Best Bike Paths (BBP) project

POLITECNICO MILANO 1863

Design Document

Deliverable:	DD (Design Document)
Title:	Design Document
Authors:	Haipeng Zhu
Version:	2.0
Date:	January 10, 2026
Download page:	[https://github.com/hyperzhp/Zhu
Copyright:	Copyright © 2025, Haipeng Zhu – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this Design Document (DD) is to provide a comprehensive technical specification for the **Best Bike Paths (BBP)** system. While the Requirement Analysis and Specification Document (RASD) focused on the functional and non-functional requirements from a user's perspective, this document describes the software architecture, the decomposition into components, and the design choices made to satisfy those requirements.

The DD serves as a primary reference for the development team, providing the blueprints necessary for the implementation phase. It bridges the gap between the problem description (RASD) and the final source code.

1.2 Scope

The BBP system is a modular platform designed to support individual cyclists. The scope of this design document covers the following technical areas:

- **Mobile Client Application**: A cross-platform application responsible for user interaction, real-time GPS tracking, and path visualization.
- **Backend Infrastructure**: A RESTful web service handling authentication, data persistence, and the business logic for calculating path safety scores.
- **External Integrations**: The interfaces with third-party Map Providers (for rendering tiles and routing) and Meteorological Services (for atmospheric data enrichment).

1.3 Definitions, Acronyms, Abbreviations

- **API**: Application Programming Interface.
- **REST**: Representational State Transfer, an architectural style for distributed hypermedia systems.
- **DTO**: Data Transfer Object, an object that carries data between processes.
- **DAO**: Data Access Object, an object that provides an abstract interface to some type of database.
- **JWT**: JSON Web Token, a compact, URL-safe means of representing claims to be transferred between two parties.
- **DBMS**: Database Management System.
- **HTTPS**: Hypertext Transfer Protocol Secure.

1.4 Revision History

Version	Date	Description
1.0	January 8, 2026	Initial submission of the Design Document.
2.0	January 9, 2026	**Major Revision (Post-Review)**: Comprehensive upgrade of architectural diagrams, UI fidelity, and algorithm specifications. Detailed changes are listed in Section 1.4.1.

Table 1: Revision history of the DD

1.4.1 Changelog: Enhancements in Version 2.0

This version represents a significant iteration over the initial submission, focusing on technical depth and architectural clarity. The key improvements include:

- **Architectural Refinement**:
  - Upgraded **Component** and **Deployment Diagrams** to include specific protocols (HTTPS, JDBC), port numbers, and execution environments (Docker), replacing generic blocks with industry-standard notations.
  - Introduced explicit interfaces (APIs) and defined the "Offline-First" synchronization logic.
- **Advanced Runtime Logic**:
  - Rewrote all **Sequence Diagrams (UC1-UC6)** to include error handling (`alt` blocks), asynchronous processing (`par/group`), and security validations (`ref` to UC5), moving beyond simple "happy paths".
  - Added a detailed **State Diagram** covering the complex lifecycle of the "Safety Score" visibility.
- **High-Fidelity User Interface**:
  - Replaced wireframes with **High-Fidelity Mockups** featuring real Milan geospatial data (Polimi $\rightarrow$ Duomo route).
  - Implemented "Situational Design" principles with high-contrast modes and specific safety alerts (e.g., Tram Tracks).
- **Technical Specifications**:
  - Added the **Safety Score Algorithm** pseudocode (Section 2.9) to mathematically define how user reports are aggregated.
  - Included specific JSON Data Contracts for the core RESTful endpoints.

1.5 Reference Documents

- Politecnico di Milano, *Assignment RDD AY 2024-2025 - Best Bike Paths.*
- Haipeng Zhu, *Best Bike Paths - Requirement Analysis and Specification Document (RASD)*, 2024.
- IEEE Std 1016-2009, *IEEE Standard for Information Technology - Systems Design - Software Design Descriptions.*

1.6 Document Structure

The document is organized into the following main sections:

- **Section 2 - Architectural Design**: Provides a high-level overview of the system architecture, component views, deployment strategies, and runtime behavior.
- **Section 3 - User Interface Design**: Describes the user experience and provides mockups for the mobile application.

- **Section 4 - Requirements Traceability**: Maps each functional requirement from the RASD to specific design elements in the DD.
- **Section 5 - Implementation, Integration & Testing**: Outlines the development plan and the strategy for verifying system correctness.

Appendix: Use of AI Tools

In accordance with the academic integrity guidelines, the following AI tools were utilized to assist in the drafting and refinement of this Design Document.

1. Text Refinement & Content Structuring

- **Tool**: Large Language Models (ChatGPT-4 / Gemini)
- **Usage**: Used to refine the technical English phrasing of the "Architectural Design" and "Design Rationale" sections to ensure professional tone and clarity.
- **Prompt Example**: *"Refine the following description of the Offline-First strategy to highlight data consistency and synchronization logic."*

2. Diagram Generation Code

- **Tool**: LLM for PlantUML Generation
- **Usage**: Generated the initial PlantUML code structures for Sequence Diagrams (UC1-UC6) and the Component Diagram based on the logical flow descriptions.
- **Prompt Example**: *"Generate a PlantUML sequence diagram for a User Login flow implementing JWT stateless authentication with error handling."*

3. User Interface Prototyping

- **Tool**: AI-assisted Code Generation (Tailwind CSS / HTML)
- **Usage**: Generated the HTML/CSS code to visualize the high-fidelity UI mockups for the Dashboard and Recording screens, which were then rendered and included as figures.
- **Rationale**: This allowed for rapid prototyping of the "Situational Design" paradigm without manual pixel-pushing in vector tools.

2 Architectural Design

2.1 Overview

The **Best Bike Paths (BBP)** system is engineered as a distributed application adhering to the **3-tier Client-Server architectural pattern**. This choice is driven by the need to decouple the high-frequency interaction logic (mobile client) from the heavy analytical processing (backend).

The system architecture is organized into the following logical layers:

- **Presentation Tier (Mobile Client)**: This layer acts as the primary interface for cyclists. It is responsible for real-time sensor management (GNSS/GPS), local data buffering to support the "Offline-First" strategy, and the rendering of vector-based map tiles.

- **Application Tier (Backend Server)**: Hosted on a cloud infrastructure, this tier exposes a RESTful API. It orchestrates business processes, validates user credentials via JWT, and executes the core "Safety Score" algorithm by aggregating crowdsourced data.

- **Data Tier (Persistence)**: This layer ensures data durability and consistency. It leverages a relational database (PostgreSQL) enhanced with geospatial extensions (PostGIS) to efficiently index and query spatiotemporal data (trajectories) and topological data (road segments).

2.2 Component View

The static decomposition of the system is illustrated in Figure 1. The design follows a **Service-Oriented** approach within the backend to ensure modularity.

2.2.1 Key Component Responsibilities

- **Mobile Client**:

  - **Trip Manager**: The central controller for the recording session. It implements a 1Hz sampling loop and manages the transition between "Recording", "Paused", and "Stopped" states.
  - **Local Cache**: A lightweight SQLite database that acts as a buffer for trip data, ensuring no data is lost during temporary network outages.

- **Backend Server**:

  - **API Gateway**: The single entry point for all client requests. It handles SSL termination, rate limiting, and routes requests to the appropriate internal controller.
  - **Safety Scorer**: An analytical component that runs scheduled jobs to update the safety index of road segments based on recent user reports (e.g., last 30 days).
  - **Weather Integrator**: A specialized adapter that communicates with external Meteorological APIs asynchronously to enrich trip data without blocking the main response thread.

