application handles key tasks such as fetching scenic route data, points of interest (POIs), and managing user interactions with the system. It integrates with multiple external APIs, including OpenRouteService for route data and Yelp for POI information, and uses MongoDB for data storage.

Database

MongoDB

The MongoDB [4] database for the **Scenic Route Application** is responsible for storing and managing user data, saved routes, and points of interest (POIs). As a **NoSQL database**, MongoDB provides flexibility in handling geospatial data, making it well-suited for mapping applications.
It integrates with the **Flask backend** to efficiently store and retrieve route details, user preferences, and POI information fetched from external APIs like **OpenRouteService** and

Yelp. MongoDB's document-based structure allows for **fast queries** and **efficient data storage**, ensuring a seamless user experience when searching for scenic routes and POIs.

Sequence Diagram

This sequence Diagram represents how the application will be working.

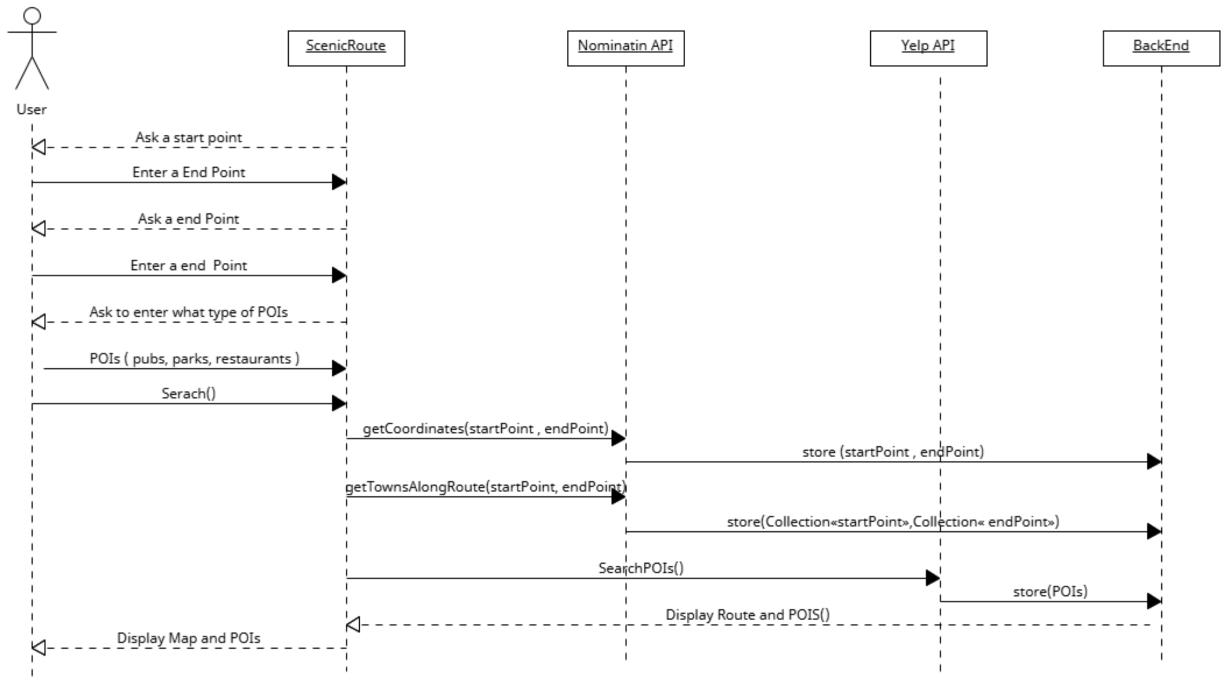

User
ScenicRoute
Nominatin API
Yelp API
BackEnd
Ask a start point
Enter a End Point
Ask a end Point
Enter a end Point
Ask to enter what type of POIs
POIs (pubs, parks, restaurants)
Serach()
getCoordinates(startPoint , endPoint)
store (startPoint , endPoint)
getTownsAlongRoute(startPoint, endPoint)
store(Collection«startPoint»,Collection« endPoint»)
SearchPOIs()
store(POIs)
Display Route and POIS()
Display Map and POIs

