Best Bike Paths - Hu Peng, Zhang Zihao

POLITECNICO
MILANO 1863

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Hu Peng, Zhang Zihao
Version:	1.0
Date:	5-January-2026
Download page:	https://github.com/euorrl/HuZhang
Copyright:	Copyright © 2025, Hu Peng, Zhang Zihao – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

This document presents the Design Document (DD) for the Best Bike Paths (BBP) system. The purpose of this document is to describe the architectural and detailed design decisions adopted to implement the system, starting from the requirements defined in the Requirement Analysis and Specification Document (RASD).

While the RASD focuses on what the system is required to do, this document focuses on how the system is structured and how its components interact in order to fulfill the specified requirements. The design aims to provide a clear, modular, and maintainable structure that supports the implementation and future evolution of the BBP system.

The design choices presented in this document are driven by the functional and non-functional requirements identified in the RASD, with particular attention to simplicity, modularity, and ease of implementation.

1.1 Purpose

The purpose of this Design Document is to provide a detailed description of the software architecture and the main design decisions adopted for the implementation of the Best Bike Paths (BBP) system.

This document serves as a reference for developers by defining the system components, their responsibilities, and the interactions among them. The DD supports the transition from requirements to implementation and ensures that the system architecture is consistent with the requirements specified in the RASD.

1.2 Scope

This Design Document addresses the design of the BBP system as a whole, including the high-level architecture, the decomposition of the system into subsystems, and the description of the main software components.

The document focuses on the logical and architectural aspects of the system and does not provide low-level implementation details. User interface layout details and specific technology-dependent configurations are considered out of scope unless necessary to clarify architectural decisions.

1.3 Definitions, Acronyms, Abbreviations

The definitions, acronyms, and abbreviations used in this document are the same as those introduced in the Requirement Analysis and Specification Document (RASD) for the Best Bike Paths (BBP) system.

1.4 Revision History

Version	Date	Description
1.0	5-January-2025	Initial version of the Design Document

1.5 Reference Documents

The following documents are referenced in the preparation of this Design Document:

- Requirement Analysis and Specification Document (RASD) for the Best Bike Paths (BBP) system.
- Software Engineering II course material, Politecnico di Milano.
- IEEE Standard for Software Design Descriptions.

1.6 Document Structure

This Design Document is organized as follows:

- **Introduction**: provides an overview of the document, its purpose, scope, definitions, and reference information.
- **Architectural Design**: describes the high-level architecture of the BBP system, including its main components, their interactions, deployment configuration, runtime behavior, component interfaces, and the selected architectural styles and patterns.
- **User Interface Design**: presents the design of the user interfaces, focusing on the main interaction flows and the organization of the user-facing components.
- **Requirements Traceability**: illustrates the mapping between the requirements defined in the RASD and the architectural and design elements described in this document.
- **Implementation, Integration and Testing**: outlines the planned approach for implementing the system, integrating its components, and performing testing activities at different levels.
- **Effort Spent**: reports the effort spent by each group member during the preparation of this document.
- **References**: lists the documents and standards referenced in this Design Document.

2 Architectural Design

2.1 Overview

The Best Bike Paths (BBP) system is designed as a distributed software system based on a client–server architecture. The system separates user-facing components from backend services in order to improve modularity, maintainability, and scalability.

At a high level, the BBP system consists of three main parts: a client application, a backend application, and a persistent data storage layer. The client application is responsible for user interaction and visualization, while the backend application manages business logic, data processing, and communication with external services. The data storage layer is used to persist user data, trip records, and information about bike paths.

The interaction between components follows a request–response model, where clients send requests to the backend through well-defined interfaces. The backend processes the requests, interacts with the data storage and external services if needed, and returns the results to the client.

2.2 Component View

The BBP system is decomposed into a set of logical components, each responsible for a specific subset of functionalities.

The main components of the system are:

- **Client Application**: provides the user interface and handles user interactions, including trip recording requests, route visualization, and consultation of community information.
- **Backend Application**: implements the core business logic of the system, manages user requests, enforces access control, and coordinates data processing.
- **Data Storage**: stores persistent data such as user profiles, recorded trips, and bike path information.
- **External Services**: include third-party services such as mapping and weather services, which are accessed by the backend when required.

Each component interacts with others through well-defined interfaces, allowing components to evolve independently as long as the interfaces are preserved.

2.3 Deployment View

The BBP system is deployed in a distributed environment. The client application runs on user devices, such as mobile devices or personal computers, and communicates with the backend application over a network.