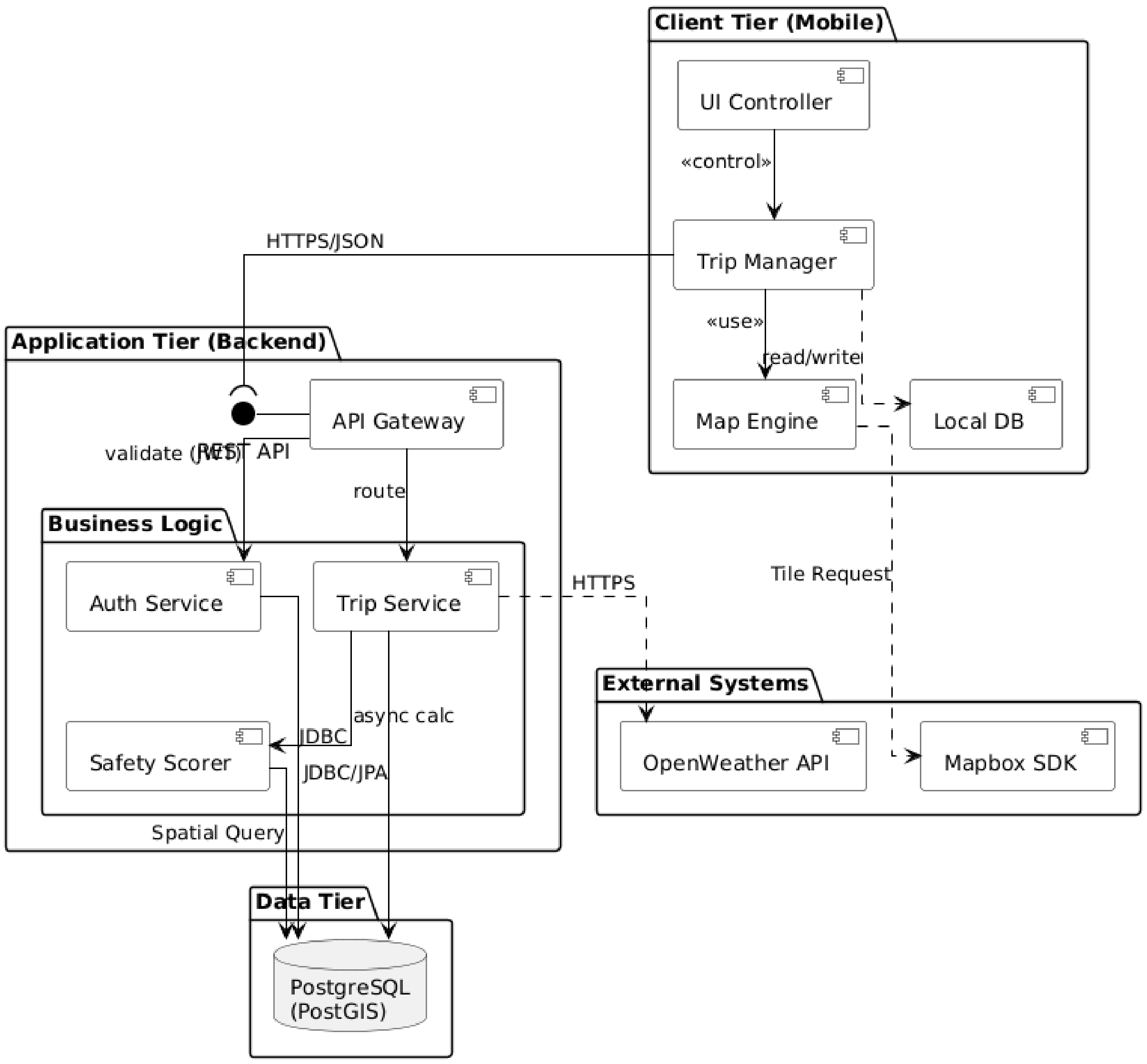


Figure 1: BBP Component Diagram detailing tier decomposition and service dependencies

2.3 Deployment View

The physical deployment, shown in Figure 2, is designed for high availability and security.

- **Execution Environment**: The backend services are containerized (e.g., via Docker) to ensure consistency between development and production environments.
- **Network Isolation**: The Database Server is deployed in a **Private Subnet**, inaccessible directly from the public internet. Only the Application Server can establish a connection via a secure JDBC link.
- **Scalability**: The stateless nature of the Application Tier allows for horizontal scaling (adding more server instances) behind a Load Balancer to handle peak cycling traffic.

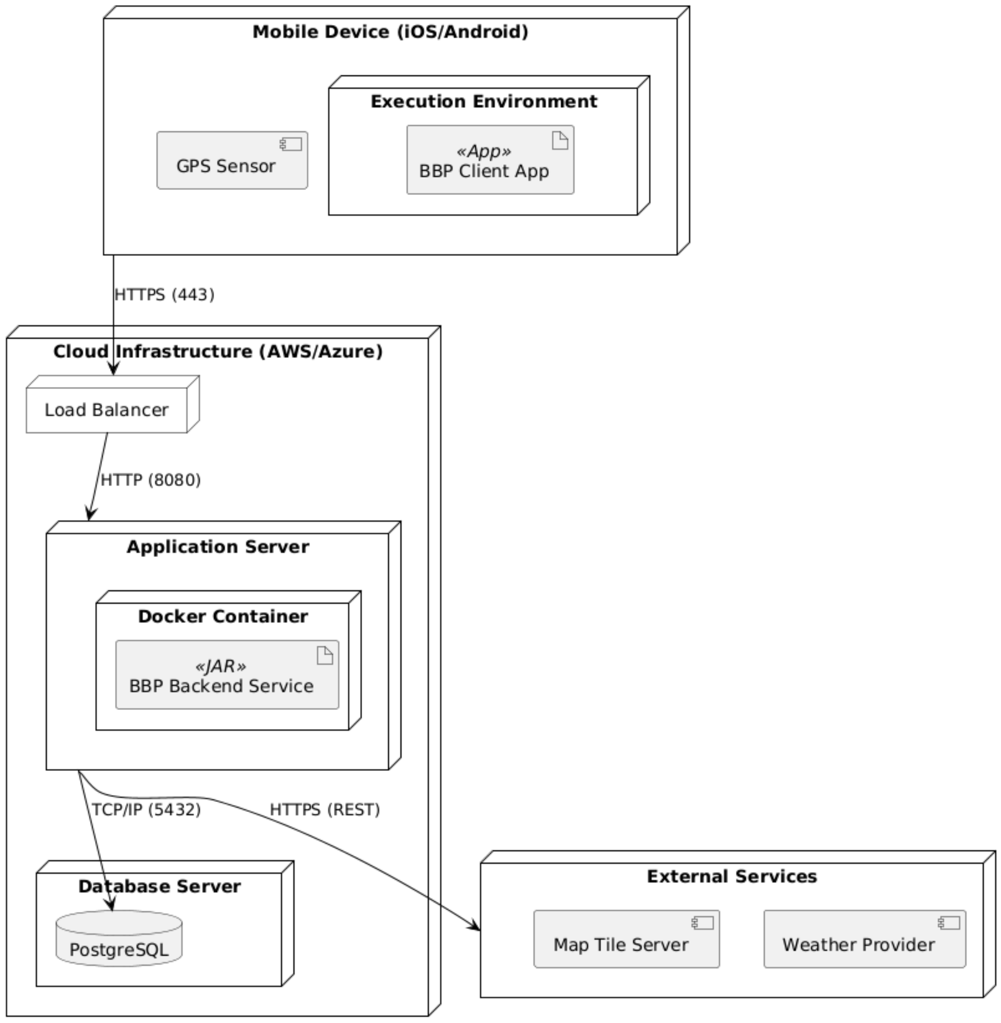


Figure 2: BBP Deployment Diagram illustrating mobile-to-cloud mapping

Communication Protocols The nodes interact using the following protocols:

- **HTTPS/TLS 1.3**: Used for all communication between the Mobile App and the Application Server to ensure data confidentiality and integrity.
- **JDBC/SSL**: Secure connection protocol used for internal communication between the Application Server and the Database.
- **REST/JSON**: The architectural style and data format used for API requests and responses.