Figure 2: Main Functionality Diagram [3]

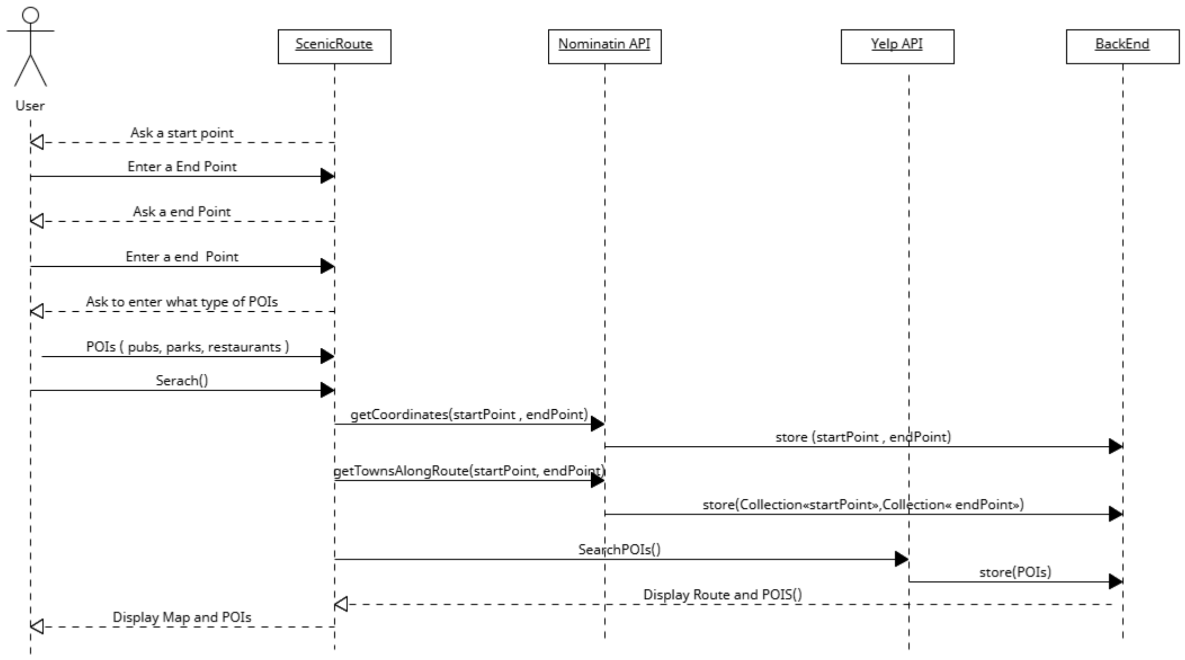

User
ScenicRoute
Nominatin API
Yelp API
BackEnd
Ask a start point
Enter a End Point
Ask a end Point
Enter a end Point
Ask to enter what type of POIs
POIs (pubs, parks, restaurants)
Serach()
getCoordinates(startPoint , endPoint)
store (startPoint , endPoint)
getTownsAlongRoute(startPoint, endPoint)
store(Collection«startPoint»,Collection« endPoint»)
SearchPOIs()
store(POIs)
Display Route and POIS()
Display Map and POIs

Figure 3: Save Routes and POIs Diagram [3]

Database Layout

This is how the MongoDB[4] will be able to store users information POIs and save routes that they have liked.

