Design Document

Best Bike Paths

Giuliano Crescimbeni

Luca De Nicola

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

In the world of cycling, it is often useful to record information about trips and track personal performance, as well as to share these experiences with others. Having access to updated data about bike paths—such as their conditions, safety, and suitability—can greatly enhance both the enjoyment and safety of cyclists.

Best Bike Paths (BBP) was conceived in this context, with the goal of creating a digital platform where cyclists can explore, record, and share information about cycling paths. The system promotes collaboration among users, encouraging the community to contribute and maintain reliable data on the status of bike paths.

1.2 Scope

Best Bike Paths (BBP) is an application designed to support cyclists in discovering, recording, and sharing information about bike paths. Through BBP, registered users can record their rides, visualize the paths on a map, and obtain performance statistics such as distance, speed, and duration. When available, the system automatically integrates meteorological information—including temperature, wind speed, and weather conditions.

Users can insert path information either manually, by specifying the streets and their conditions, or automatically, by allowing the application to collect GPS, accelerometer, and gyroscope data during a trip. This data helps identify irregularities such as potholes or rough road segments. Before being published, automatically detected issues must be validated by the user to ensure reliability.

The platform provides map-based search and visualization tools allowing any user, registered or not, to explore available paths between a chosen origin and destination.

1.3 Definitions

- **Automatic Tracking Mode**: A functional mode of the mobile application where the system continuously collects data from the device's sensors (GPS, accelerometer, gyroscope) to reconstruct the cyclist's path and detect road irregularities without requiring manual input during the ride.
- **Data Freshness**: A parameter used by the Merging Engine representing the age

of a submitted report. The system prioritizes more recent data (high freshness) over older data when calculating the consolidated status of a road segment.

- **Draft**: A temporary state of a recorded path (specifically from Automatic Creation) that has been saved locally or on the server but has not yet been validated or finalized by the user. Drafts are not visible in the personal history until confirmed.
- **Merging Engine**: The logical component of the system responsible for consolidating overlapping path segments from different user submissions into a single, consistent global view. It applies the majority consensus algorithm and data freshness logic.
- **Obstacle**: A verified physical impediment or hazard located along a bike path that affects the ride quality or safety.
- **Path Score**: A numerical value computed by the system for each path. Provide a single indicator of the path's "goodness" for ranking search results.
- **Path Segment**: The atomic unit of a path. A path is composed of an ordered sequence of segments.
- **Path Status**: A categorical value assigned to a segment or path indicating its condition. The valid values are: Optimal, Good, Sufficient, Poor, Unsuitable.
- **Path/Trip**: A defined path consisting of a sequence of segments connecting an origin to a destination. A path can be created manually or derived from a recorded trip and can be stored as Private (visible only to the creator) or Public (visible to the community).
- **Post-Ride Validation** : The process where a Registered Cyclist reviews the data collected automatically to confirm real issues and remove false positives before the data is permanently saved.
- **Segment Snapshot**: A specific instance of a road section recorded by a user at a specific point in time. The system aggregates multiple snapshots of the same location to derive the global status.

1.4 Acronyms

- **BBP**: Best Bike Paths
- **UML**: Unified Modeling Language

- **API**: Application Programming Interface
- **GDPR**: General Data Protection Regulation
- **GPS**: Global Positioning System
- **UI**: User Interface

1.5 Abbreviations

- **Gn**: Goal number n
- **Rn**: Requirement number n
- **Dn**: Domain assumption number n
- **WPn**: World phenomena number n
- **SPn**: Shared phenomena number n
- **UC**: Use case

1.6 Revision History

- **Version 1.0**: 07/01/26

1.7 Reference Documents

- Specification Document: Assignment RDD AY 2025-2026
- Course slides

1.8 Document Structure

- **Introduction**: Outlines the purpose, scope, definitions, acronyms, and reference documents for the *Best Bike Paths* project.
- **Architectural Design**: Details the system's high-level architecture.
- **User Interface Design**: Visualizes the application's look and feel via mockups and navigation schemes, linking them to RASD requirements.

- **Requirements Traceability**: Maps functional and non-functional requirements to specific design modules to ensure full coverage.
- **Implementation, Integration and Test Plan**: Defines the development sequence, integration strategy, and testing procedures.
- **Effort Spent**: Summarizes the amount of work dedicated to each phase of the project, specifying the contribution of each team member and the approximate time spent on the various activities.
- **References**: Software used to develop the document.

2 Architectural Design

2.1 Overview

The *Best Bike Paths* (BBP) system adopts a *Four-Tier Client-Server* architecture. This choice ensures separation of concerns, scalability, and independent maintainability of the user interface, business logic, and data storage.

The architecture consists of the following layers:

1. **Presentation Layer (Client Tier)**: Handles user interaction and data visualization. Consistent with the requirements specified in the RASD, the system supports multiple client types, with limitations for the WebApp Client
2. **Proxy Tier**: This tier acts as the entry point for all client requests. It is responsible for handling HTTPS requests and load balancing, distributing incoming traffic across multiple instances of the backend servers. This layer was introduced to enhance system scalability
3. **Application Logic Layer (Server Tier)**: Hosts the core business logic of the system.
4. **Data Layer (Persistence Tier)**: Manages the persistent storage of data. It ensures data integrity and supports concurrent access from multiple application server instances.

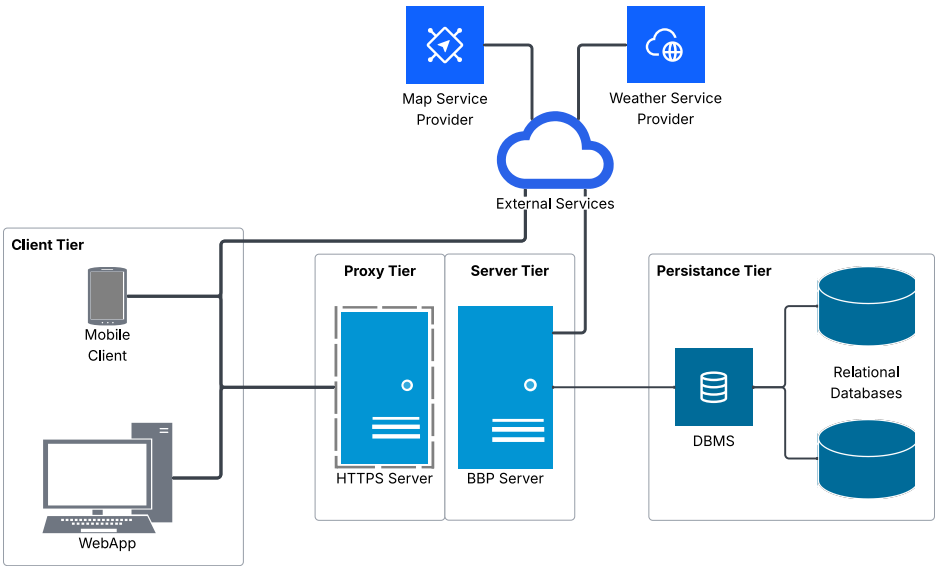


Figure 1: High Level Architecture Diagram

2.2 Component View

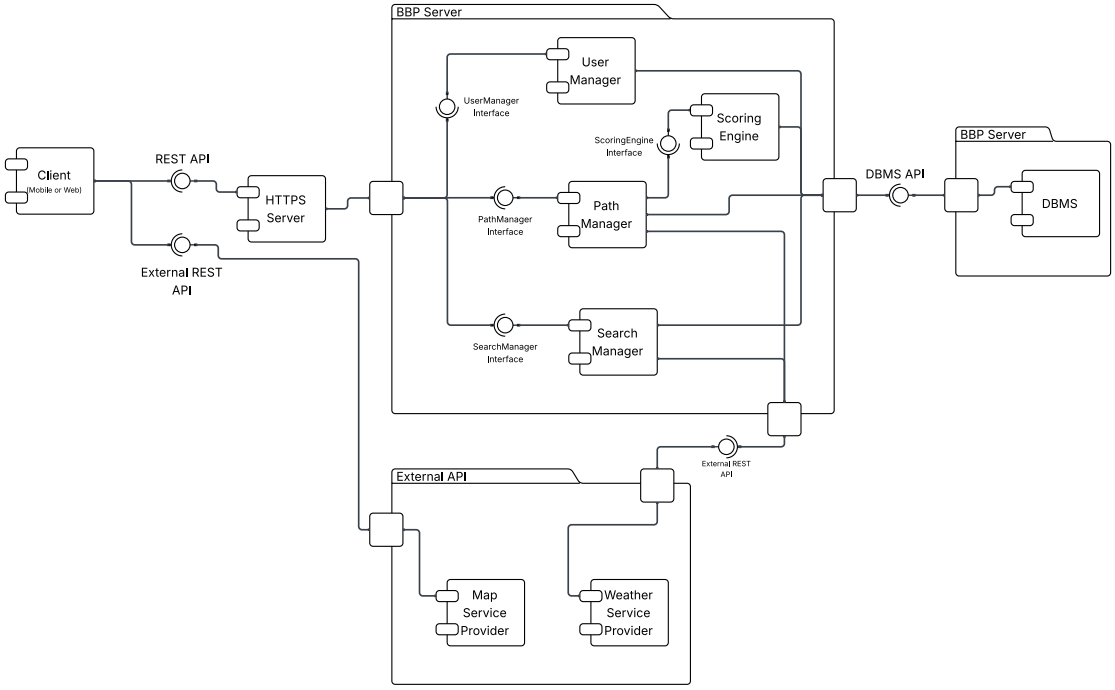


Figure 2: System Components Diagram

User Manager: This component handles the lifecycle of user accounts, including registration, authentication, and the modification of profile information.

Score Engine: It is responsible for calculating the "Path Score" by aggregating user feedback to rank routes based on quality.

Path Manager: This component manages the creation and validation of trips, handling both manually inserted paths and those recorded via automatic tracking, publishing, and details retrieving.

Search Manager: It processes user queries to identify and retrieve available bike paths connecting a specific origin and destination, ensuring results are sorted by their score.

HTTPS Server: It acts as the secure entry point for the system, managing encrypted communication between client applications and the backend to ensure the confidentiality and integrity of data in transit.

DBMS: This component is responsible for the persistent storage and retrieval of all system data.

Weather Service: An external interface used to retrieve real-time meteorological data to enrich trip records based on location and time.

Map Service: An external provider responsible for rendering interactive maps, resolving addresses, and supporting the visualization of paths and obstacles.

2.2.1 User Manager Component

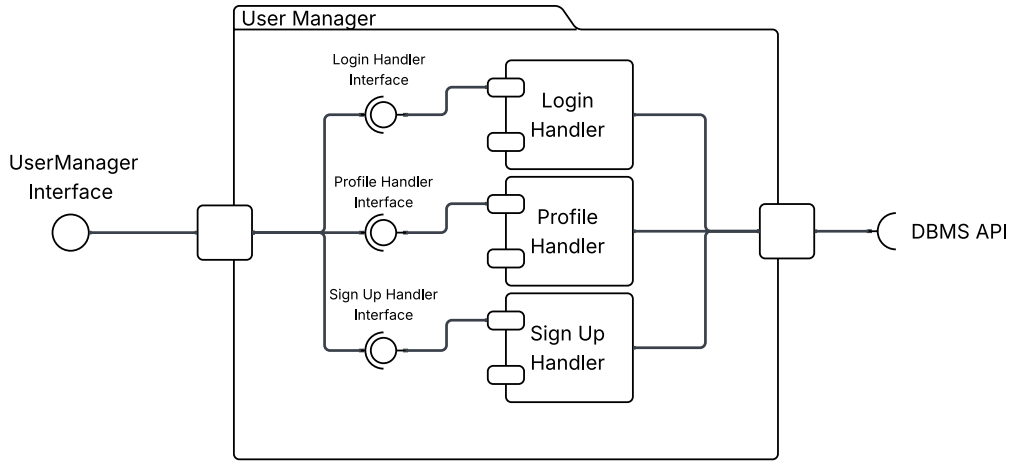


Figure 3: User Manager Component Diagram

Sign Up Handler: This component manages the secure login process for registered cyclists. It is responsible for validating user credentials. It also enforces security constraints by handling password hashing and salting mechanisms to ensure sensitive data is never stored in plain text.

Sign Up Handler: This component handles the sign-up logic for new users. It validates input data (name, surname, email, password) and performs a check to ensure the provided email address is not already associated with an existing account before creating a new profile.