2.4 Data Design

The logical data structure is derived directly from the domain entities. Figure 3 represents the relational schema, with specific focus on the persistence of `GeoPoints` which constitute the foundation of trip trajectories and bike path definitions.

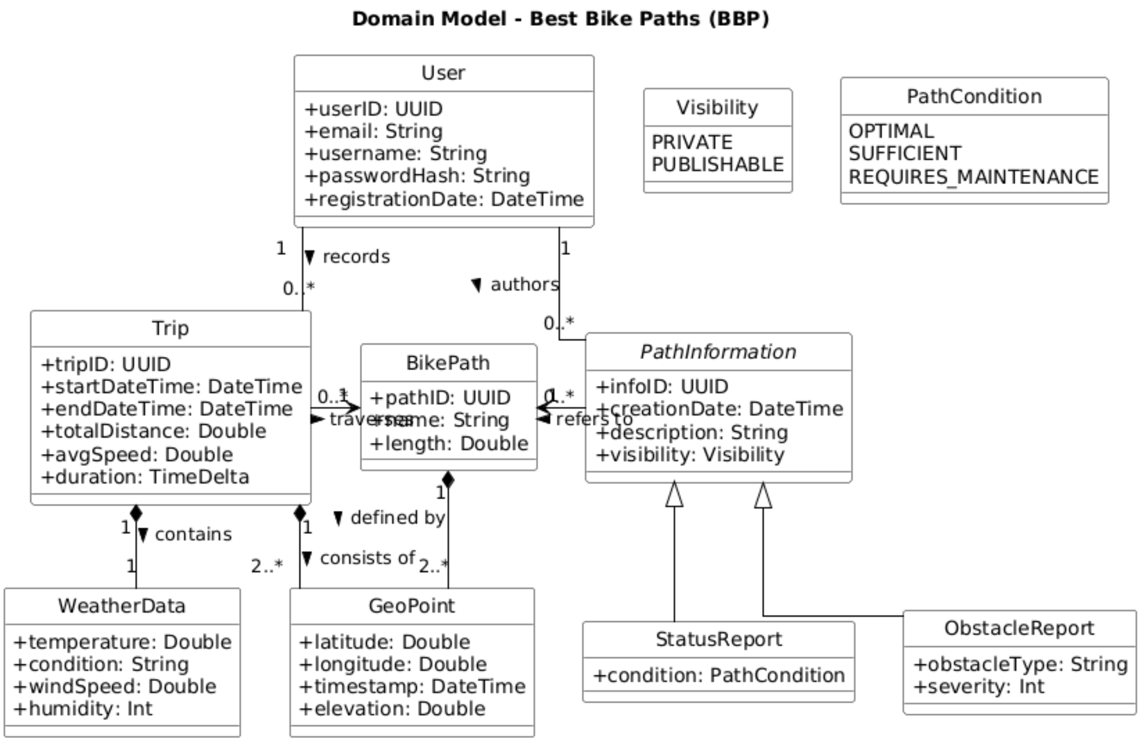


Figure 3: Logical Data Model highlighting geospatial and user relationships

2.5 Runtime View

This section describes the dynamic behavior of the system, illustrating how components collaborate over time and how the system manages the lifecycle of core entities.

2.5.1 Entity Lifecycles

The system enforces strict state transitions for trips and reports to maintain data consistency. Figure 4 details the recording lifecycle, emphasizing the asynchronous weather enrichment phase.

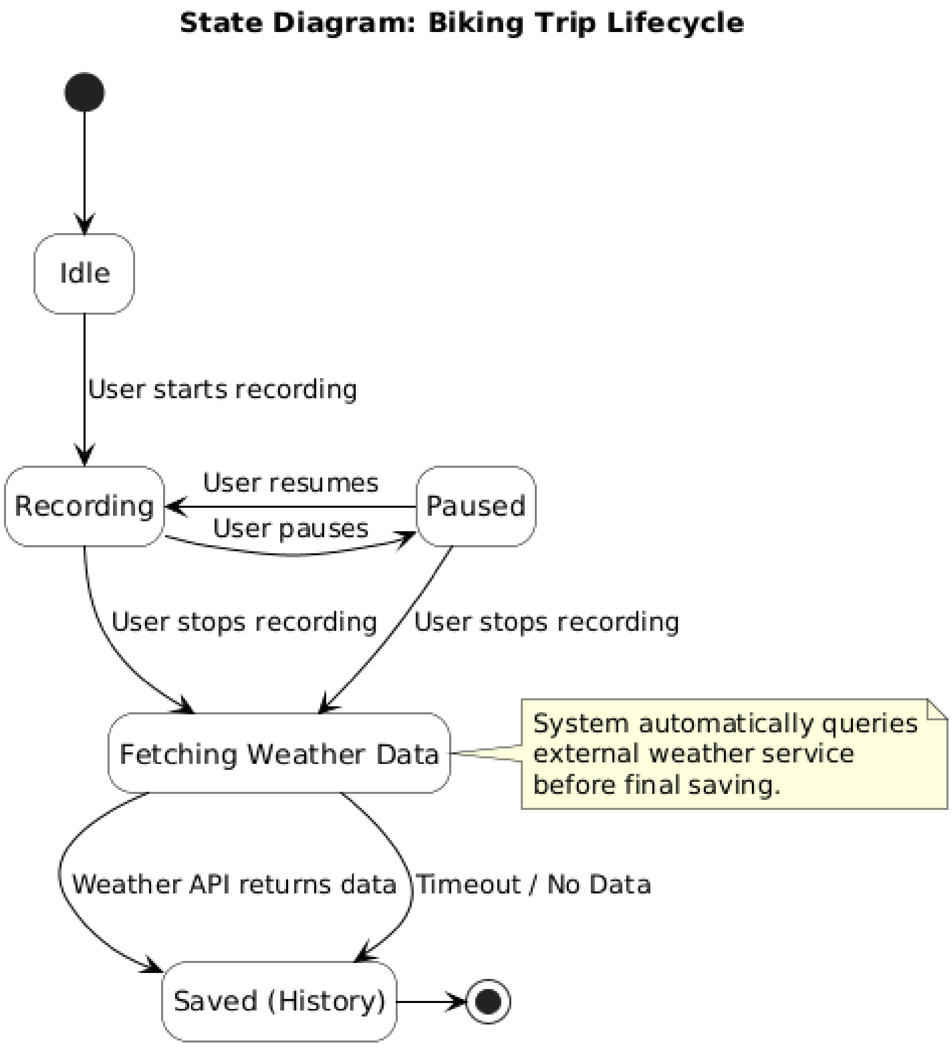


Figure 4: State Diagram: Biking Trip Lifecycle and asynchronous data enrichment

Figure 5 illustrates the privacy-preserving logic for user reports, allowing users to control information visibility.