The backend application is deployed on a server environment and is responsible for handling client requests and accessing the data storage. The data storage component is deployed on a database server, which may be co-located with the backend application or deployed as a separate service.

External services are deployed outside the BBP system boundary and are accessed through network-based APIs.

2.4 Runtime view

This section describes the dynamic behavior of the Best Bike Paths (BBP) system at runtime. The runtime view focuses on how the main system components interact during the execution of key use cases, highlighting the flow of control and data among the client application, the backend application, the data storage, and external services.

The runtime interactions are illustrated using sequence diagrams, which provide a clear representation of the temporal order of messages exchanged between components during system execution.

Figure 1 shows the runtime interactions related to the *Record Trip* use case (UC1). The diagram illustrates how a registered user initiates a trip recording through the client application, how the backend creates and manages the trip, and how location data is progressively stored in the data storage component until the trip is finalized.

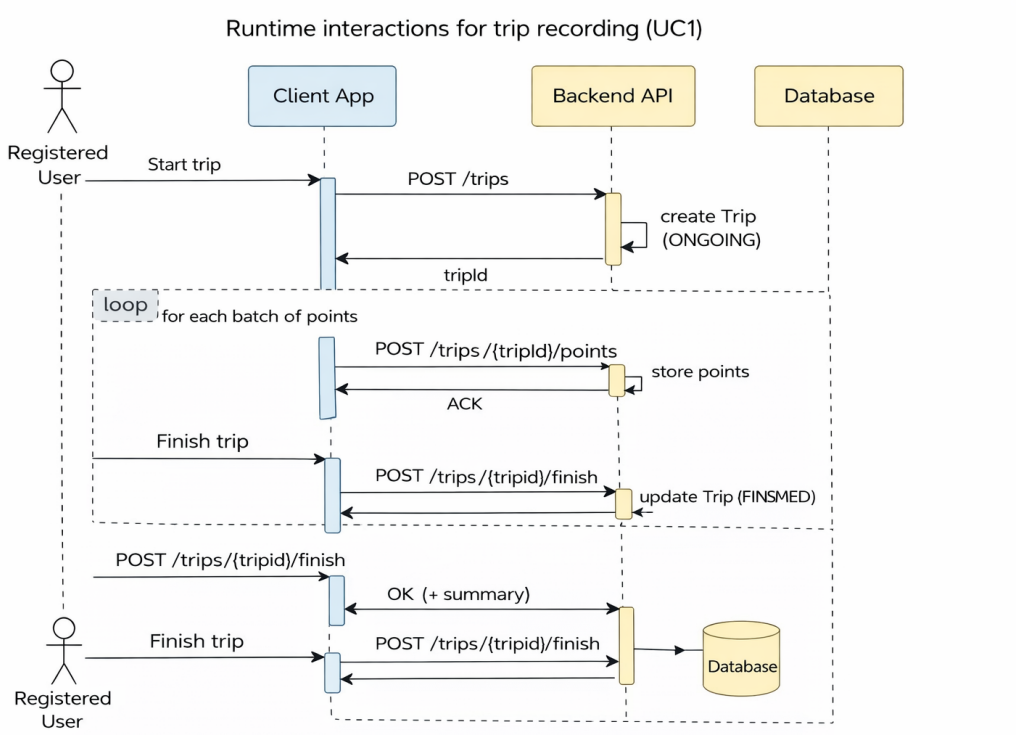

Figure 1: Runtime interactions for trip recording (UC1)

Figure 2 shows the runtime interactions related to the *Route Visualization* use case (UC5). The diagram highlights how a user requests the visualization of routes between an origin and a destination, how the client forwards the request to the backend, and how the backend optionally interacts with an external mapping service to retrieve candidate routes before returning the results to the client for visualization.

In addition to the core use cases described above, Figures 3, 4, and 5 illustrate the runtime interactions for other supported use cases. These use cases follow simpler interaction patterns based on standard request–response communication between the client application, the backend, and the data storage.

Figure 3 shows the runtime interactions related to the *View Trip* use case (UC2), in which a registered user retrieves and visualizes previously recorded trips.

Figure 4 shows the runtime interactions related to the *Insert Bike Path Information* use case (UC3), illustrating how a registered user submits path-related information to the backend and how the information is stored in the data storage component.

Figure 5 shows the runtime interactions related to the *Publication Control* use case (UC4), illustrating how a registered user can change the publishable status of previously inserted bike path information.