Profile Handler: This component enables registered users to view and modify their personal account information. Processes updates to profile details (such as name and surname) and validates the new input data before persisting changes to the database.

2.2.2 Path Manager Component

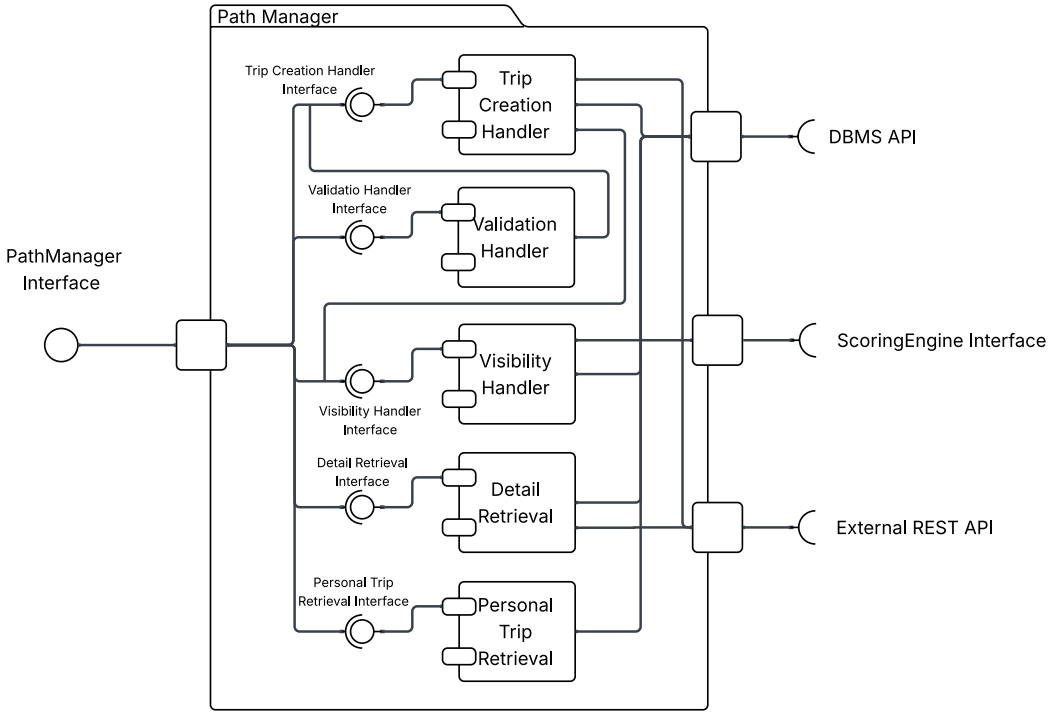


Figure 4: Path Manager Component Diagram

Trip Creation Handler: This component manages the initial recording of a trip. It handles Manual Trip Creation by processing the sequence of streets and status entered by the user. It coordinates with the Weather Service Interface to enrich the trip with meteorological data upon saving.

Validation Handler: This component is responsible for the Post-Ride Validation workflow. It serves the list of automatically detected obstacles and streets to the user for review. Processes the user's confirmation or rejection before the trip is permanently finalized by saving it with the Trip Creation Handler Component

Visibility Handler: This component handles the state changes of a trip. When a trip becomes Public, or changes from public to private, the component triggers the Scoring Engine Component to incorporate the new data into the global dataset.

Detail Retrieval: This component handles the retrieval of comprehensive data for a single specific trip. It serves data for both private trips (for the creator) and public trips (for any user).

Personal Trip Retrieval: This component is responsible for fetching the collection of trips created by a specific registered user. It queries the database to retrieve the personal history list, providing summary details and distinguishing between 'Private' and 'Public' visibility states.

2.2.3 Search Manager Component

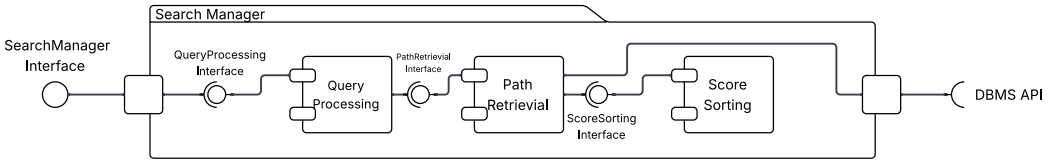


Figure 5: Search Manager Component Diagram

Query Processing: This component validates the user's search inputs, ensuring that the specified origin and destination (provided via address text or map selection) are valid.

Path Retrieval: This component is responsible for querying the database to fetch all available bike paths that physically connect the origin and destination points. It retrieves the list of matching routes along with their associated data.

Score Sorting: This component processes the list of retrieved paths and orders them based on their "Path Score".

2.2.4 Scoring Engine Component

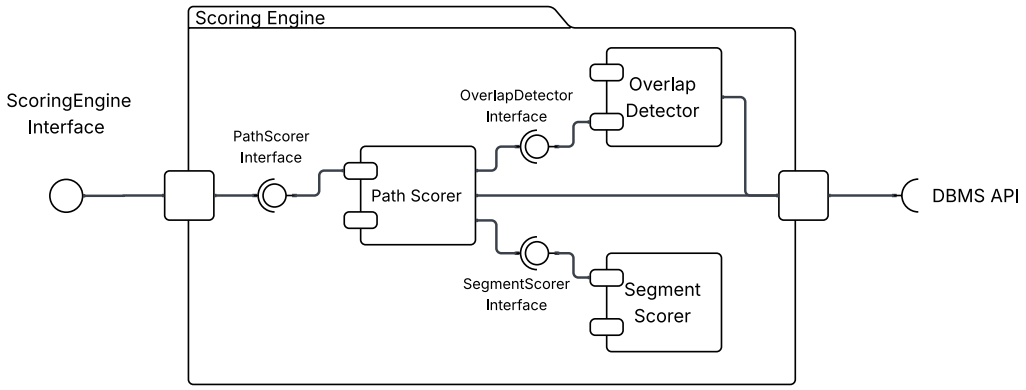


Figure 6: Scoring Engine Component Diagram

Path Scorer: It coordinates the update logic by triggering the detection of shared road sections and the recalculation of their score. Finally, it computes the score for the new input path, if any, and re-evaluates the scores of all existing paths affected by the changes.

Overlap Detector: It queries the database to identify segments affected by the status change of a path. It returns the set of segments that overlap with it.

Segment Scorer: Performs data merging. Generate a single, updated quality score for each individual road segment affected by the changes.

2.3 Deployment View

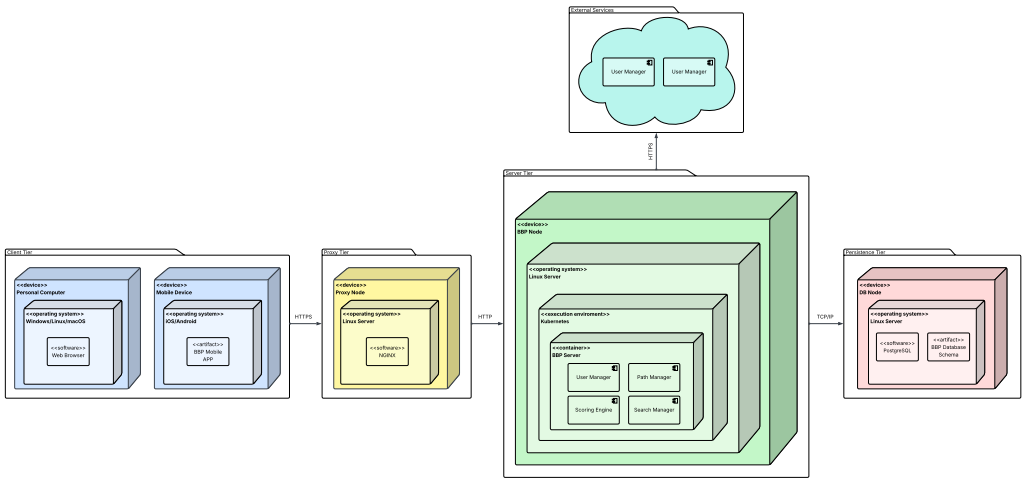


Figure 7: Deployment View Diagram

The deployment is structured as follows:

- **Client Tier**: Users access the system via two primary device types:
  - **Mobile Devices** (iOS/Android): Run the native *BBP Mobile App*, which communicates directly with the proxy.
  - **Personal Computers** (Windows/Linux/macOS): Access the web interface via a standard *Web Browser*.
- **Proxy Tier**: Represents the system's entry point. It consists of a **Proxy Node** running **Linux** and the **NGINX** server software. This node handles HTTPS and load balancing, and forwards requests to the application server via internal HTTP.
- **Server Tier**: Hosts the application logic. It is composed of **BBP Nodes** (Linux Servers) managed by a **Kubernetes** execution environment. The *BBP Server* container runs inside this cluster, encapsulating the core components (*User Manager, Path Manager, Scoring Engine, Search Manager*). This tier also manages requests to external services.
- **Persistence Tier**: Dedicated to data storage. It consists of a **DB Node** (Linux Server) running the **PostgreSQL** DBMS. This node hosts the *BBP Database Schema* artifact and communicates with the Server Tier via TCP/IP.

2.4 Runtime View

2.4.1 UC1 - User Registration

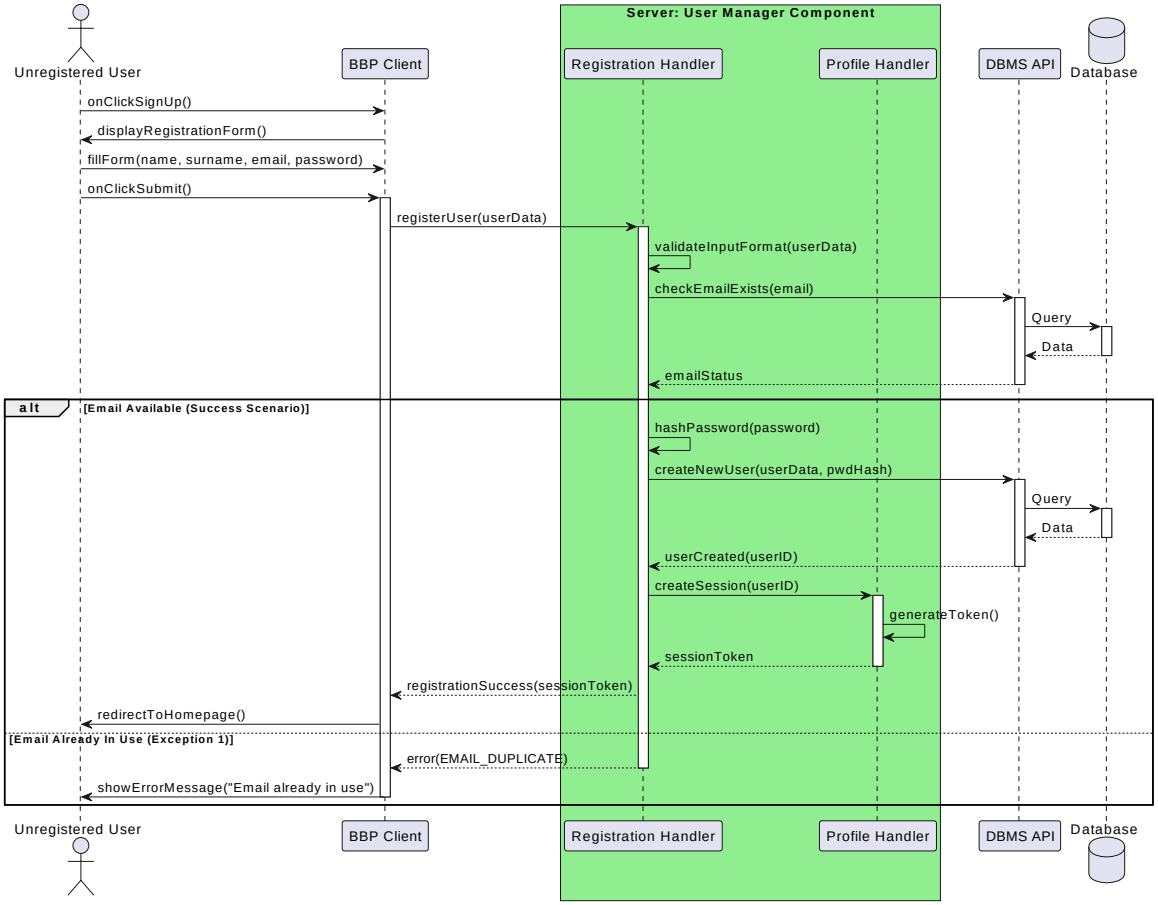


Figure 8: UC1 Runtime View

Figure 8 describes the sequence where an unregistered user interacts with the **Registration Handler** to create a new account. The system validates the email uniqueness and, via the **DBMS API**, creates the record, and automatically establishes a session via the **Session Manager**.