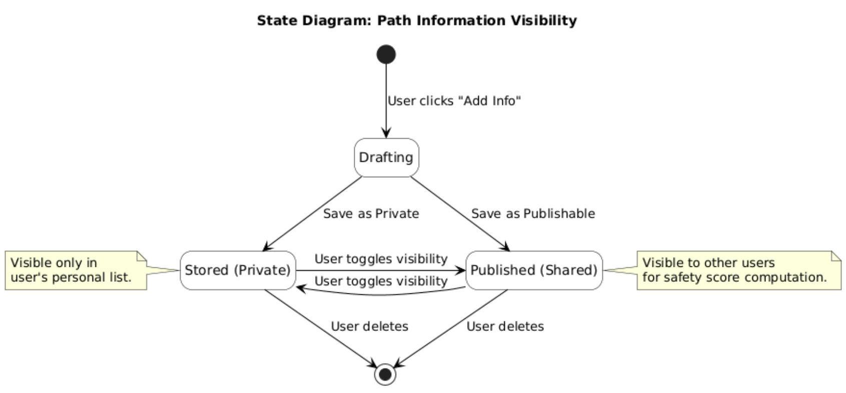


Figure 5: State Diagram: Path Information Visibility and sharing logic

2.5.2 Component Interaction Sequences

The following sequence diagrams demonstrate the runtime interaction between the Client, API Gateway, and Backend Services to fulfill primary use cases.

UC1: Record a Biking Trip Figure 6 depicts the core value stream of the application. The sequence highlights the system's resilience to network instability and its focus on performance.

- **Local Buffering**: As the user rides, the *GPS Sensor* pushes coordinates to the *Trip Manager*. Instead of transmitting each point immediately (which would drain battery and require constant connectivity), the data is buffered in the local SQLite storage.
- **Atomic Upload**: Upon finishing the trip, the trajectory is serialized into a `TrajectoryDTO` and uploaded in a single HTTP POST request.
- **Asynchronous Enrichment**: To provide immediate UI feedback, the backend acknowledges the upload (`HTTP 201`) immediately. The fetching of weather data from the external API is handled as a background background task, decoupling external latency from the user experience.

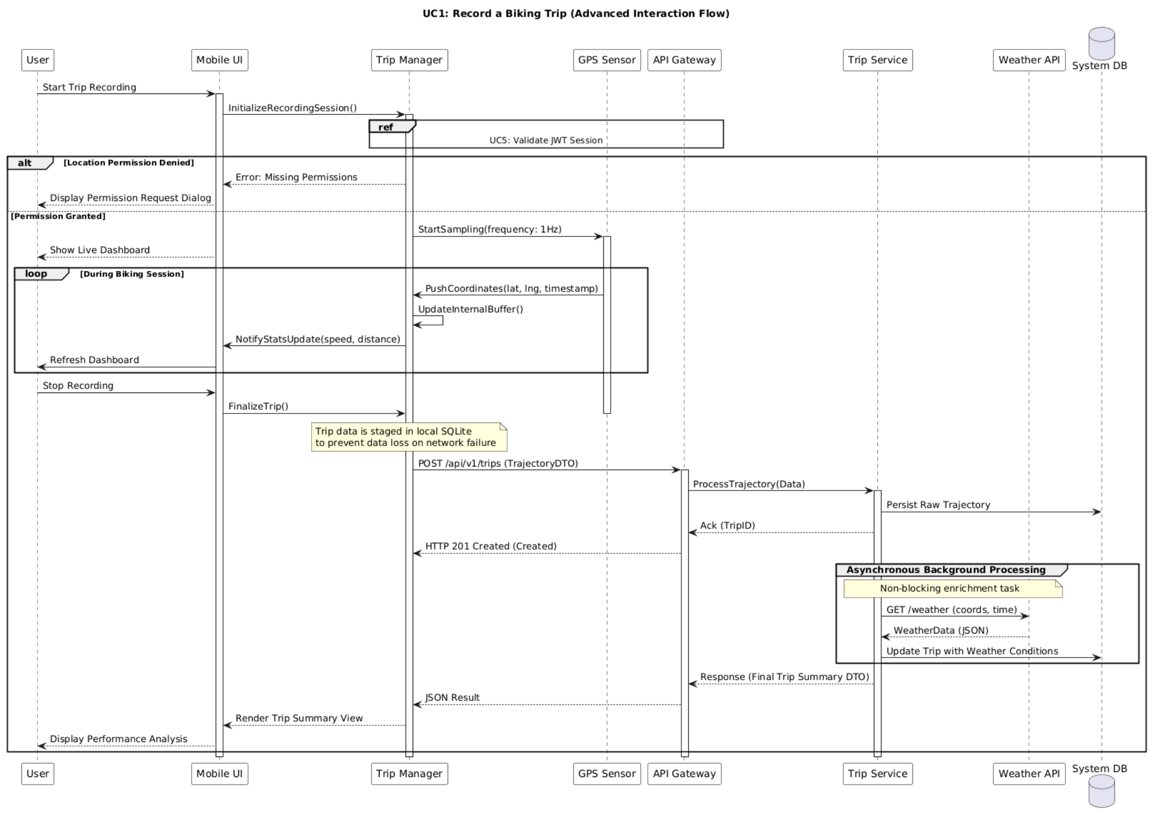

UC1: Record a Biking Trip (Advanced Interaction Flow)
User
Mobile UI
Trip Manager
GPS Sensor
API Gateway
Trip Service
Weather API
System DB
Start Trip Recording
InitializeRecordingSession()
ref
UC5: Validate JWT Session
alt [Location Permission Denied]
Error: Missing Permissions
Display Permission Request Dialog
[Permission Granted]
StartSampling(frequency: 1Hz)
Show Live Dashboard
loop [During Biking Session]
PushCoordinates(lat, lng, timestamp)
UpdateInternalBuffer()
NotifyStatsUpdate(speed, distance)
Refresh Dashboard
Stop Recording
FinalizeTrip()
Trip data is staged in local SQLite to prevent data loss on network failure
POST /api/v1/trips (TrajectoryDTO)
ProcessTrajectory(Data)
Persist Raw Trajectory
Ack (TripID)
HTTP 201 Created (Created)
Asynchronous Background Processing
Non-blocking enrichment task
GET /weather (coords, time)
WeatherData (JSON)
Update Trip with Weather Conditions
Response (Final Trip Summary DTO)
JSON Result
Render Trip Summary View
Display Performance Analysis

Figure 6: Sequence Diagram: Recording a Biking Trip (UC1)

UC2: Insert Path Information The reporting flow (Figure 7) demonstrates how the system ensures high data quality and respects user privacy.

- **Client-Side Validation**: The *Report Manager* validates mandatory fields (e.g., obstacle type) before the request leaves the device, reducing unnecessary load on the server.

- **Privacy Gate**: When the *Path Service* receives the report, it checks the visibility flag. Private reports are stored with a flag that excludes them from the global Safety Score aggregation algorithm, ensuring that user notes remain personal if desired.