Together, these runtime diagrams demonstrate that the BBP architecture supports both complex and simple interaction patterns, and that the system design is consistent with the functional requirements defined in the RASD.

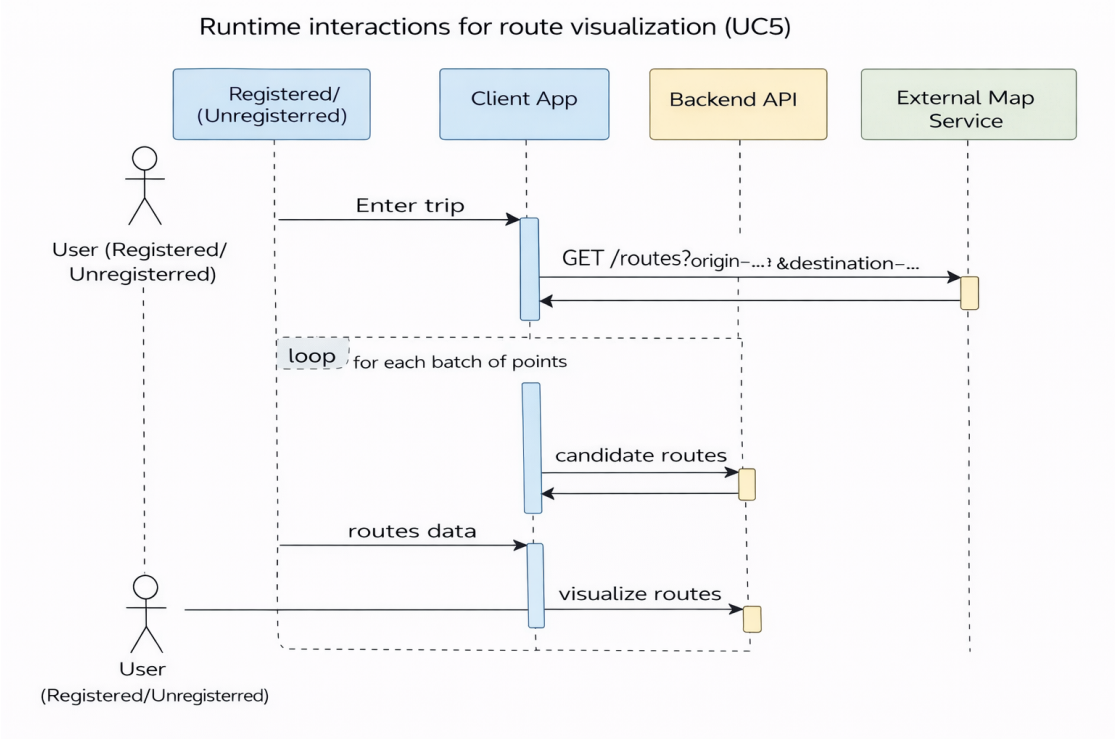

Figure 2: Runtime interactions for route visualization (UC5)

2.5 Component interfaces

The backend application exposes a set of RESTful interfaces to the client application. The interfaces are grouped by domain area and are designed to support the core use cases described in the RASD (UC1–UC5). All interfaces exchange data using standard JSON representations.

2.6 Selected Architectural Styles and Patterns

The BBP system adopts a client–server architectural style to clearly separate user interaction concerns from backend processing.

Within the backend application, a layered architectural pattern is adopted. The backend is conceptually divided into presentation, business, and data access layers, each with distinct responsibilities.

This architectural choice improves separation of concerns and supports maintainability and extensibility.

2.7 Other Design Decisions

Several additional design decisions have been adopted to support the overall quality of the BBP system.

The system prioritizes simplicity over optimization, focusing on clarity and ease of implementation. Design decisions avoid unnecessary complexity and advanced mechanisms unless they are required to satisfy specific requirements.

This approach supports a robust and understandable design that can be implemented and maintained effectively.