2.4.2 UC2 - User Login

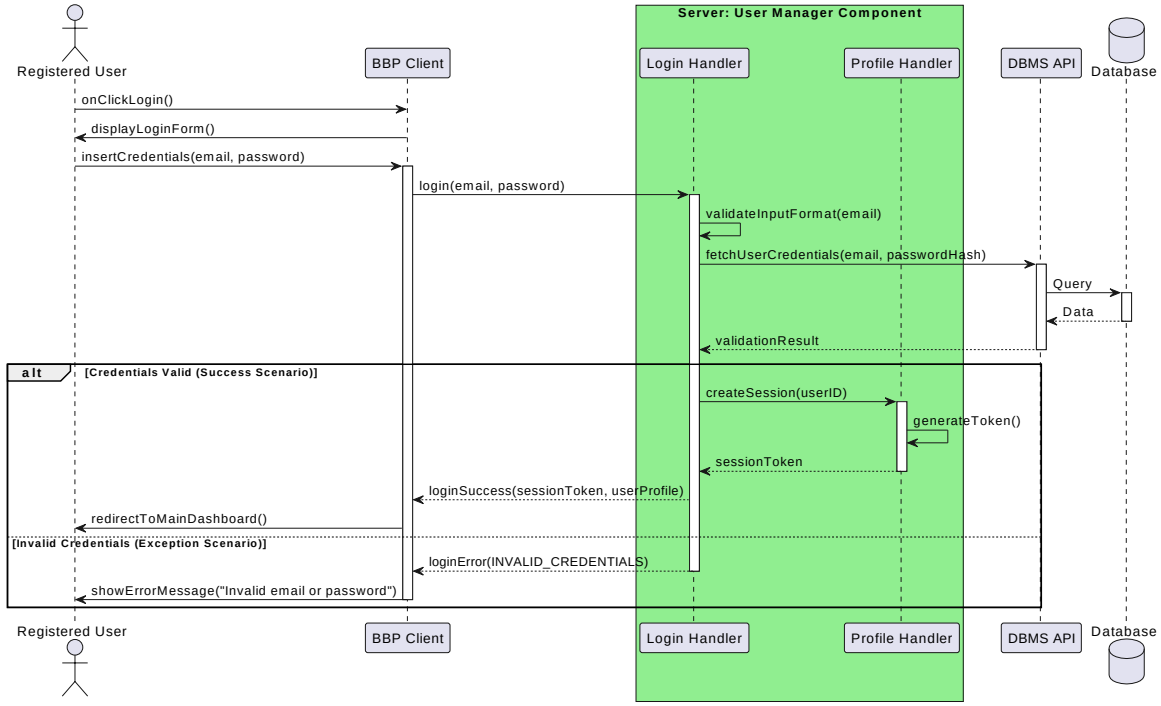


Figure 9: UC2 Runtime View

Figure 9 illustrates the authentication process managed by the **Login Handler**. It verifies the provided credentials against the database and, upon success, triggers the **Session Manager** to issue a secure token for the client.

2.4.3 UC3 - View Personal Trips

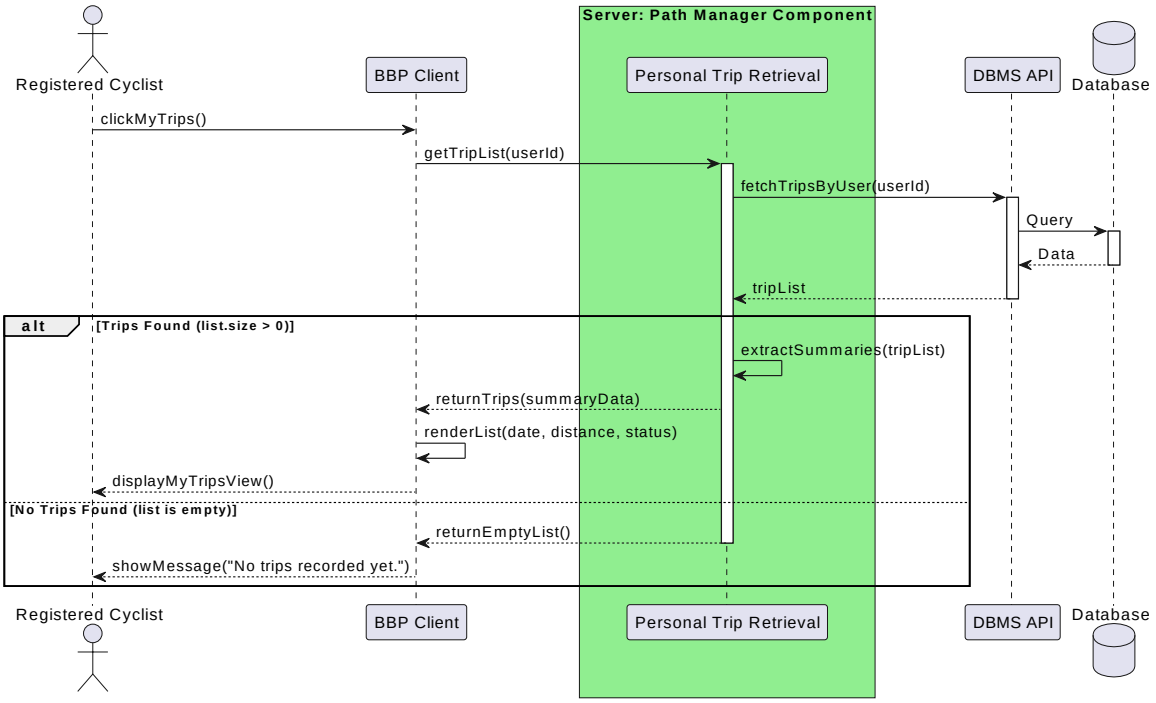


Figure 10: UC3 Runtime View

Figure 10 shows the retrieval of the current user's personal trips. The **Personal Trip Retrieval** fetches the data via the **DBMS API** and returns all the trips to be rendered by the client.

2.4.4 UC4 - View Trip Details

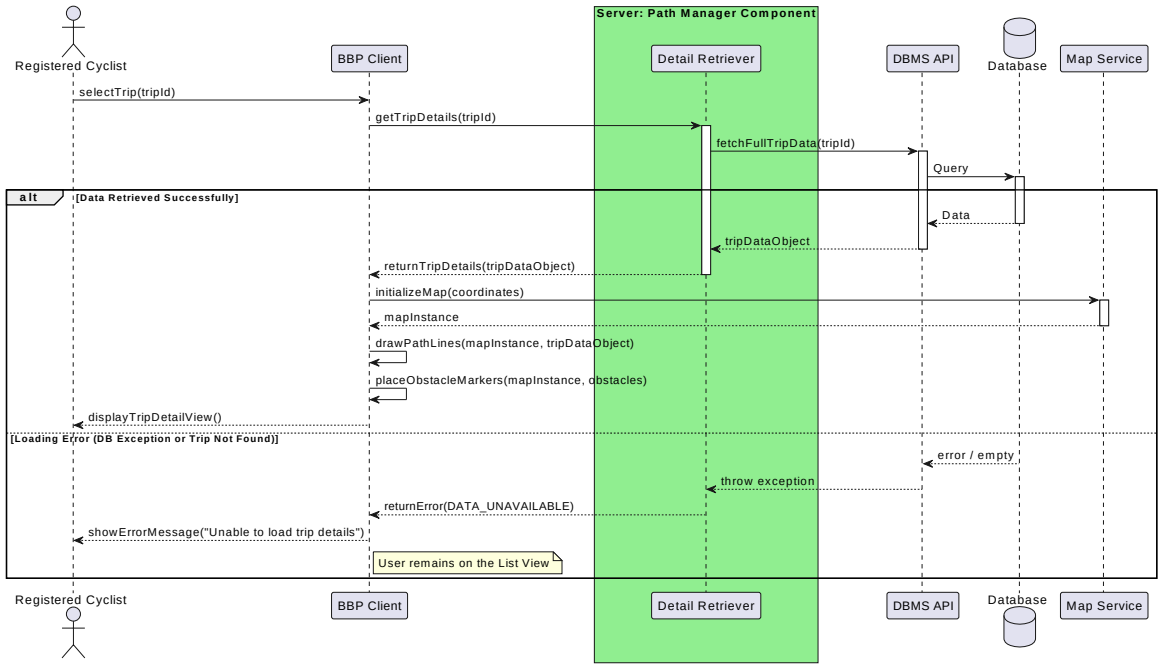


Figure 11: UC4 Runtime View

Figure 11 covers the retrieval of comprehensive data for a specific trip selection. The **Trip Detail Retriever** fetches the full path, metrics, and associated obstacles to allow the client to render the detailed map view.

2.4.5 UC5 - Modify Profile Informations

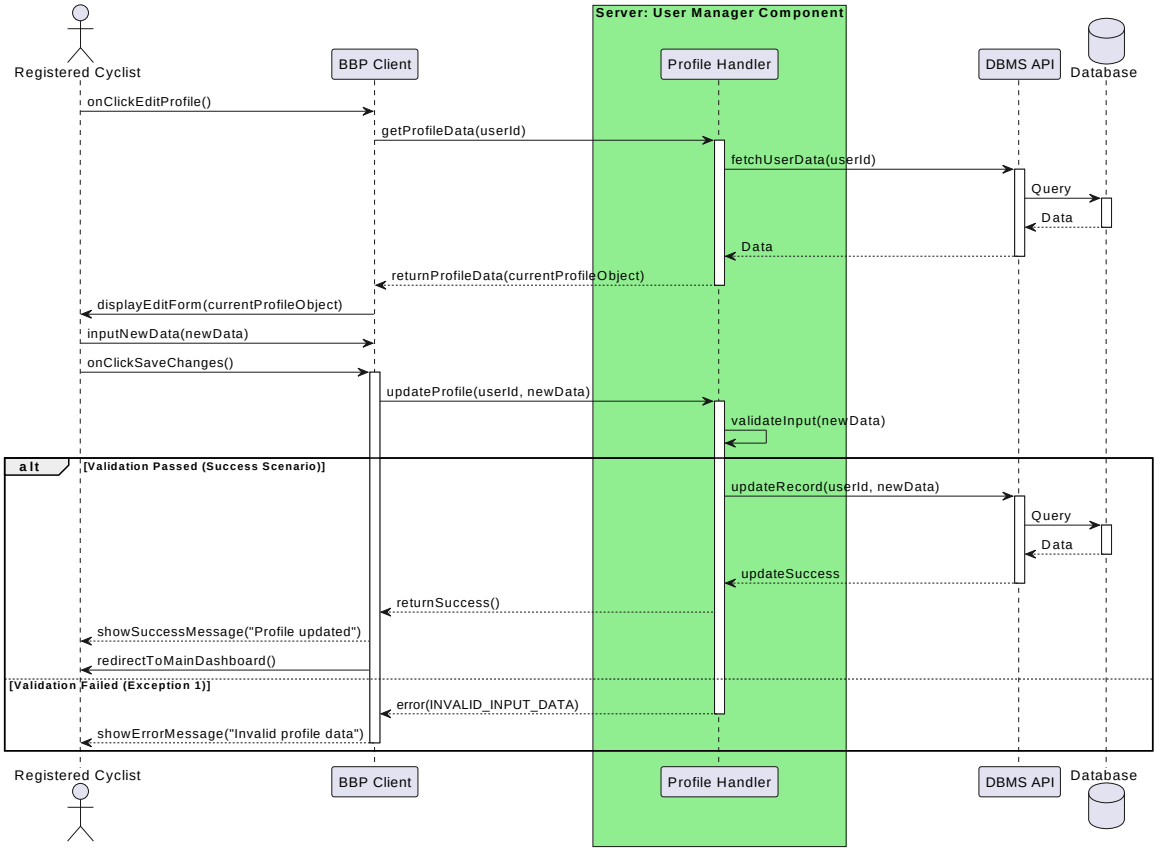


Figure 12: UC5 Runtime View

Figure 12 shows the flow for updating personal details. The **Profile Manager** validates the new input before asking the **DBMS API** to save the changes in the database.