Users Collection

Routes Collection

```
{
  "_id": ObjectId,
  "user_id": ObjectId,  // References `users` collection
  "start_location": {
    "name": "New York",
    "coordinates": [40.7128, -74.0060]
  },
  "end_location": {
    "name": "Los Angeles",
    "coordinates": [34.0522, -118.2437]
  },
  "route_data": { /* JSON from OpenRouteService */ },
  "pois": [ObjectId]  // References `pois` collection
}
```

POIs Collection

```
{
  "_id": ObjectId,
  "route_id": ObjectId,  // References `routes` collection
  "name": "Central Park",
  "category": "Park",
  "coordinates": [40.785091, -73.968285],
  "rating": 4.5,
  "yelp_id": "abcd1234"
}
```

Screens

Sign-in Screen

Users can create an account by providing their **name, email, phone number, address, username, and password**.

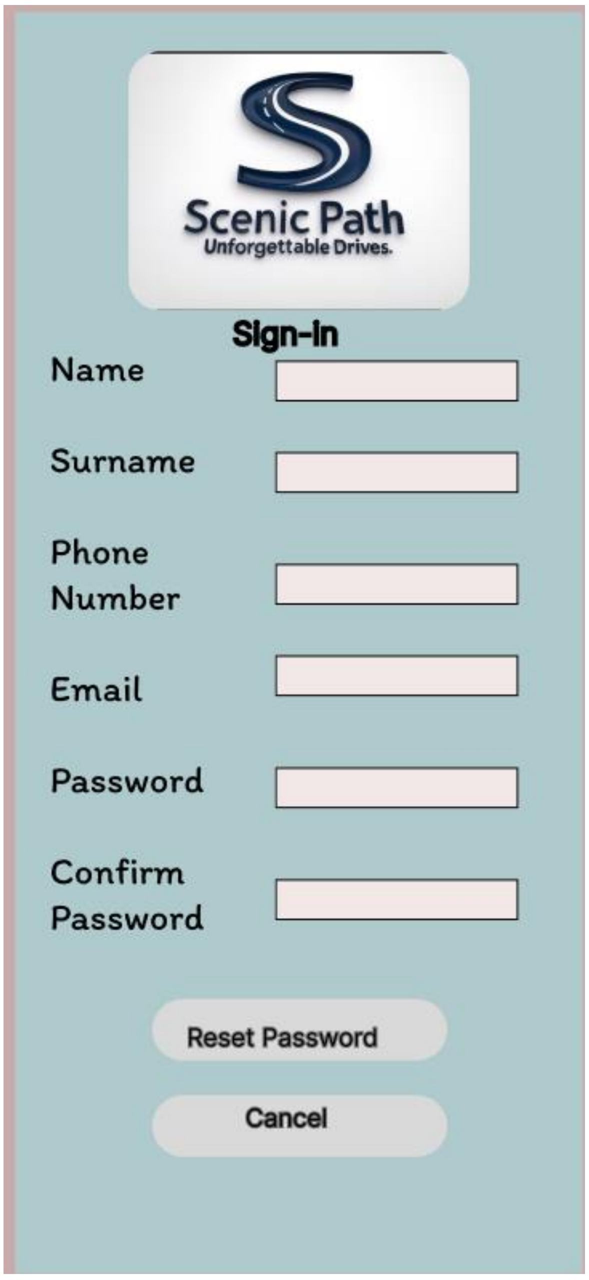


Figure 4: Sign-in Screen [1]

Login Screen

Users can log in using their **username and password**.