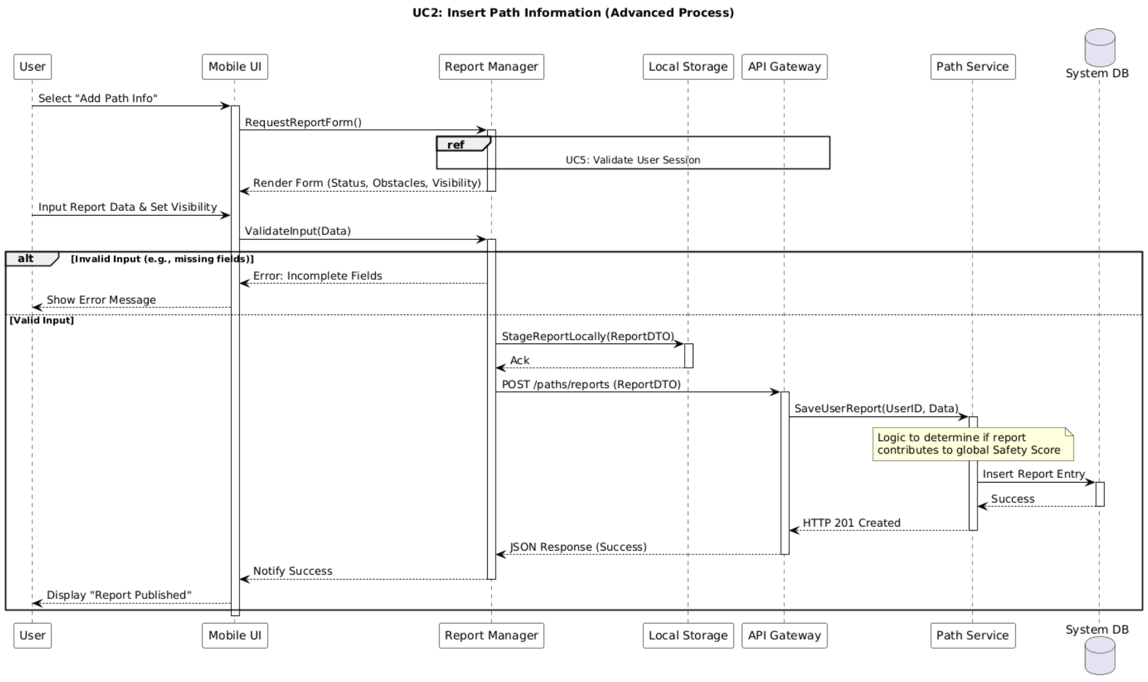

UC2: Insert Path Information (Advanced Process)
User
Mobile UI
Report Manager
Local Storage
API Gateway
Path Service
System DB
Select "Add Path Info"
RequestReportForm()
ref
UC5: Validate User Session
Render Form (Status, Obstacles, Visibility)
Input Report Data & Set Visibility
ValidateInput(Data)
alt
[Invalid Input (e.g., missing fields)]
Error: Incomplete Fields
Show Error Message
[Valid Input]
StageReportLocally(ReportDTO)
Ack
POST /paths/reports (ReportDTO)
SaveUserReport(UserID, Data)
Logic to determine if report contributes to global Safety Score
Insert Report Entry
Success
HTTP 201 Created
JSON Response (Success)
Notify Success
Display "Report Published"

Figure 7: Sequence Diagram: Submitting Manual Path Reports (UC2)

UC3: Visualize Bike Paths Figure 8 illustrates the complex orchestration required to render safe paths. This is a read-heavy operation that combines external topology with internal analytics.

- **Data Aggregation**: The *API Gateway* acts as a facade, fetching geometric routes from the *External Map Provider* while simultaneously querying the internal *Safety Scorer* for metadata.

- **Temporal Filtering**: The scoring logic queries the database for reports within a specific time window (e.g., the last 30 days) to ensure the "Safety Score" reflects current road conditions rather than outdated history.

- **Visual Coding**: The result is a GeoJSON object enriched with color-coded properties (Green/Yellow/Red), which the mobile *Map Engine* renders as an overlay.

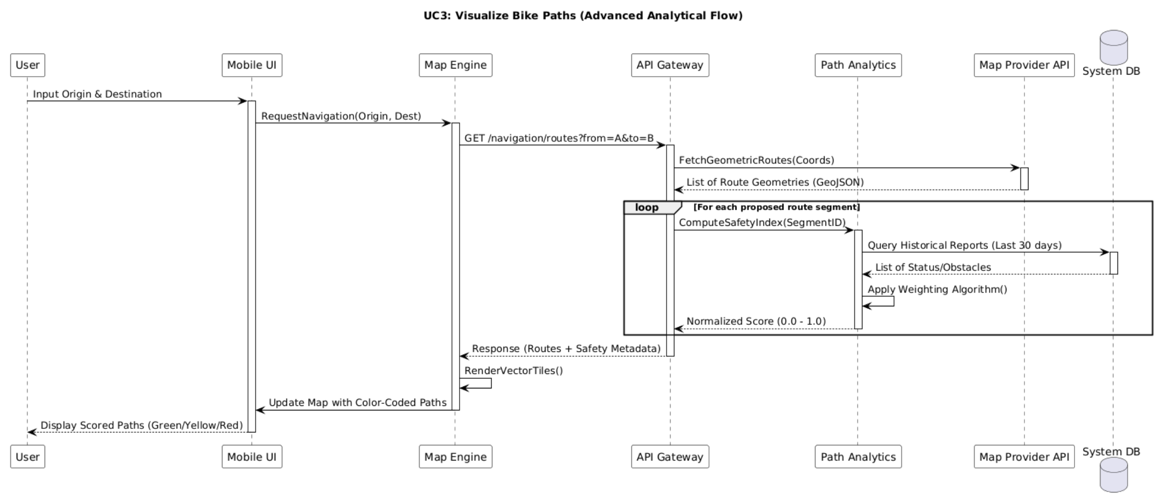

UC3: Visualize Bike Paths (Advanced Analytical Flow)
User
Mobile UI
Map Engine
API Gateway
Path Analytics
Map Provider API
System DB
Input Origin & Destination
RequestNavigation(Origin, Dest)
GET /navigation/routes?from=A&to=B
FetchGeometricRoutes(Coords)
List of Route Geometries (GeoJSON)
loop [For each proposed route segment]
ComputeSafetyIndex(SegmentID)
Query Historical Reports (Last 30 days)
List of Status/Obstacles
Apply Weighting Algorithm()
Normalized Score (0.0 - 1.0)
Response (Routes + Safety Metadata)
RenderVectorTiles()
Update Map with Color-Coded Paths
Display Scored Paths (Green/Yellow/Red)

Figure 8: Sequence Diagram: Visualizing Bike Paths with Safety Scores (UC3)

UC4: User Registration The registration process (Figure 9) is designed to enforce security best practices from the ground up.

- **Data Integrity**: The *Auth Service* first checks the database to ensure the email is unique, returning a `409 Conflict` if it already exists.
- **Secure Storage**: Passwords are never stored in plain text. The diagram explicitly shows a hashing step (using algorithms like Argon2 or BCrypt) before persistence. This ensures that even in the event of a database breach, user credentials remain protected.