2.4.6 UC6 - Search for Paths

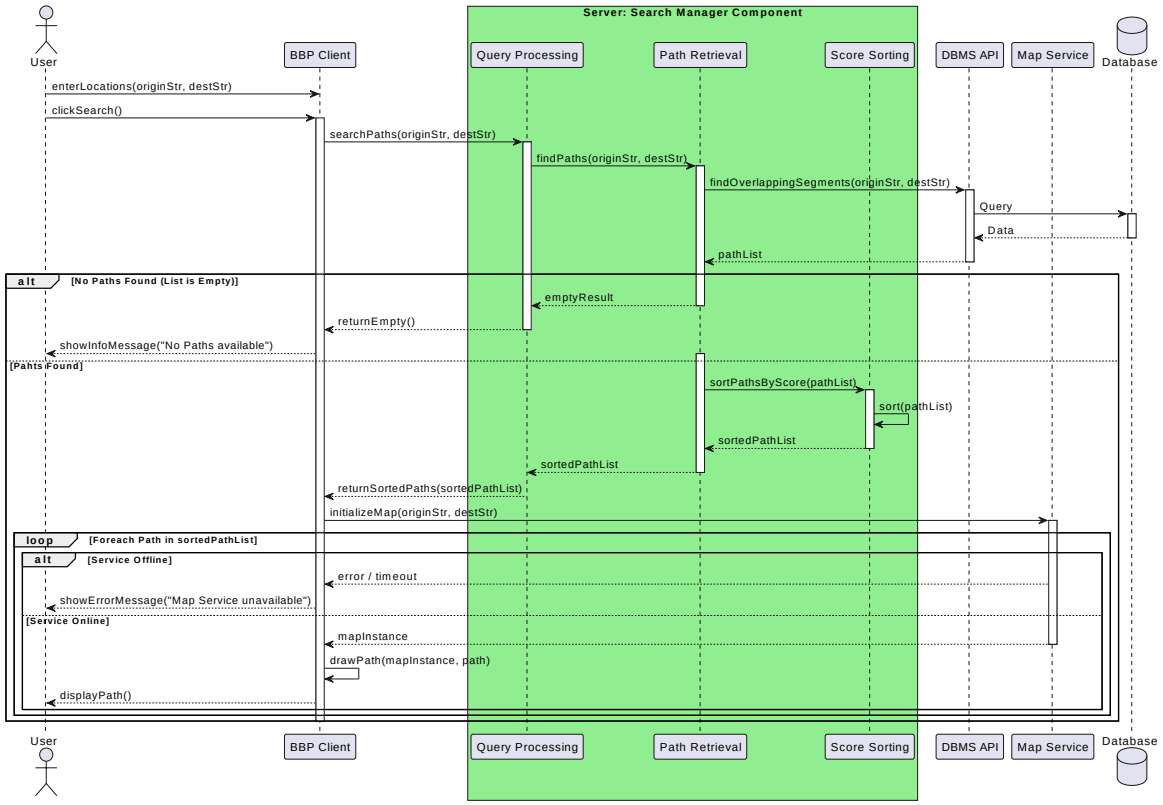


Figure 13: UC6 Runtime View

Figure 13 details the search sequence initiated by the **Search Manager**. It retrieves all the paths from two specific addresses and invoke the **Score Sorting** component to rank results before displaying them on the map.

2.4.7 UC7/UC10 - Manual Trip Creation / Path Visibility

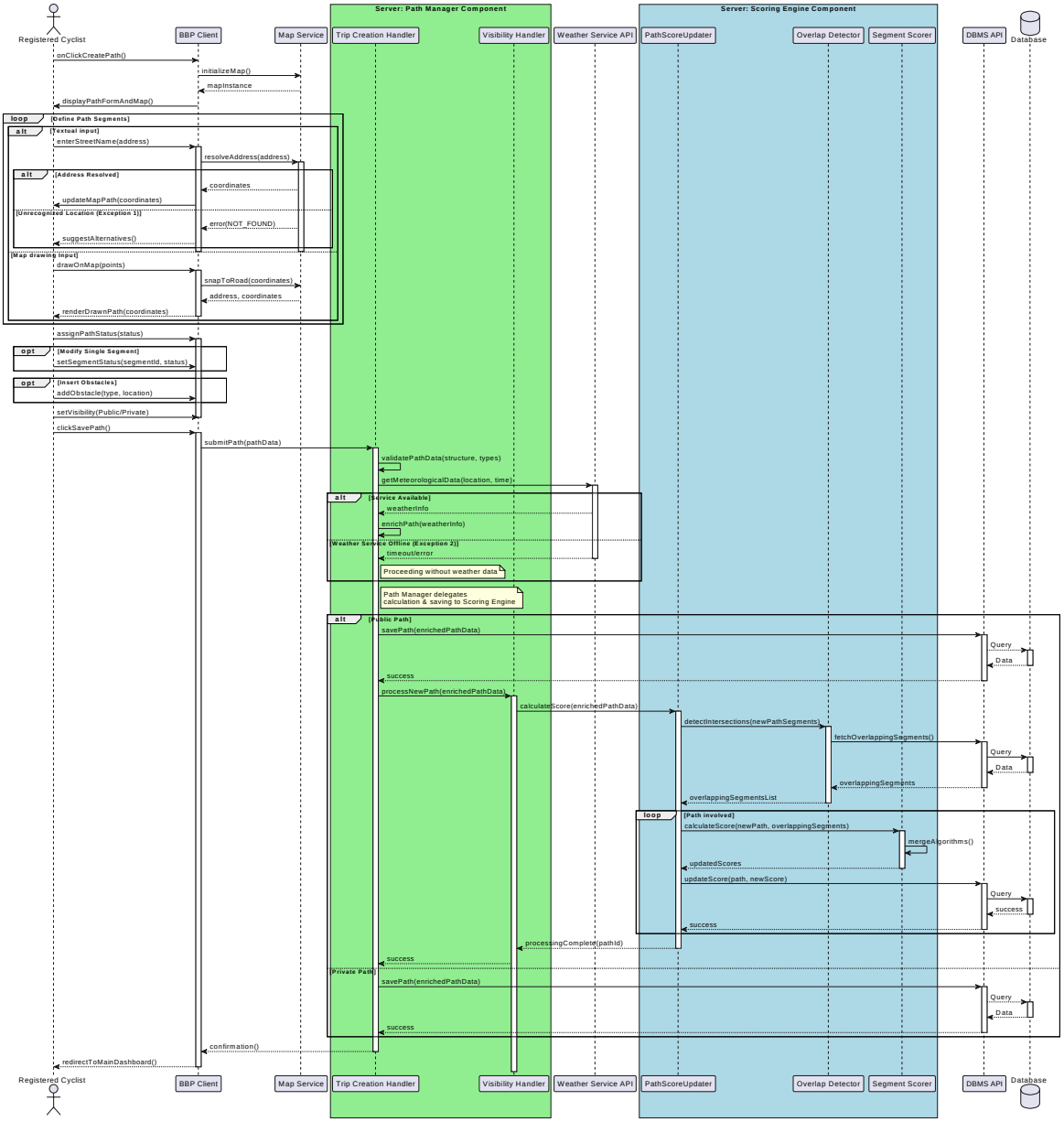


Figure 14: UC7-UC10 Runtime View

Figure 14 describes the manual creation of a route. The **Path Submission Handler** coordinates with external weather services and the **Scoring Engine** to process, calculate metrics, and save the new path.

*Note: The functionality of **UC10 (Change Path Visibility)** has been incorporated into this flow to present the complete process. A standalone diagram for UC10 has been omitted, as its interaction flow would be identical to the "Scoring Engine Component" sub-sequence shown here, rendering a separate figure redundant.*

2.4.8 UC8 - Automatic Path Creation

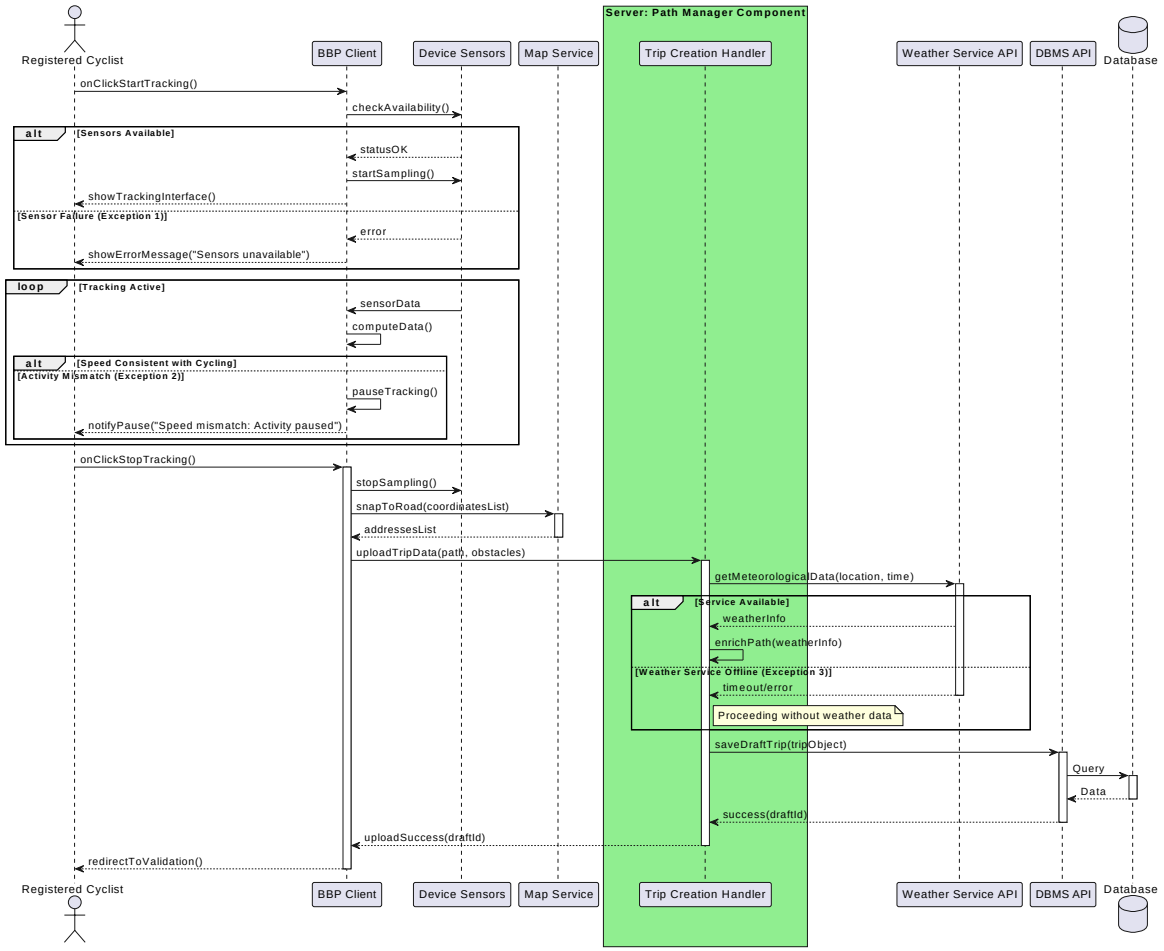


Figure 15: UC8 Runtime View

Figure 15 illustrates the automatic recording phase. The Client handles sensor data collection and local validation (Activity Recognition). Upon completion, the **Trip Creation Manager** on the server enriches the data with weather info and saves it as a 'Draft'.