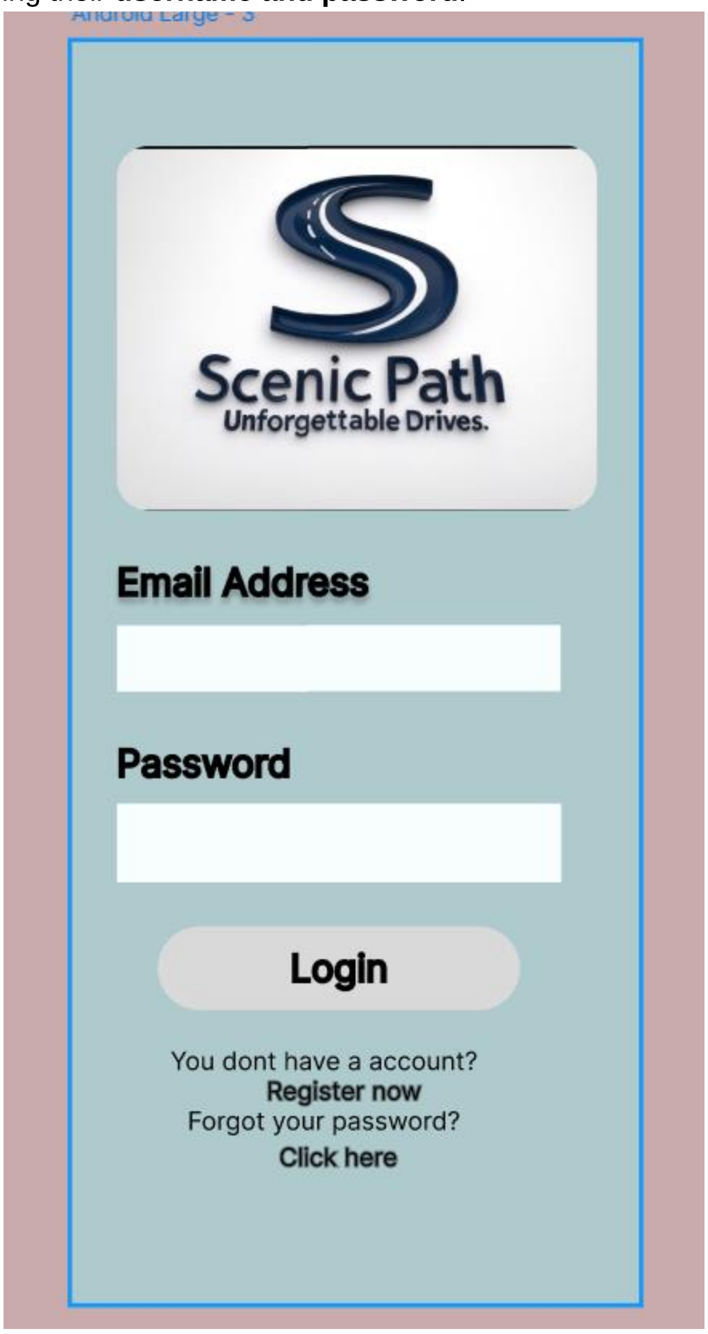


Figure 5: Login Screen[1]

Forgot Password Screen

If a user forgets their password, they can reset it by verifying their **email address** and following the password reset instructions.