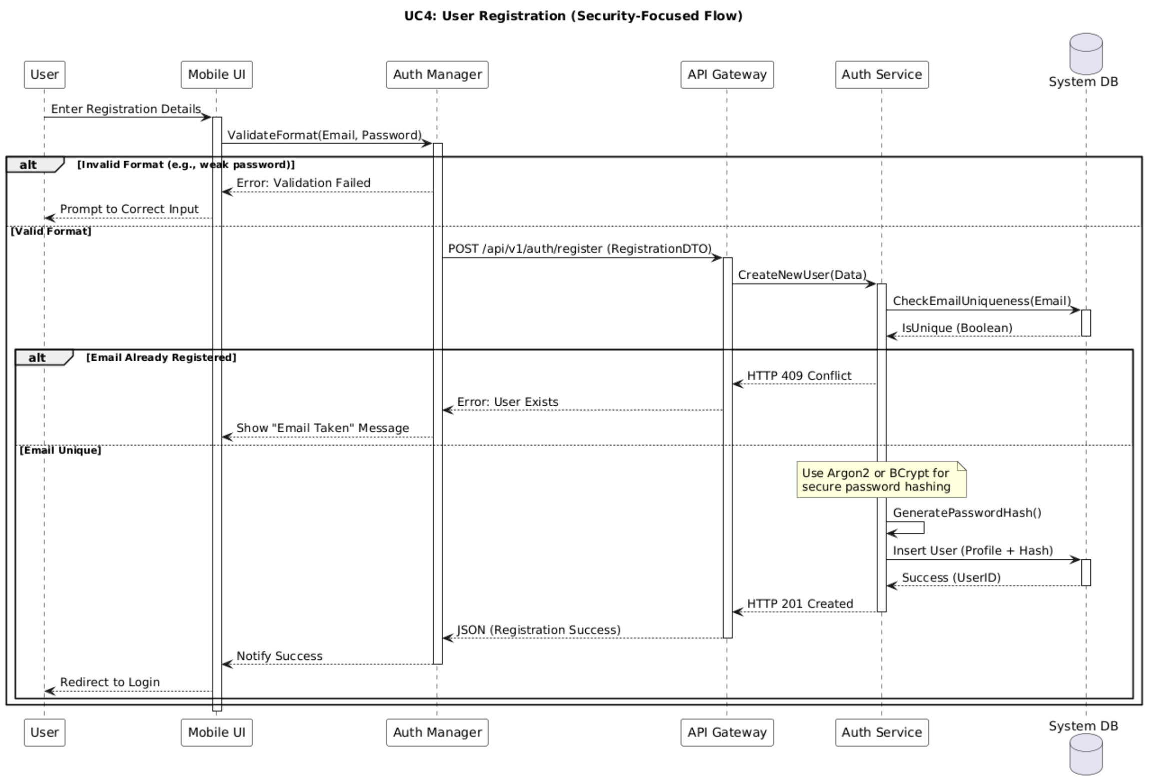
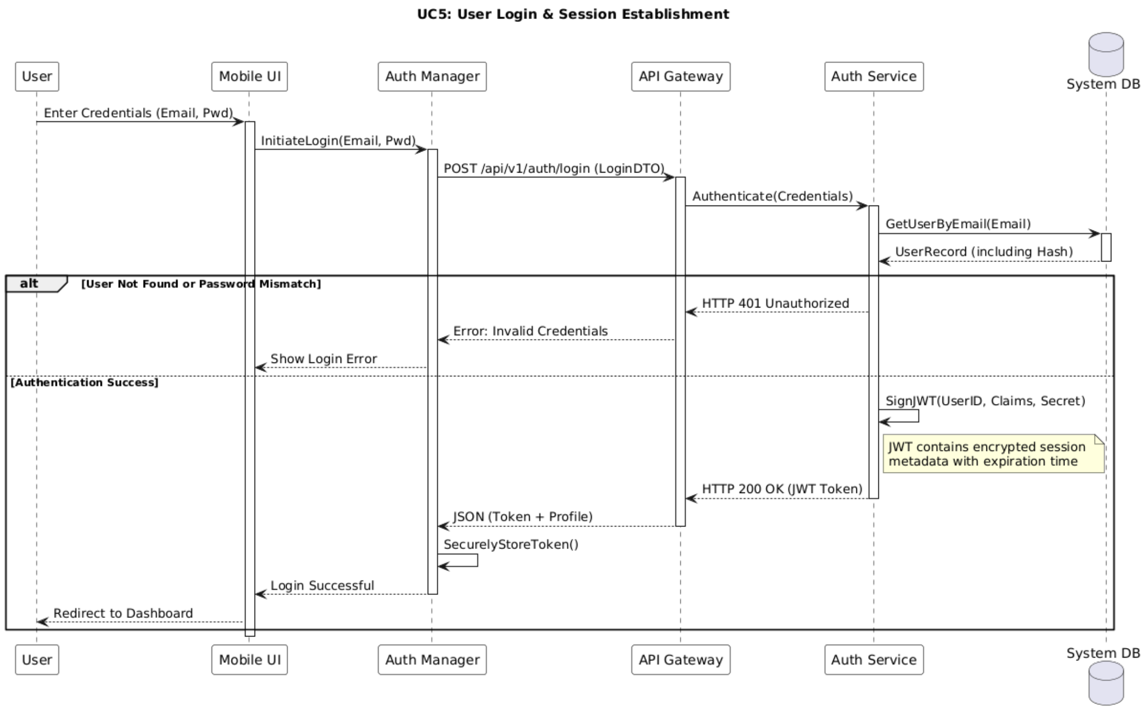

UC4: User Registration (Security-Focused Flow)
User
Mobile UI
Auth Manager
API Gateway
Auth Service
System DB
Enter Registration Details
ValidateFormat(Email, Password)
alt [Invalid Format (e.g., weak password)]
Error: Validation Failed
Prompt to Correct Input
[Valid Format]
POST /api/v1/auth/register (RegistrationDTO)
CreateNewUser(Data)
CheckEmailUniqueness(Email)
IsUnique (Boolean)
alt [Email Already Registered]
HTTP 409 Conflict
Error: User Exists
Show "Email Taken" Message
[Email Unique]
Use Argon2 or BCrypt for secure password hashing
GeneratePasswordHash()
Insert User (Profile + Hash)
Success (UserID)
HTTP 201 Created
JSON (Registration Success)
Notify Success
Redirect to Login

Figure 9: Sequence Diagram: User Registration (UC4)

UC5: User Login Figure 10 details the authentication mechanism, which serves as the foundation for all protected interactions.

- **Stateless Architecture**: Upon successful credential verification, the server generates a **JSON Web Token (JWT)**. This token contains encrypted claims (user ID, expiration) and is signed by the server's private key.
- **Session Management**: By returning this token to the client, the server avoids maintaining a session state in memory (Sessionless), allowing the backend to scale horizontally without sticky sessions.



UC5: User Login & Session Establishment
User
Mobile UI
Auth Manager
API Gateway
Auth Service
System DB
Enter Credentials (Email, Pwd)
InitiateLogin(Email, Pwd)
POST /api/v1/auth/login (LoginDTO)
Authenticate(Credentials)
GetUserByEmail(Email)
UserRecord (including Hash)
alt [User Not Found or Password Mismatch]
HTTP 401 Unauthorized
Error: Invalid Credentials
Show Login Error
[Authentication Success]
SignJWT(UserID, Claims, Secret)
JWT contains encrypted session metadata with expiration time
HTTP 200 OK (JWT Token)
JSON (Token + Profile)
SecurelyStoreToken()
Login Successful
Redirect to Dashboard

Figure 10: Sequence Diagram: User Login (UC5)

UC6: View Trip History The history retrieval flow (Figure 11) is optimized for mobile data bandwidth.

- **Two-Phase Retrieval**: To prevent fetching megabytes of GPS coordinates for the entire history list, the system implements a "Summary First" pattern. The initial request fetches only metadata (Date, Distance, Score).
- **On-Demand Details**: The heavy trajectory data (GeoJSON) is fetched only when the user explicitly taps on a specific trip card. This *Lazy Loading* strategy significantly reduces the initial load time and data usage.