2.4.9 UC9 - Post Ride Validation

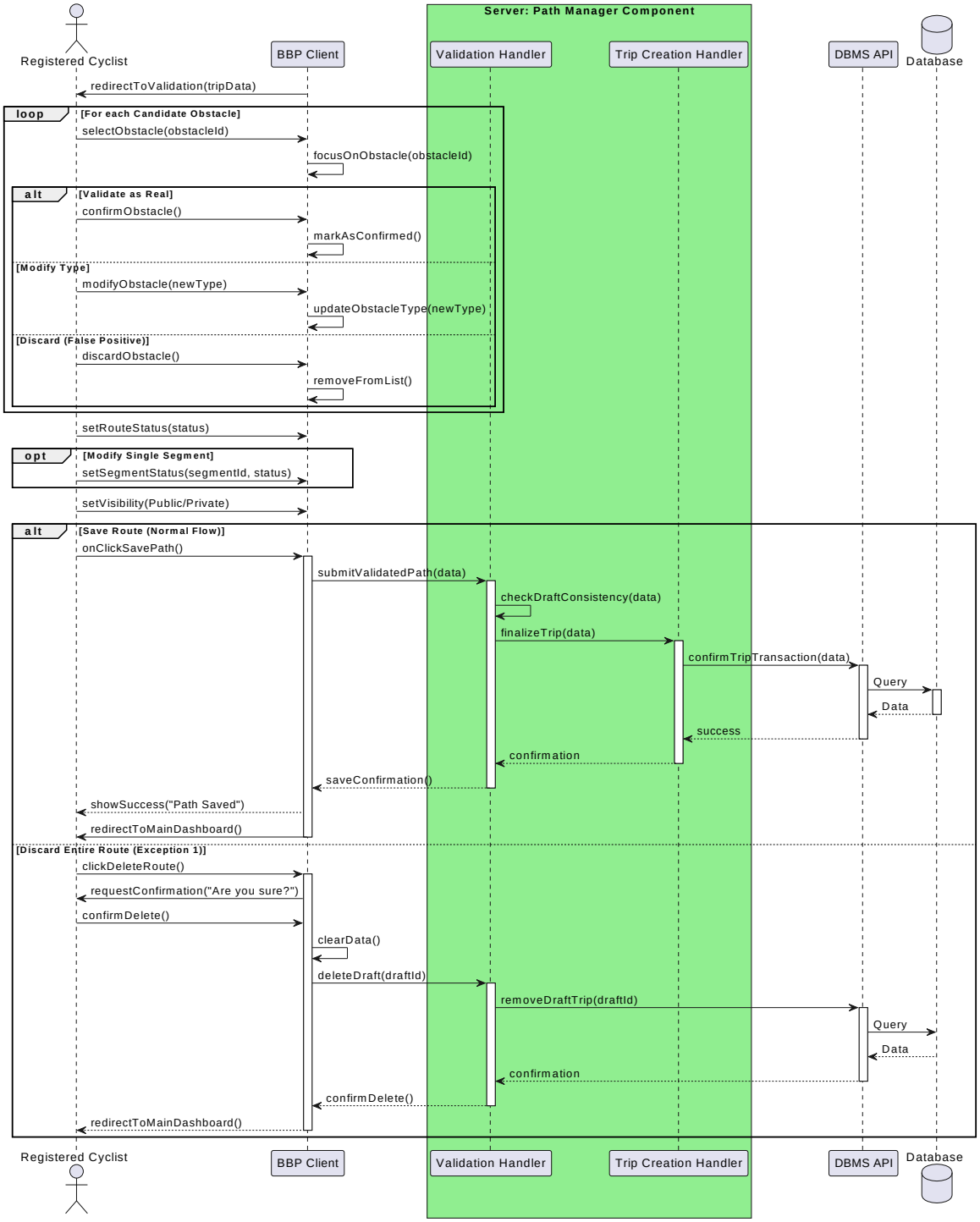


Figure 16: UC9 Runtime View

Figure 16 shows the consolidation of a draft trip. The **Validation Handler** processes user confirmations regarding obstacles and route status. Finally, the **Trip Creation Handler** persists the validated trip .

Note: The logic for score calculation and visibility handling is omitted into this validation workflow. Considering it would result in a diagram with identical interactions to the visibility settings step depicted above.

2.5 Component Interfaces

User Manager Component

- Login Handler
  - `login(email, password)`
  - `validateInputFormat(email)`
- Registration Handler
  - `registerUser(userData)`
  - `validateInputFormat(userData)`
- Profile Handler
  - `createSession(userID)`
  - `generateToken()`
  - `getProfileData(userId)`
  - `validateInput(newData)`
  - `updateProfile(userId, newData)`

Path Manager Component

- Personal Trip Retrieval
  - `getTripList(userId)`
  - `extractSummaries(tripList)`
- Detail Retriever
  - `getTripDetails(tripId)`
- Trip Creation Handler
  - `submitPath(pathData)`
  - `validatePathData(structure, types)`
  - `enrichPath(weatherInfo)`
  - `uploadTripData(path, obstacles)`
  - `finalizeTrip(data)`
- Validation Handler
  - `submitValidatedPath(data)`

  - `checkDraftConsistency(data)`
  - `deleteDraft(draftId)`

Search Manager Component

- Query Processing
  - searchPaths(originStr, destStr)
- Path Retrieval
  - `findPaths(originStr, destStr)`
- Score Sorting
  - `sortPathsByScore(pathList)`

Scoring Engine Component

- Path Score Updater
  - `calculateScore(enrichedPathData)`
- Overlap Detector
  - `detectIntersections(newPathSegments)`
- Segment Scorer
  - `calculateScore(newPath, overlappingSegments)`
  - `mergeAlgorithm`

2.6 Selected Architectural Styles and Patterns

This section details the architectural choices and design patterns adopted for the development of the *Best Bike Paths* system, justifying them based on the non-functional requirements defined in the RASD.

2.6.1 4-Tier Client-Server Architecture

As outlined in the architectural overview (Section 2.1), the system implements a **Four-Tier Architecture**. The introduction of a dedicated **Proxy Tier** was deemed necessary to meet the high scalability and security requirements of the project.

This separation allows for independent scaling of each tier and enhances security by hiding the application topology behind the proxy.

2.6.2 REST (Representational State Transfer) Architectural Style

Given the distributed nature of the system (Mobile App, Web Client, and External Services), the **REST** architectural style was selected for communication between the Client Tier and the Server Tier, and also between components in the Server Tier.

2.6.3 DAO (Data Access Object) Pattern

Internally, the server components employ the **Data Access Object** pattern. By encapsulating all database queries within the DAO layer, the business components doesn't need to manage SQL implementation and queries.

3 User Interface Design

This chapter presents the visual and interactive design of the *Best Bike Paths* system. The interface is designed to ensure a seamless and intuitive user experience across the two supported platforms.

The following sections illustrate the navigational structure through the **User Interface Flow Diagrams**, followed by the visual **Mockups** of the key system screens for the mobile application artifact.