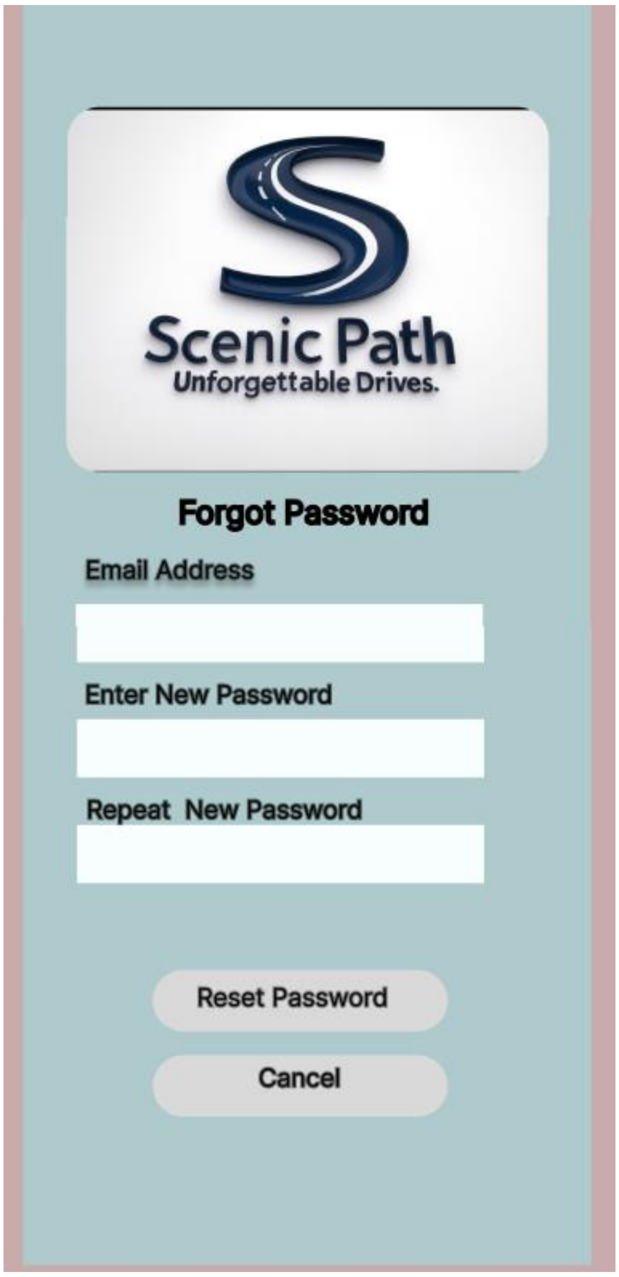


Figure 6: Forgot Password [1]

Home Screen

The first screen displayed after login, providing quick access to navigation, saved routes, and user settings.

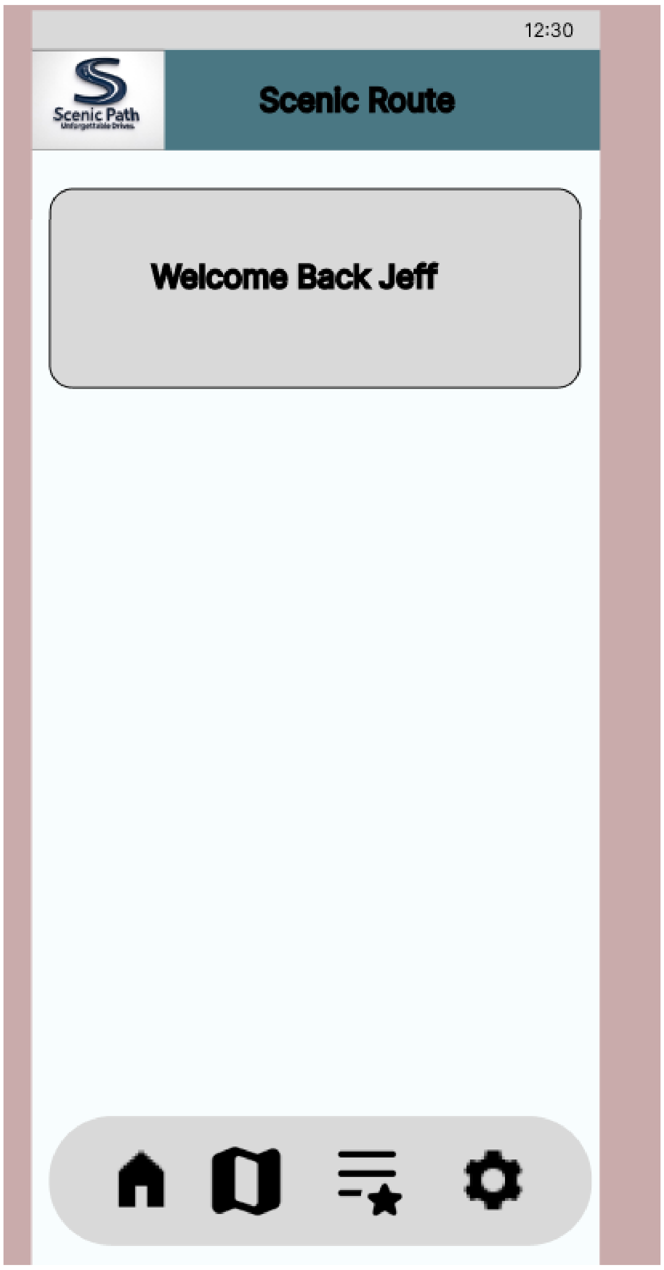


Figure 7: Home Screen [1]