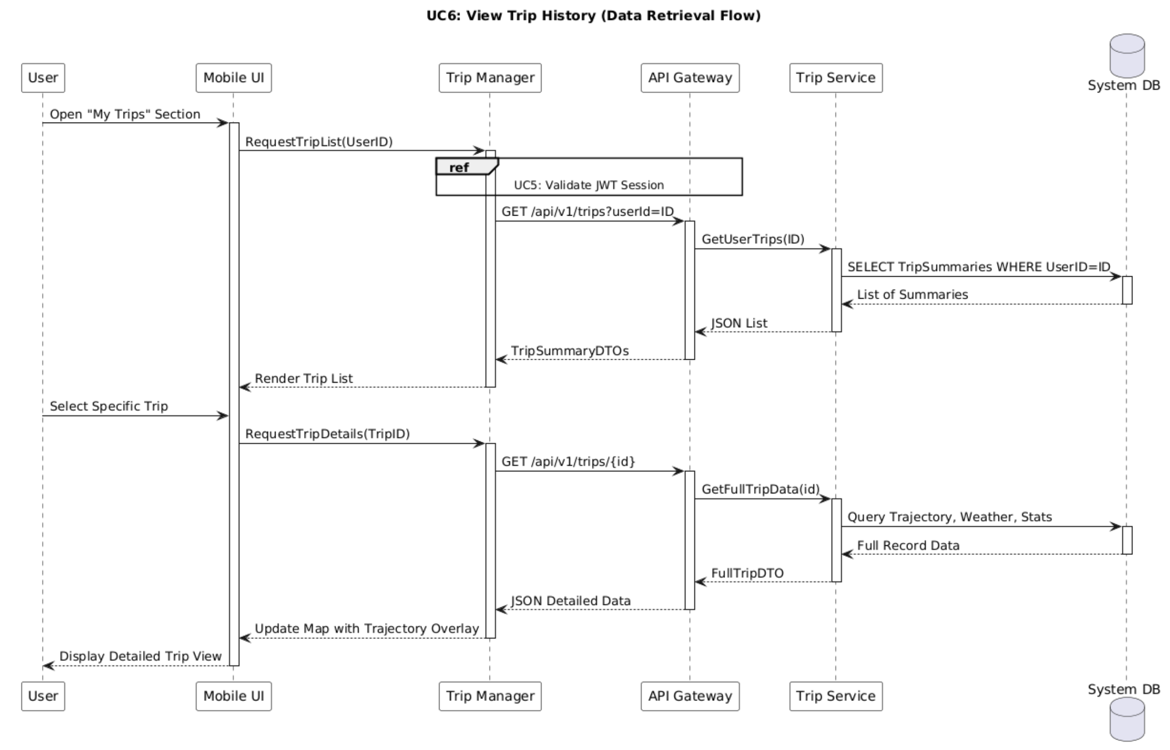


Figure 11: Sequence Diagram: Accessing Personal Trip History (UC6)

2.6 Component Interfaces

The BBP Backend exposes a RESTful API to the mobile client. The interfaces are designed to be stateless and resource-oriented.

Endpoint	Verb	Description / Payload
`/auth/login`	POST	Authenticates user; returns a JWT token.
`/trips/upload`	POST	Accepts a JSON array of coordinates and metrics.
`/paths/info`	POST	Submits road status reports or obstacle alerts.
`/map/scored-paths`	GET	Returns paths within a coordinate bounding box with safety scores.

Table 2: Core RESTful API Endpoints

2.7 Selected Architectural Styles and Patterns

2.7.1 Client-Server Architecture

Choice: We adopted a 3-tier Client-Server architecture (Presentation, Application, Data). **Rationale**: This style allows for independent scalability. The backend can be scaled horizontally on cloud infrastructure without requiring updates to the mobile client binary. Furthermore, it centralizes sensitive business logic (like the Safety Score algorithm) to prevent client-side tampering, ensuring data integrity.

2.7.2 RESTful API

Choice: Communication is handled via stateless REST APIs over HTTPS. **Rationale**: REST is universally supported by mobile platforms (iOS/Android) and allows for caching of non-volatile data (e.g., map tiles), reducing bandwidth consumption for the user in outdoor environments.

2.7.3 Layered Pattern

Choice: The backend is structured into Controller, Service, and Repository layers. **Rationale**: This strict separation of concerns enhances maintainability. If the database technology changes (e.g., migrating from PostgreSQL to Oracle), only the Repository layer needs modification, leaving the Business Logic (Service layer) intact.

2.8 Other Design Decisions

2.8.1 Offline-First Strategy

Given the outdoor nature of cycling, network connectivity is often unstable. To mitigate data loss, the system prioritizes data availability:

- **Local Buffering**: GPS coordinates are written to a local SQLite database immediately upon capture. Synchronization with the server occurs in the background only when a stable connection is detected.
- **Optimistic UI**: When a user submits a report, the UI updates immediately as if the request succeeded, while the actual network request is queued. This ensures a fluid user experience even in poor network conditions.

2.8.2 Geospatial Data Optimization

To handle heavy geospatial queries efficiently, we utilize **PostGIS** indexing (R-Tree). This ensures that querying "all safe paths within a 5km radius" remains performant ($O(\log n)$) even as the dataset grows to millions of records.

3 User Interface Design

4 User Interface Design

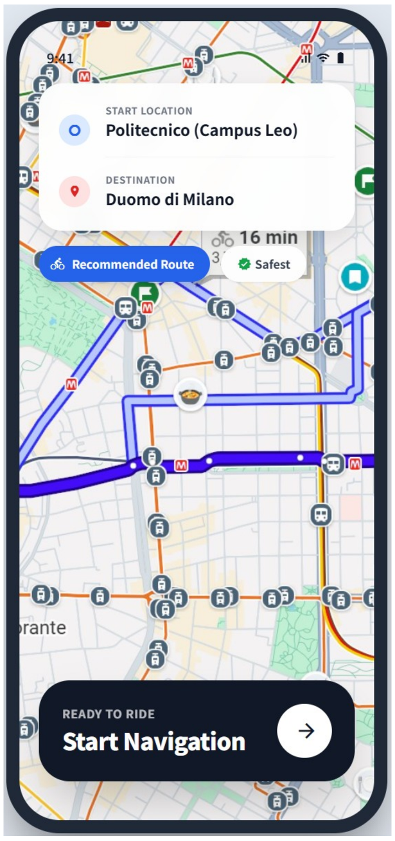
4.1 Design Rationale

The User Interface (UI) of the **Best Bike Paths (BBP)** application follows the **"Situational Design"** paradigm. Recognizing that cyclists interact with the app outdoors in Milan's busy streets, the design prioritizes:

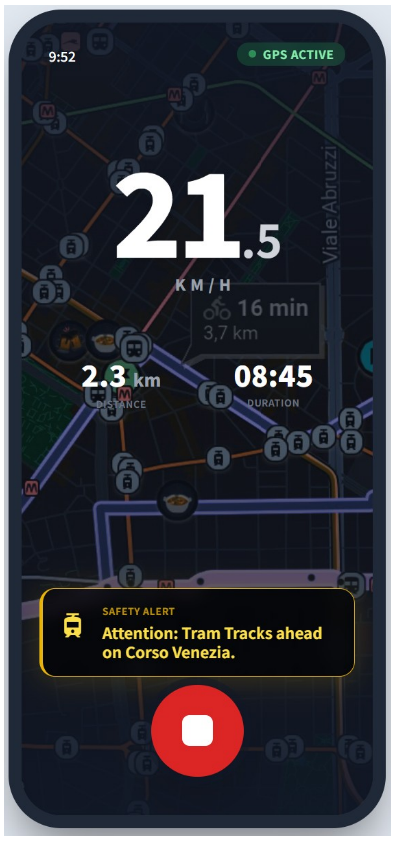
- **Context-Aware Visibility**: As seen in the Recording mode, the interface switches to a high-contrast dark theme to ensure legibility under direct sunlight.
- **Thumb Zone Navigation**: Critical actions (Start/Stop) are oversized and placed in the bottom 30% of the screen for safe one-handed interaction.
- **Safety-First Alerts**: The system uses distinct color coding (Yellow/Red) to warn users of specific hazards like tram tracks.