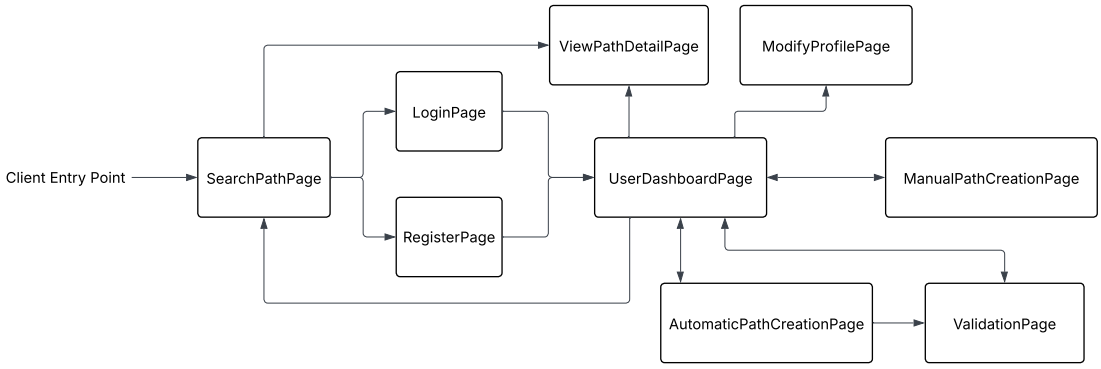


Figure 17: User Flow Diagram

3.1 Landing Page - Path Search Page

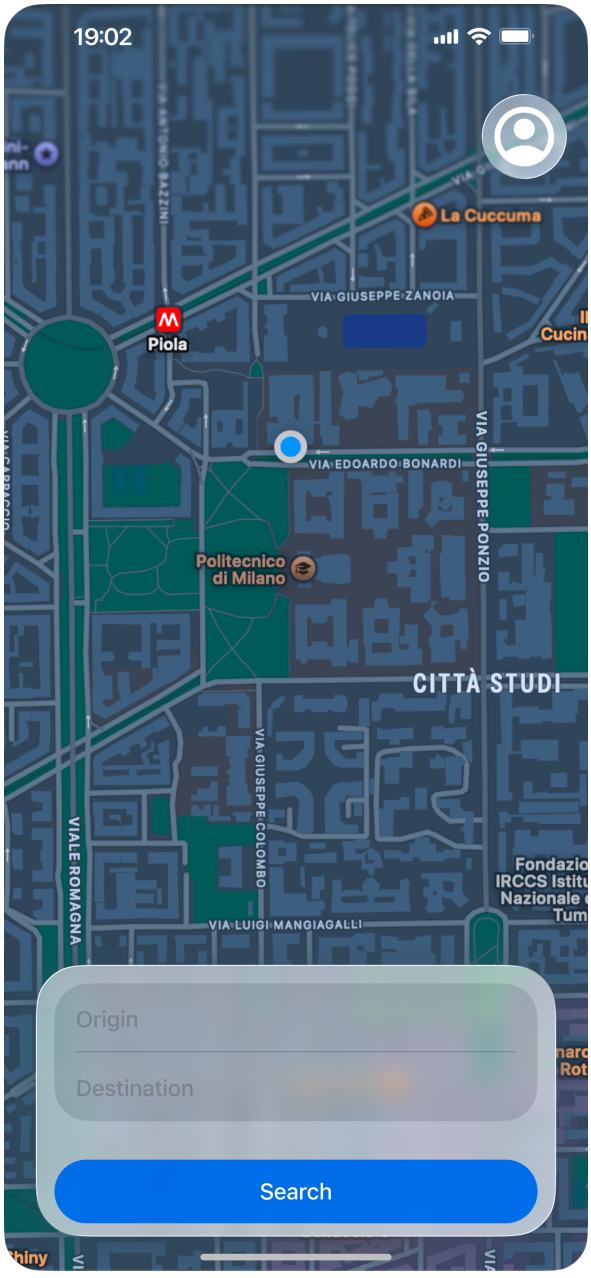


Figure 18: Landing Page

3.2 Login and register

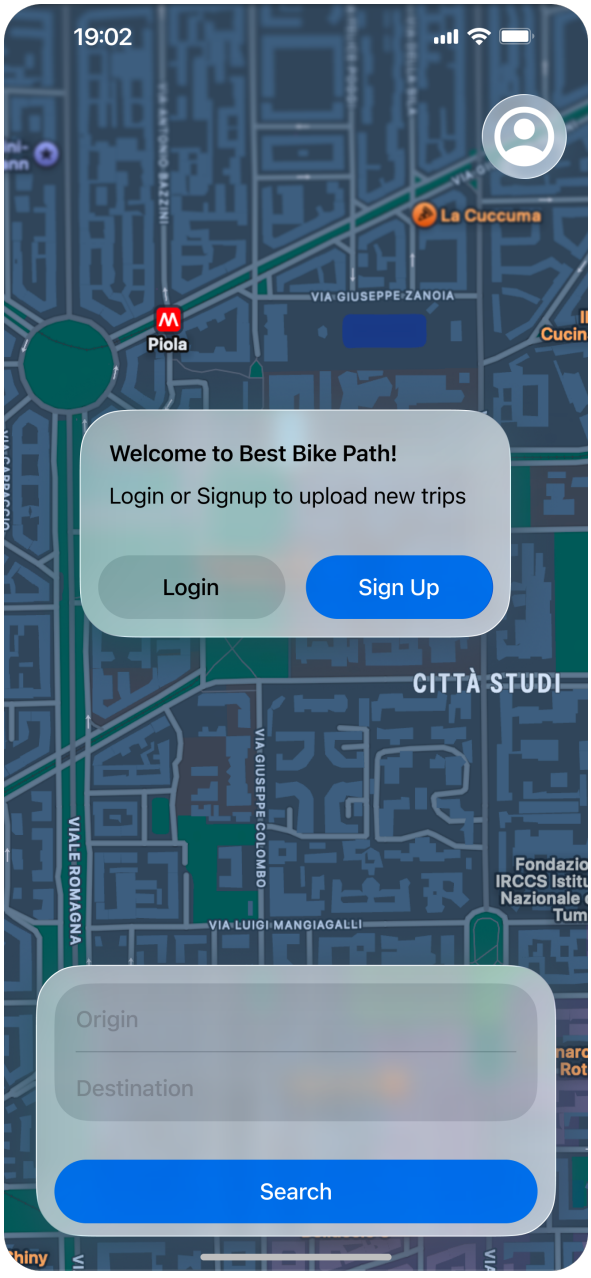
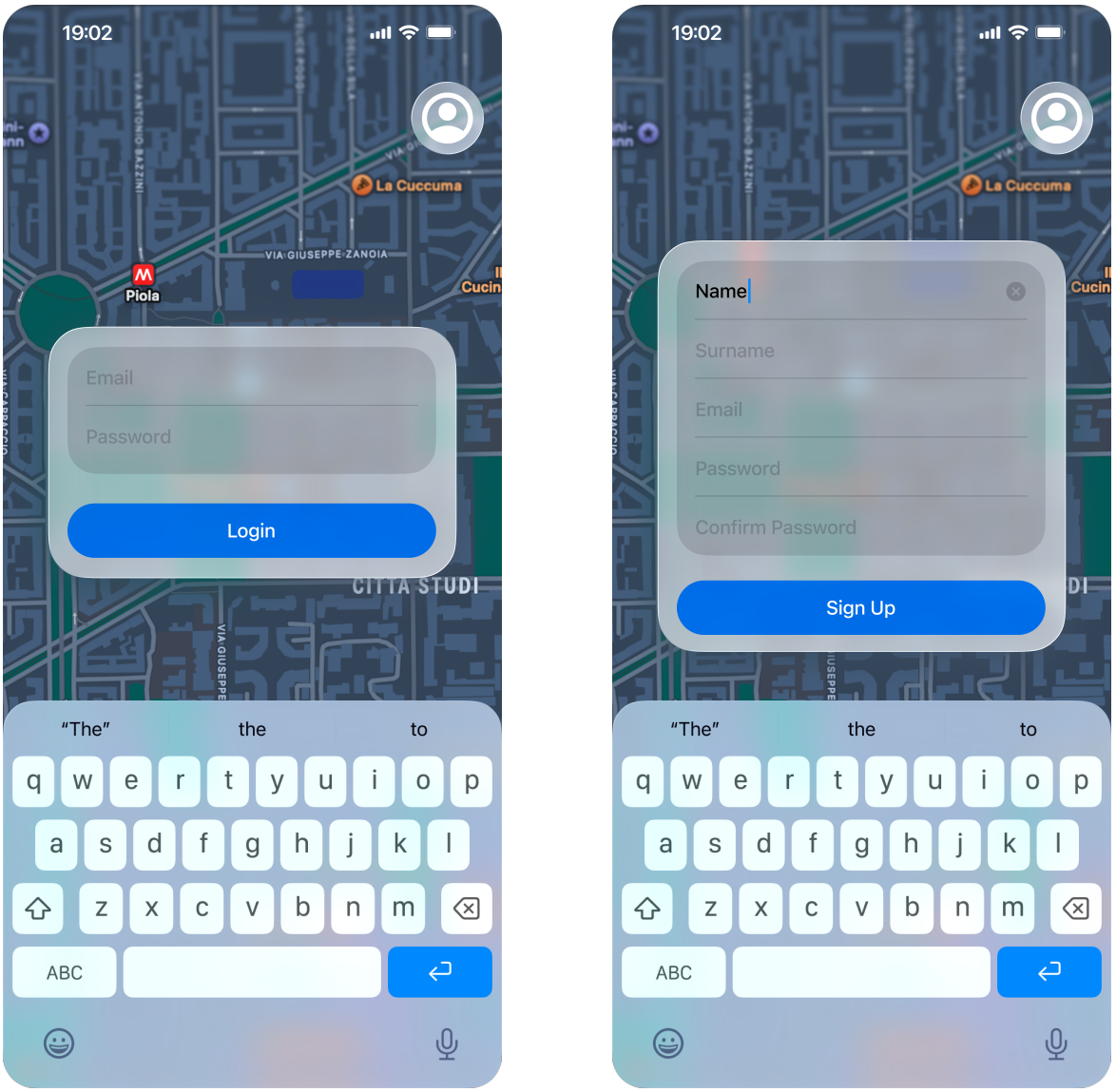