Map Screen

The user will be having to interact with the map to search for a destination by insert their current location and by putting the destination which they would like to go. And then the map will display the path and will display the POIs that the user has selected.

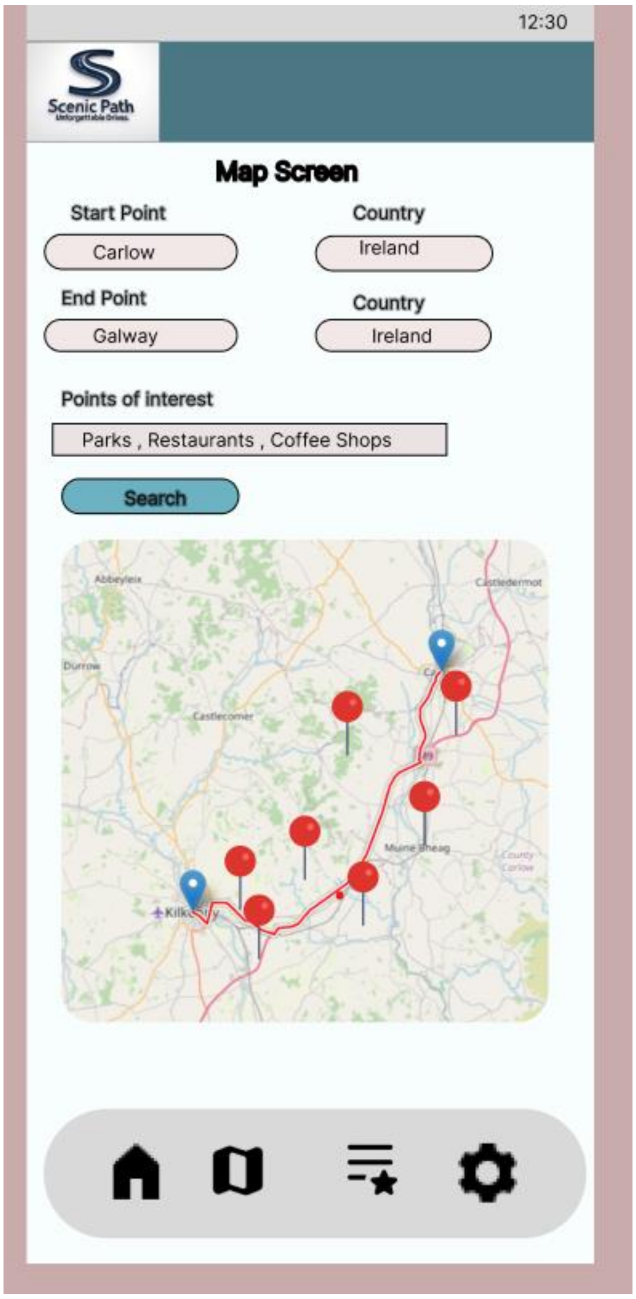
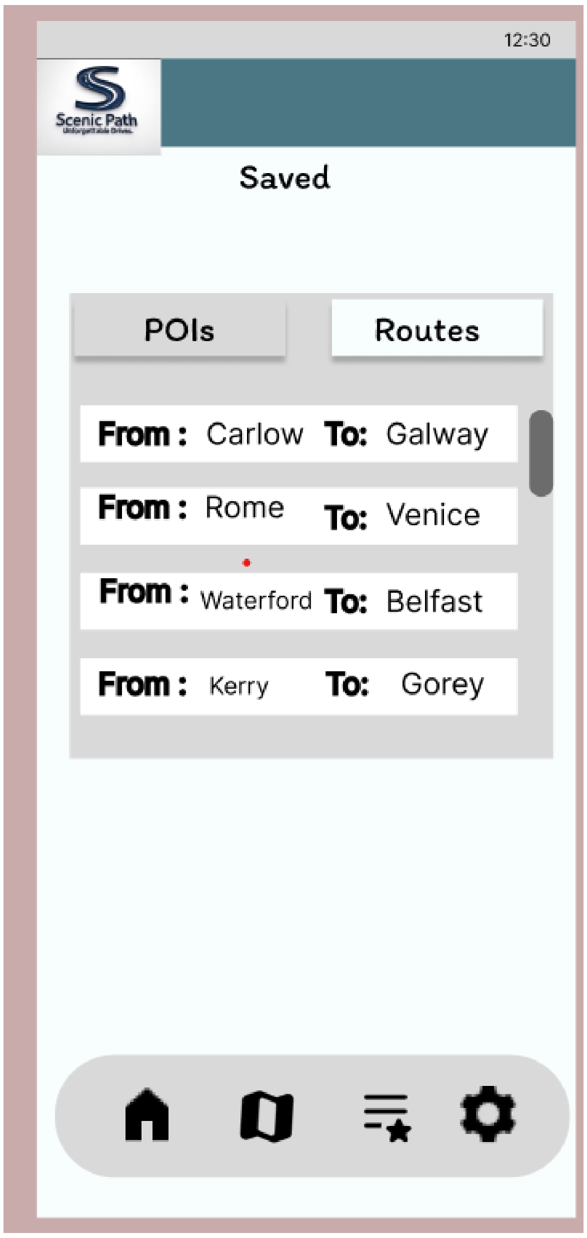