4.2 Core Interaction Flow

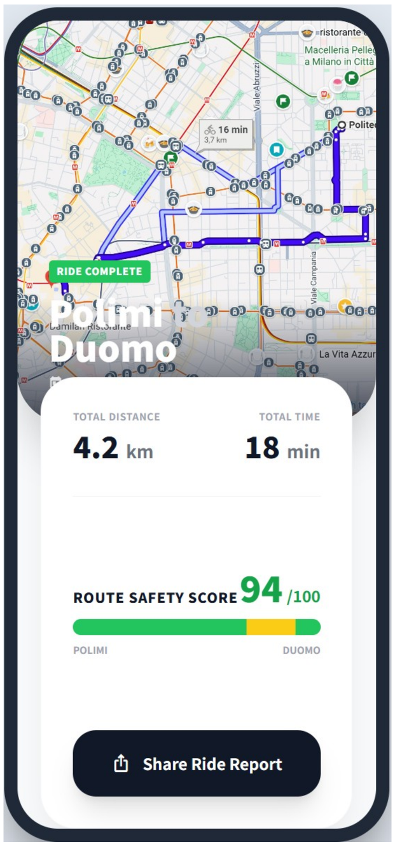
The figures below illustrate the primary user journey for a commute from the university campus to the city center.



(A) Dashboard



(B) Recording



(C) Summary

Figure 12: Core Workflow: (A) Trip Initiation at **Polimi**, (B) Active monitoring with **Tram Alerts**, (C) Trip Completion at **Duomo**.

4.3 Screen Descriptions

1. **Dashboard**: As shown in **Figure 12 (A)**, the central hub automatically detects the user's location at *Piazza Leonardo da Vinci* and suggests the safest route to *Duomo*.

2. **Active Recording**: During the ride (**Figure 12 (B)**), the UI minimizes distractions. It features a large speedometer and dynamic alerts. In this scenario, the app has detected upcoming **tram tracks** on *Corso Venezia*.

3. **Summary View**: Upon arrival (**Figure 12 (C)**), the user receives a "Safety Score" (94/100). The route visualization uses a gradient to indicate safe vs. caution zones.

5 Requirements Traceability

5.1 Traceability Overview

The following section demonstrates the completeness of the BBP system design by mapping each functional requirement defined in the *Requirement Analysis and Specification Document (RASD)* to the specific architectural components and behavioral models (Sequence and State Diagrams) presented in this Design Document.

This traceability ensures that every requirement has a corresponding design element and that no functionality has been omitted during the architectural decomposition.

5.2 Traceability Matrix

The table below provides a detailed mapping between the Requirements (R), the Design Components, and the corresponding Use Case (UC) sequence diagrams.

Req. ID	Requirement Summary	Design Components	Diagrams
R1	Start/Stop the recording of a biking trip.	Mobile: Trip Manager	Fig. 6
R2	Sample geolocation coordinates at a fixed frequency.	Mobile: Trip Manager, GPS Sensor	Fig. 6, 4
R3	Calculate trip statistics (distance, speed).	Mobile: Trip Manager; Server: Trip Controller	Fig. 6
R4	Fetch weather data upon trip completion.	Server: Trip Service, Weather Integration	Fig. 6, 4
R5	Browse personal trip history.	Mobile: UI Layer; Server: Trip Controller	Fig. 11
R6	Manually insert path status information.	Mobile: Path Reporter; Server: Path Controller	Fig. 7
R7	Categorize path status (Optimal, Bad, etc.).	Mobile: Path Reporter	Fig. 7
R8	Flag the presence of obstacles.	Mobile: Path Reporter	Fig. 7
R9	Set information visibility (Private/Shared).	Server: Path Controller, Database	Fig. 7, 5
R10	Input origin and destination for path visualization.	Mobile: Map Engine / UI	Fig. 8
R11	Visualize possible biking paths on the map.	Mobile: Map Engine; External: Map API	Fig. 8
R12	Display computed Safety Scores for paths.	Server: Path Safety Scorer	Fig. 8

5.3 Coverage Analysis

As indicated in the matrix above:

- **Trip Management (R1–R5)** is primarily handled by the *Trip Manager* component on the client side and the *Trip Service* on the server side, with dynamic behavior captured in the Trip State Diagram and Sequence Diagrams UC1 and UC6.
- **Crowdsourced Reporting (R6–R9)** relies on the *Path Reporter* and *Path Controller* modules, with privacy logic enforced by the Visibility State Diagram and Sequence Diagram UC2.

- **Route Planning (R10–R12)** is implemented via the integration of the *Map Engine* and the backend *Path Safety Scorer*, as detailed in Sequence Diagram UC3.

6 Implementation, Integration & Testing

6.1 Overview

This section outlines the strategy for the development and verification of the **Best Bike Paths (BBP)** system. The implementation follows an incremental approach, while the testing strategy adopts the "Testing Pyramid" model to ensure robustness at the unit, integration, and system levels.

6.2 Implementation Plan

The development process is organized into four major milestones (Sprints), focusing on delivering a minimum viable product (MVP) before expanding to complex analytical features.

6.2.1 Feature Identification and Development Phases

- **Sprint 1: Foundation (Backend & Auth)**
  - Setup PostgreSQL database with PostGIS extensions.
  - Implement User Controller and JWT-based authentication.
  - Develop the API Gateway and base routing logic.
- **Sprint 2: Mobile Recording Infrastructure**
  - Implement the high-frequency GPS sampling engine on the Mobile Client.
  - Develop local SQLite persistence for the "Offline-First" strategy.
  - Create basic Map dashboard using external tile providers.
- **Sprint 3: Integration & External Services**
  - Implement the asynchronous Trip Service to handle trajectory uploads.
  - Integrate with the External Weather API for automated enrichment.
  - Enable the Path Reporter module for manual status submissions.
- **Sprint 4: Analytics & UI Polishing**
  - Develop the Path Safety Scorer algorithm on the backend.
  - Implement the Map Visualizer overlays for scored paths.
  - Final UX/UI refinement and end-to-end performance optimization.

6.3 Component Integration and Testing

Integration testing focuses on the interaction between the Mobile Client and the Cloud Backend, as well as the connectivity with external map and weather providers.

- **API Integration**: Automated tests (e.g., using Postman/Newman) to verify that the Backend correctly parses complex GeoJSON trajectories sent by the client.
- **Service Resiliency**: Testing the system's behavior when external APIs (Weather/Map) are slow or unreachable, ensuring the core trip-saving logic remains functional.
- **Data Consistency**: Verifying that private/public visibility toggles correctly filter data in the database during cross-module queries.

6.4 System Testing

System testing evaluates the BBP application in a real-world environment to ensure it meets all functional and non-functional requirements defined in the RASD.

6.4.1 Functional Correctness

Field tests will be conducted in urban environments (e.g., central Milan) to verify that recorded distances match reference odometers and that obstacles are reported at the correct geographic coordinates.

6.4.2 Non-Functional Performance Testing

- **GPS Accuracy & Battery Consumption**: Monitoring the impact of continuous GPS polling (1Hz) on device battery life over a 2-hour cycling session.
- **Concurrency**: Simulating multiple users uploading trips simultaneously to test the backend's response time and database locking behavior.
- **Recovery**: Simulating app crashes during a recording session to verify that the Local Cache Manager correctly recovers the partial trip data upon restart.

6.5 Additional Specifications on Testing

Due to the safety-critical nature of "Best Bike Paths," particular emphasis is placed on **Geospatial Accuracy Testing**. This involves validating that the Map Engine correctly aligns the recorded GPS points with the underlying road network topology (Map Matching) to prevent misleading safety score visualizations.

7 Effort Spent

7.1 Individual Contribution

The following table details the estimated amount of time spent on the development of the Design Document. The architectural modeling and traceability analysis represent the core of the effort.

Task	**Time Spent (h)**
Introduction and Document Structure	4
High-level Architectural Design	8
Component and Deployment Modeling	10
Sequence and State Diagrams	12
User Interface Design and UX Flow	6
Requirements Traceability Analysis	4
Implementation and Testing Strategy	6
Final Review and Formatting	4
Total	**54**

References