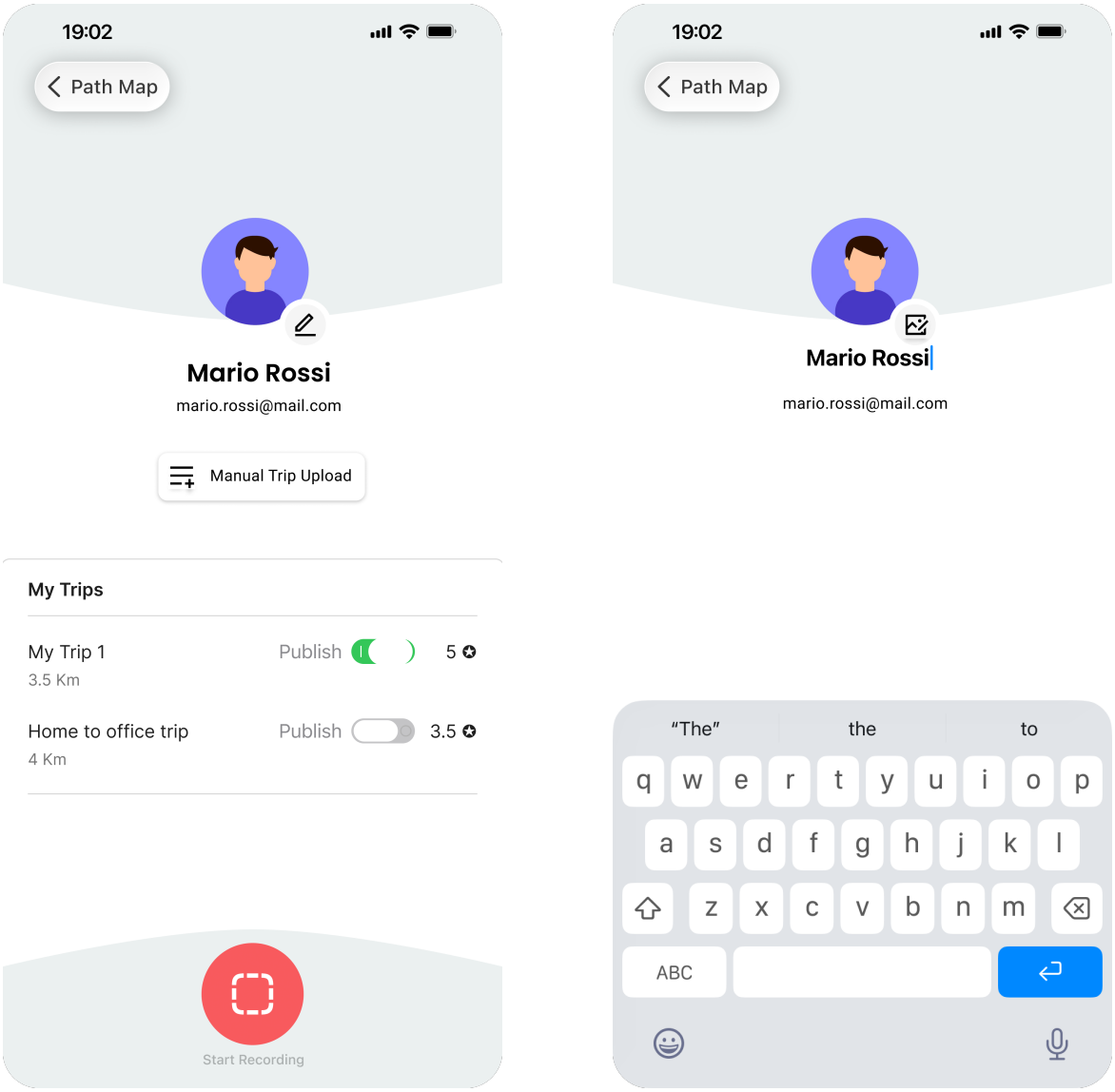
Figure 19: Login and register popup



(a) Login Screen (b) Registration Screen

Figure 20: Authentication Interface Design

3.3 User Dashboard

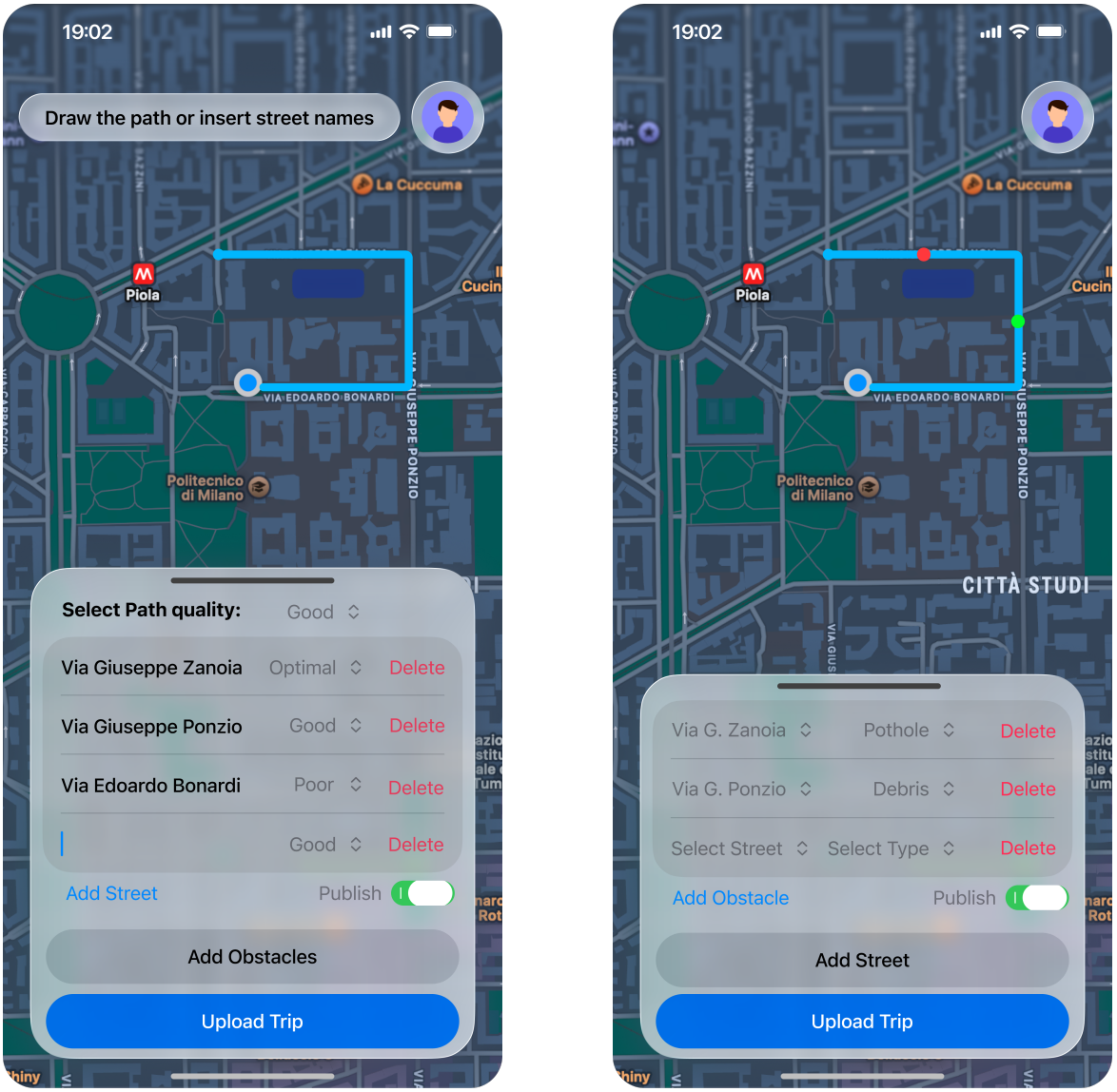


(a) User Dashboard Screen

(b) Profile info update

Figure 21: User dashboard view and update information screen

3.4 Manual Path Upload



(a) Streets Upload

(b) Obstacles Upload

Figure 22: Manual Path Upload

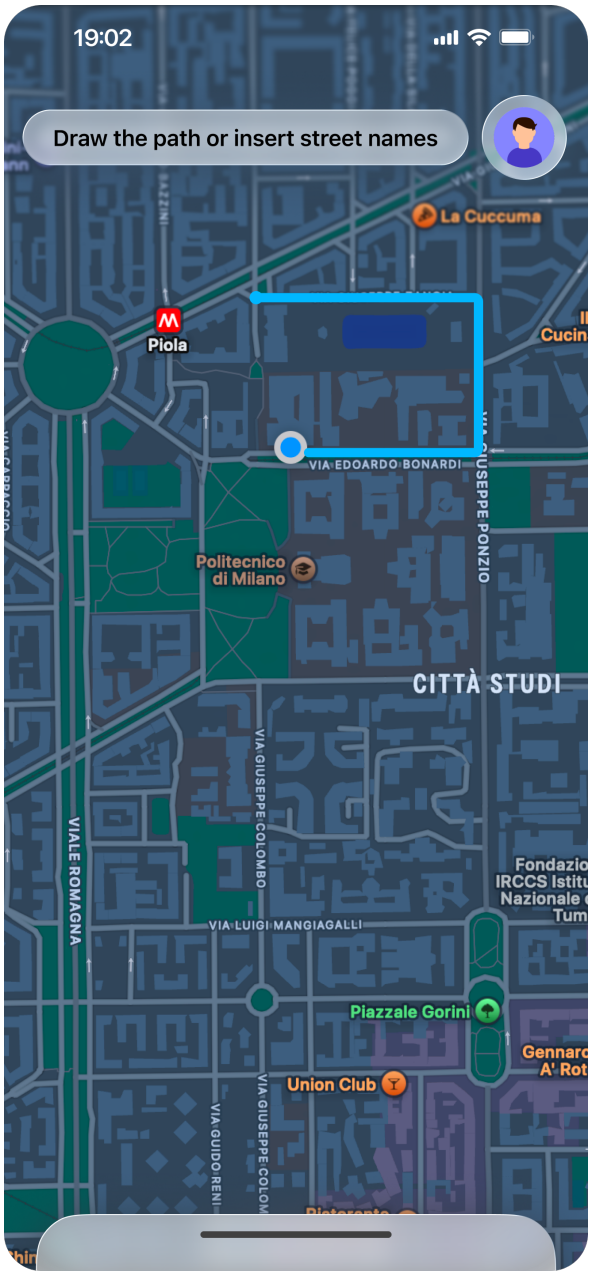


Figure 23: You can lower the bottom panel to draw directly on the map

3.5 Automatic Path Upload

3.5.1 Tracking Screen

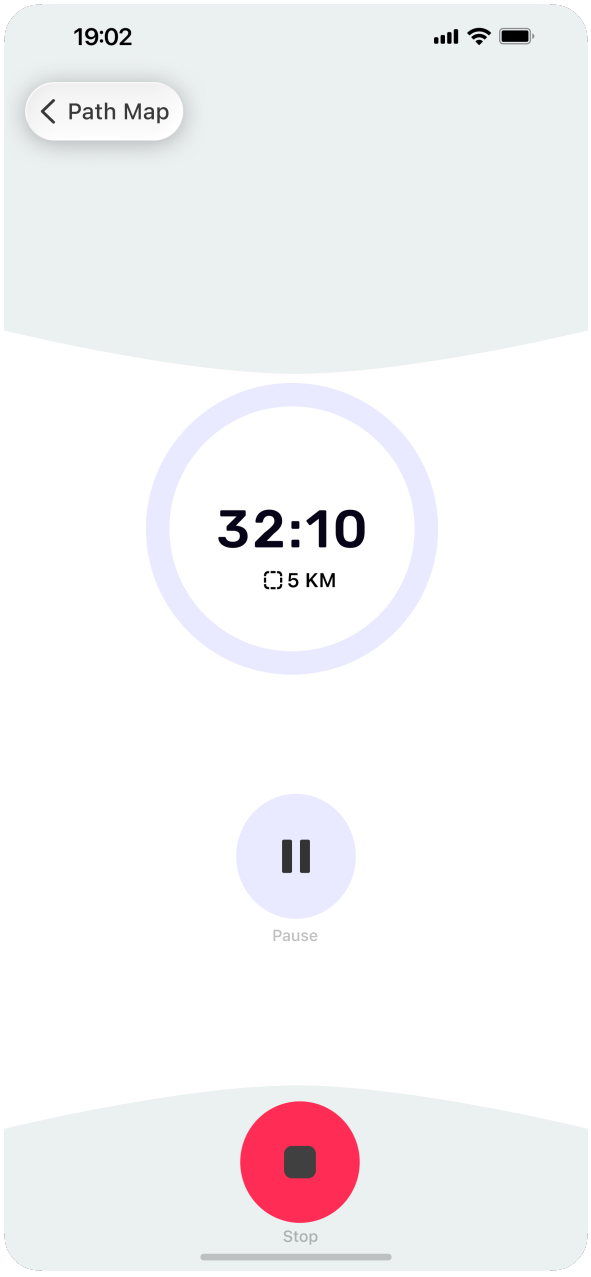
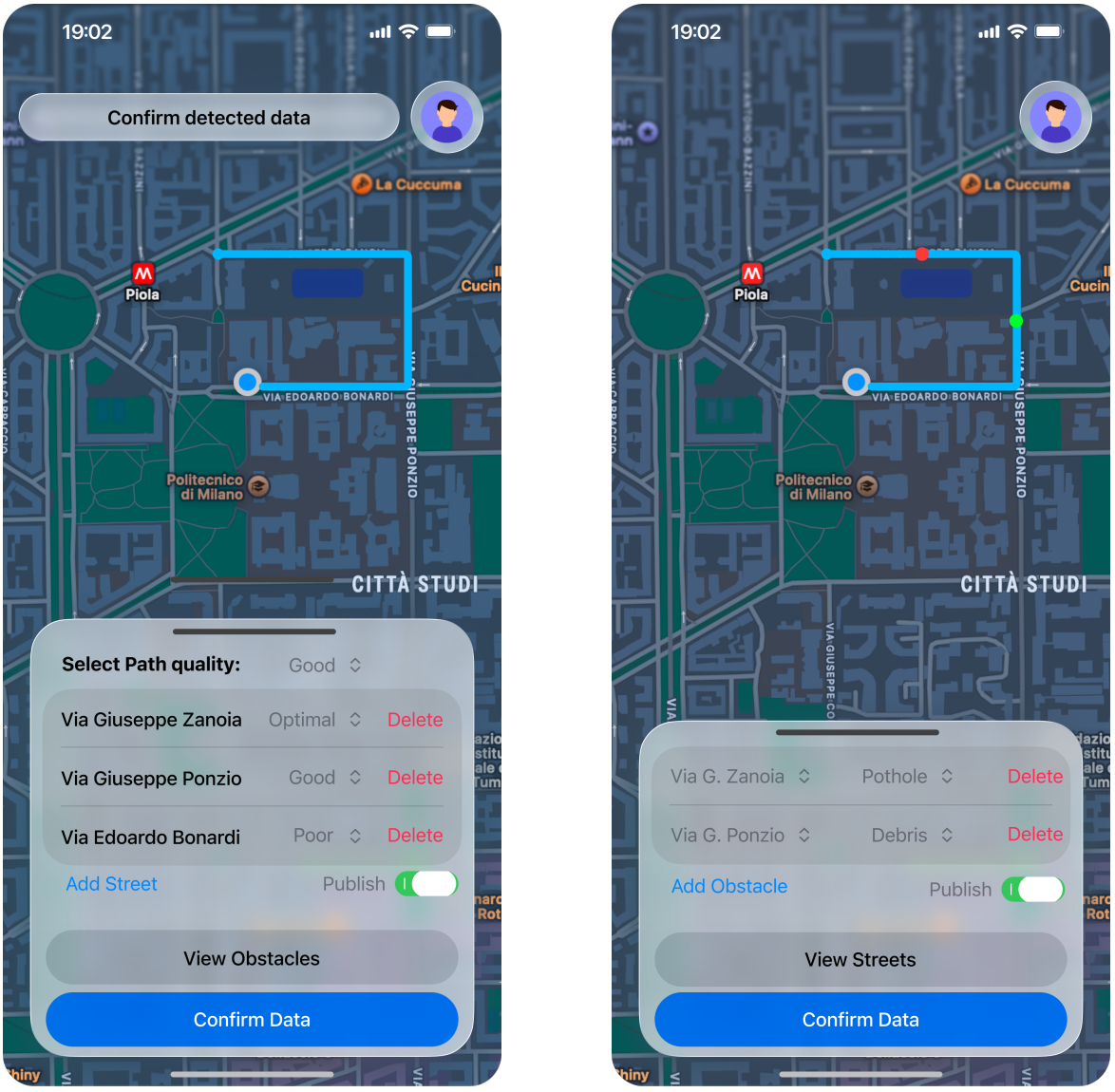