Figure 8: Map Screen [1]

Saved Screen

Users can view their **previously saved routes** and **POIs** for future reference.



From : Carlow **To:** Galway

From : Rome **To:** Venice

From : Waterford **To:** Belfast

From : Kerry **To:** Gorey

Figure 8: Saved Screen [1]

Settings Screen

Users can manage their **profile information** and **log out** from the application.

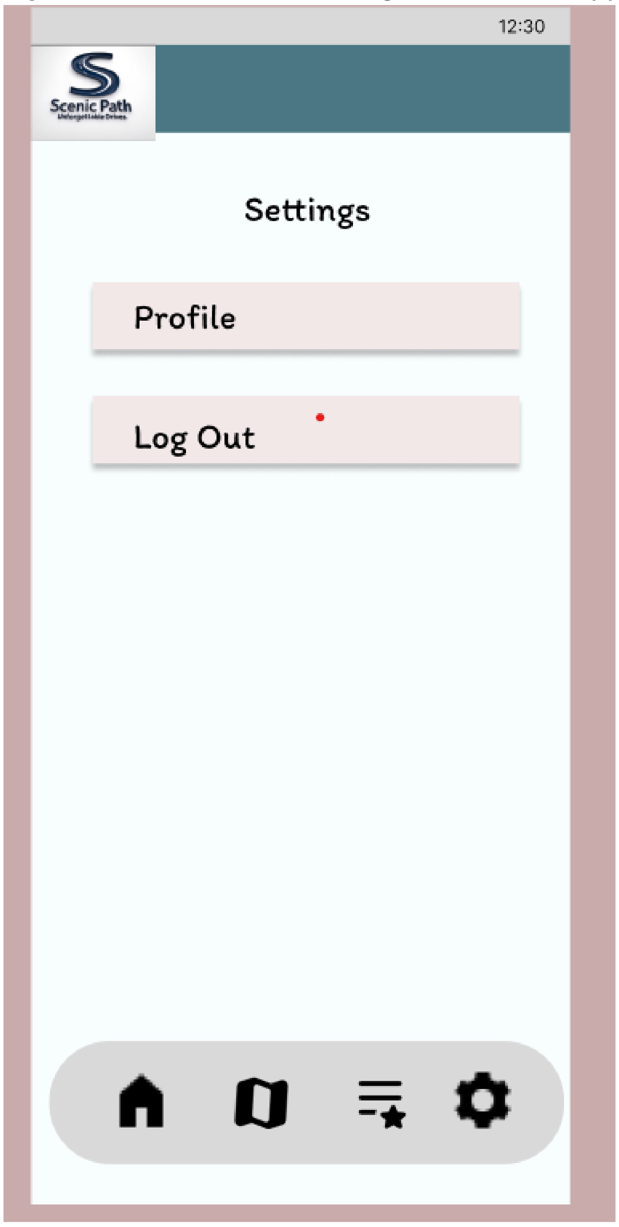


Figure 10: Settings Screen [1]

Methodology

The **Scenic Route Application** follows an **Agile methodology**, incorporating an **iterative development process** with regular feedback loops. Weekly meetings with the supervisor act as informal **Sprint Reviews**, where completed features are demonstrated, and feedback is used to refine and improve the application. This approach ensures continuous progress, adaptability to changes, and alignment with project goals. By integrating key Agile principles such as **incremental development, flexibility, and user-focused improvements**, the project evolves dynamically, incorporating features like **map integration, POI retrieval, and route optimization** in structured iterations.

Iteration Plan

Iteration Plan for Scenic Route Web Application

Iteration 2 Summary & Next Steps for Iteration 3
During this iteration, the following tasks were successfully completed:

1. **Frontend Development**
   - Designed a **responsive web interface** using HTML, CSS (Bootstrap), and JavaScript.
   - Implemented **search fields** for entering start, destination locations and country.
2. **Map Integration & Route Display**
   - Integrated **OpenStreetMap** with **Leaflet.js** for map visualization.
   - Successfully displayed the **route from Point A to Point B** on the map.
3. **POI (Points of Interest) Retrieval**
   - Used the **Yelp API** to fetch POIs along the selected route using python.
   - Displayed relevant POIs near the route based on category and user preferences.

Next Steps for Iteration 3

1. **Backend Development (Flask API)**
   - Set up a **Flask backend** to handle API requests.
   - Create API endpoints to manage user requests for routes and POIs.
   - Connect **Flask with OpenRouteService API & Yelp API** for backend processing.
2. **Database Integration (MongoDB)**
   - Design the **MongoDB schema** for users, saved routes, and POIs.
   - Implement **CRUD operations** to store and retrieve user data.
   - Ensure data storage for **saved routes and POIs**.
3. **Feature Enhancements**
   - Implement a **Save Route** feature for users.
   - Improve **map UI** with better markers and POI descriptions.
   - Add **filtering options** for POI categories (e.g., restaurants, landmarks).
4. **Performance Optimization & Testing**

- Implement error handling for **API failures and invalid inputs**.
- Conduct **frontend and backend testing** for smooth integration.

References

[1] https://www.figma.com [Accessed 08/02/2025]
[2]Choose Logo | Logo Design [Accessed 12/02/2025]
[3] https://www.umletino.com [Accessed 12/02/2025]
[4] https://www.mongodb.com [Accessed 25/01/2025]
[5] HTML Basic [Accessed 10/01/2025]
[6] CSS Tutorial [Accessed 10/01/2025]
[7] JavaScript Tutorial [Accessed 10/01/2025]
[8] Welcome to Flask — Flask Documentation (3.1.x) [Accessed 10/01/2025]
[9] Leaflet - a JavaScript library for interactive maps [Accessed 10/01/2025]
[10] OpenStreetMap [Accessed 10/01/2025]