Figure 24: Authomatic path upload frame, you can pause or stop tracking

3.5.2 Confirm Screen



(a) Confirm Streets

(b) Confirm Obstacles

Figure 25: Confirm Data

3.6 Path Search

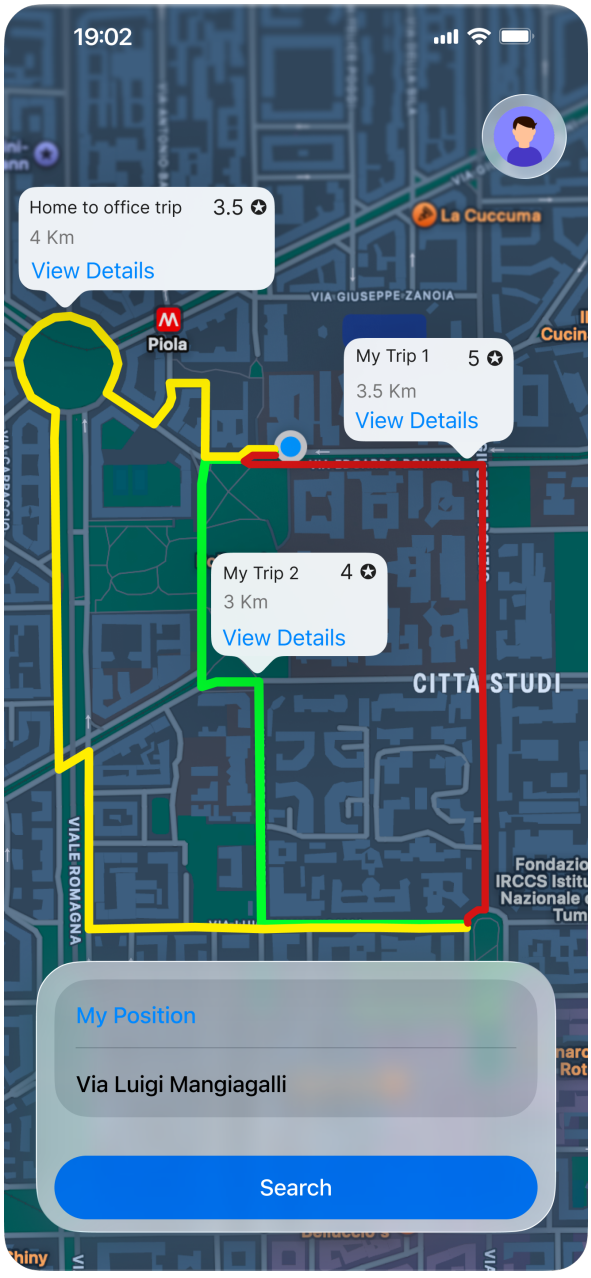


Figure 26: Search paths between two points

3.7 Path Details

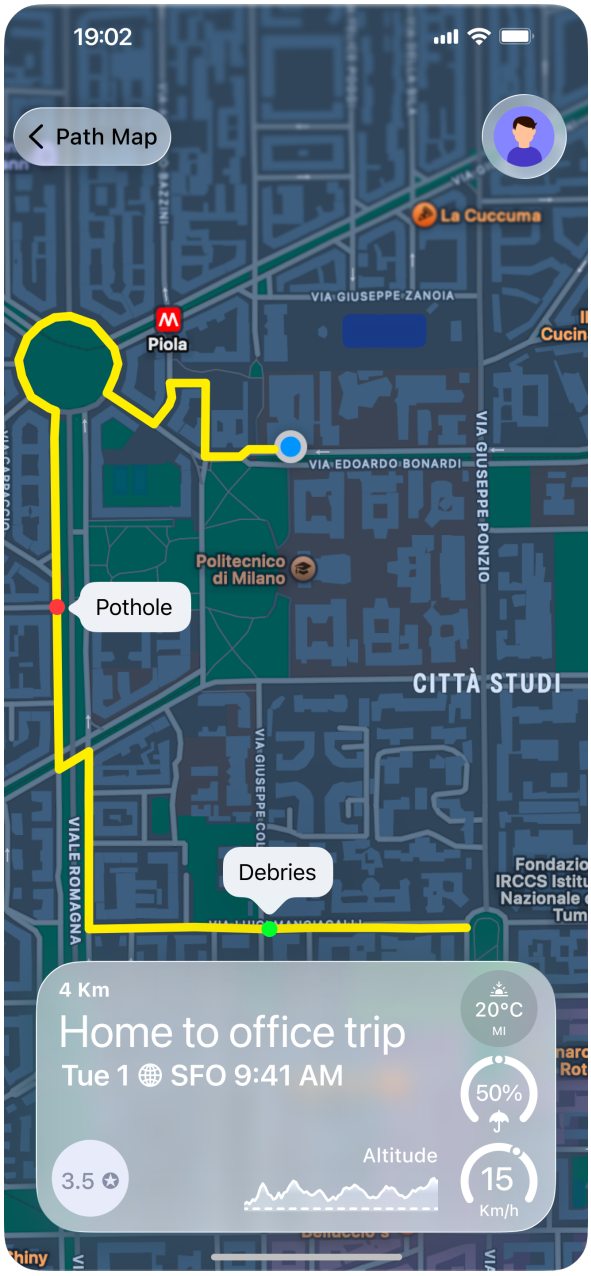


Figure 27: Detail screen for a single path

4 Requirement Traceability

This section provides a mapping between the functional requirements defined in the RASD and the system components described in the Design Document. It ensures that every requirement is addressed by the software architecture.

Req. ID	**Description**	**Component(s)**
G1 - User Account Management		
R1.1	Sign up with email, password, and personal details	**User Manager** (Sign Up Handler), **DBMS**
R1.2	Verify email uniqueness	**User Manager** (Sign Up Handler), **DBMS**
R1.3	User Login	**User Manager** (Login Handler), **DBMS**
R1.4	Modify profile and password	**User Manager** (Profile Handler), **DBMS**
G2 - Manual Trip Management		
R2.1	Create manual trip	**Path Manager** (Trip Creation Handler)
R2.2	Assign status to manual path	**Path Manager** (Trip Creation Handler)
R2.3	Add obstacles to manual trip	**Path Manager** (Trip Creation Handler)
R2.4	Save new trip to personal history	**Path Manager** (Trip Creation Handler), **DBMS**
R2.5	Delete saved trip	**Path Manager** (Personal Trip Retrieval), **DBMS**
G3 - Weather Data Integration		
R3.1	Query external Weather Service	**Path Manager** (Trip Creation Handler), **External Services** (Weather API)
R3.2	Retrieve meteorological data	**External Services** (Weather API)
R3.3	Associate weather data with trip	**Path Manager** (Trip Creation Handler), **DBMS**

Req. ID	Description	Component(s)
R3.4	Handle weather service unavailability	**Path Manager** (Trip Creation Handler)
R3.5	Prevent manual weather modification	**Path Manager** (Trip Creation Handler)
G4 - Automated Tracking		
R4.1	Sample GPS position	**Client Tier** (Mobile App)
R4.2	Collect accelerometer/gyroscope data	**Client Tier** (Mobile App)
R4.3	Analyze data for obstacles	**Client Tier** (Mobile App)
R4.4	Save detected obstacles for review	**Client Tier** (Mobile App)
R4.5	Background data acquisition	**Client Tier** (Mobile App)
R4.6	Calculate real-time metrics	**Client Tier** (Mobile App)
G5 - Post-Ride Validation		
R5.1	Display Post-Ride Review summary	**Client Tier** (Mobile App), **Path Manager** (Validation Handler)
R5.2	Confirm detected obstacle	**Path Manager** (Validation Handler)
R5.3	Discard obstacle (false positive)	**Path Manager** (Validation Handler)
R5.4	Modify obstacle type	**Path Manager** (Validation Handler)
R5.5	Visualize obstacles on map	**Client Tier** (Mobile App), **External Services** (Map Service)
G6 - Visibility Management		
R6.1	Change status Private to Public	**Path Manager** (Visibility Handler)
R6.2	Change status Public to Private	**Path Manager** (Visibility Handler)
R6.3	Trigger Scoring Engine on publication	**Path Manager** (Visibility Handler), **Scoring Engine** (Path Scorer)
G7 - Search Functionality		
R7.1	Search paths by origin/destination	**Search Manager** (Query Processing)

Req. ID	Description	Component(s)
R7.2	Display paths on interactive map	**Search Manager** (Path Retrieval), **External Services** (Map Service)
R7.3	Visualize Path Score and Status	**Search Manager** (Path Retrieval)
R7.4	View detailed path information	**Search Manager** (Path Retrieval), **Path Manager** (Detail Retrieval)
R7.5	Order results by Path Score	**Search Manager** (Score Sorting)
G8 - Path Scoring		
R8.1	Compute numerical Path Score	**Scoring Engine** (Path Scorer)
R8.2	Calculate weighted score	**Scoring Engine** (Segment Scorer)
R8.3	Trigger recalculation on update	**Scoring Engine** (Path Scorer)
R8.4	Use score to rank search results	**Search Manager** (Score Sorting)
G9 - Data Merging		
R9.1	Identify overlapping segments	**Scoring Engine** (OverlapDetector)
R9.2	Calculate consolidated Path Status	**Scoring Engine** (Segment Scorer)
R9.3	Manage data freshness	**Scoring Engine** (Segment Scorer)
R9.4	Mantain segment history	**Scoring Engine** (Segment Scorer), **DBMS**

5 Implementation, Integration and Test Plan

This chapter outlines the strategy for developing, integrating, and validating the *Best Bike Paths* system.

5.1 Implementation Strategy

The implementation will follow a **Bottom-Up** approach divided in many phases that we present here:

1. **Persistence Layer**: Setup of the PostgreSQL database. Implementation of the SQL schema.
2. **BBP Core Services**: Development of the internal logic components (*User Manager*, *Path Manager*, *Scoring Engine*, *Search Manager*).
3. **Client Application**: Development of the Mobile App and Web Application, connecting to the backend REST APIs. Integration of the external services.

5.2 BBP Core Services integration plan

Following the configuration of the Persistence Tier and the server infrastructure, we propose an implementation plan for the platform's core services (Application Tier). The figure 28 illustrates the development sequence, prioritizing functionalities from top to bottom.

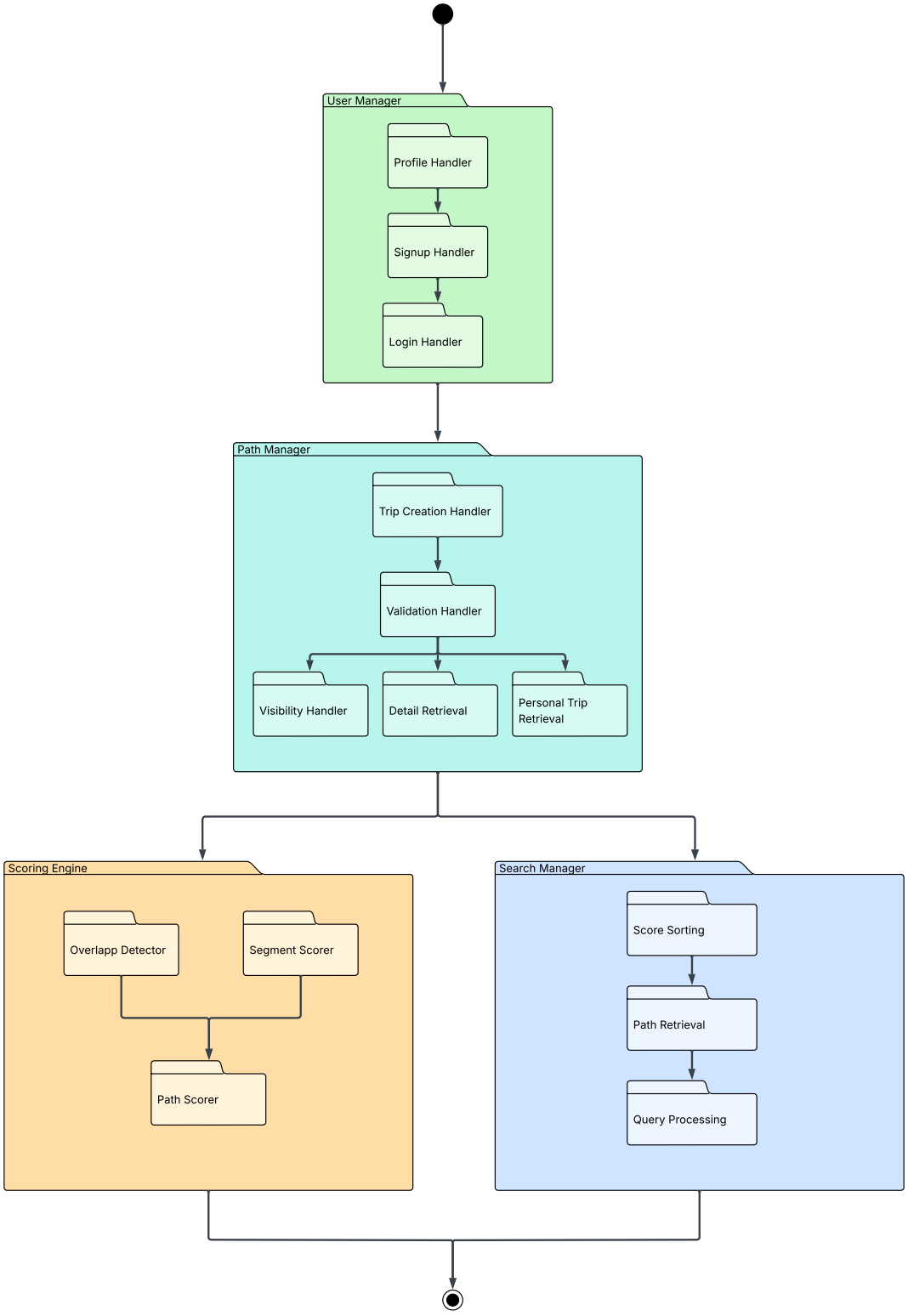


Figure 28: Core Services implementation plan

5.3 Testing Strategy

Verification of the BBP platform is required to confirm that all deployed features satisfy the project specifications. In order to achieve this, the following testing strategy can be followed:

Functional Testing: Each component of the system will be tested as a "Black box", in order to guarantee the functional correctness of the entire platform.

Performance Testing: The system will be tested with expected work load, to ensure its reliability also in real conditions.

Failure Testing: Failures can be generated to see how the system reacts and, eventually, corrects any unwanted behaviors.

Usability Testing: Mobile and Web Clients will be used by end-users to ensure the application meets their requirements and is sufficiently user friendly.

The usability testing mentioned above could be executed in these two phases:

- **Alpha Release**: Directed towards the **internal development team and key stakeholders** to validate architectural stability and logic integrity
- **Beta Release**: Targeted at a selected group of **early adopters and cycling enthusiasts**, crucial for testing the application in uncontrolled, real-world scenarios.

6 Effort Spent

Document Section	Giuliano Crescimbeni (h:m)	Luca De Nicola (h:m)
Introduction	–	–
Architectural Design	10:00	10:00
User Interface Design	7:00	7:00
Requirement Traceability	1	1
II&TP	2	2
Total Effort	**20:00**	**20:00**

Table 2: Effort spent by group members on each section of the document.

7 Software Used

Software Tool	Category	Usage in Project
GitHub	Version Control	Source code management and versioning.
Overleaf	Documentation	Collaborative LaTeX editing and compilation.
Lucidchart	Modeling	General Diagram design.
PlantText	Modeling	Designing Run Time View diagrams.
Figma	Design	UI Design.

Table 3: List of software tools used during development.