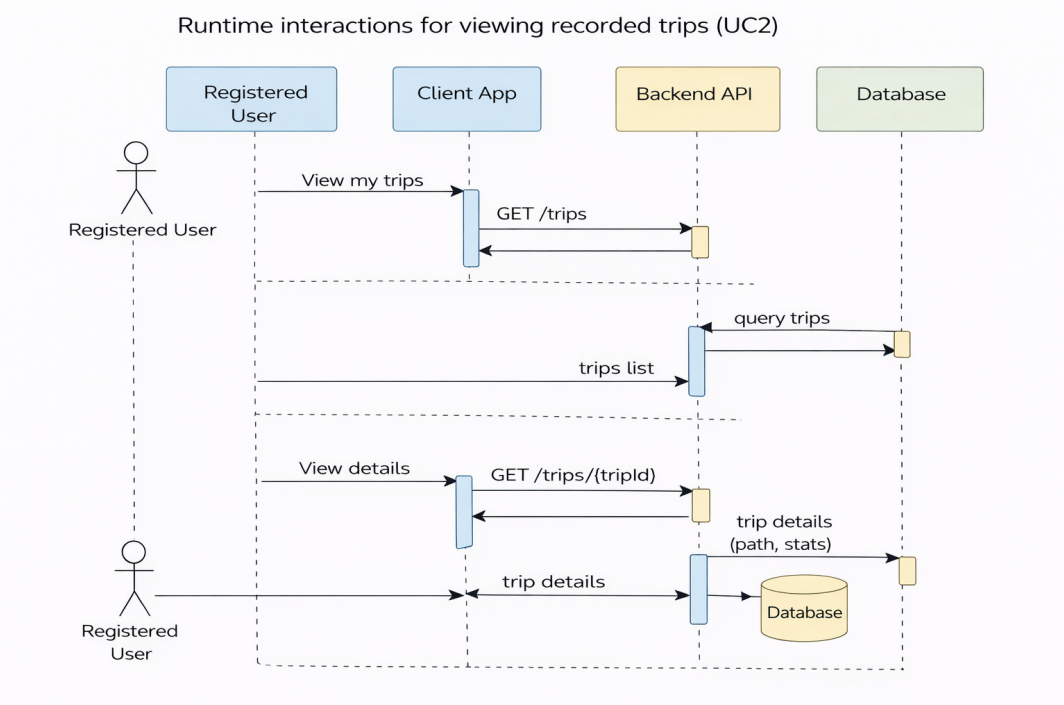

Figure 3: Runtime interactions for viewing recorded trips (UC2)

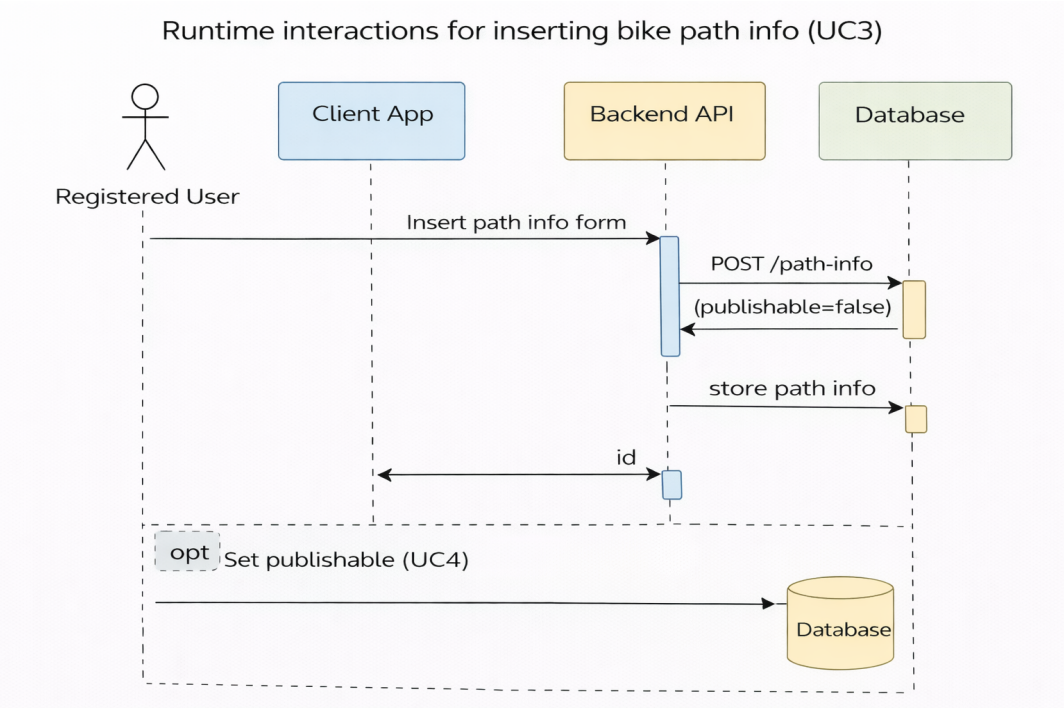

Figure 4: Runtime interactions for inserting bike path information (UC3)

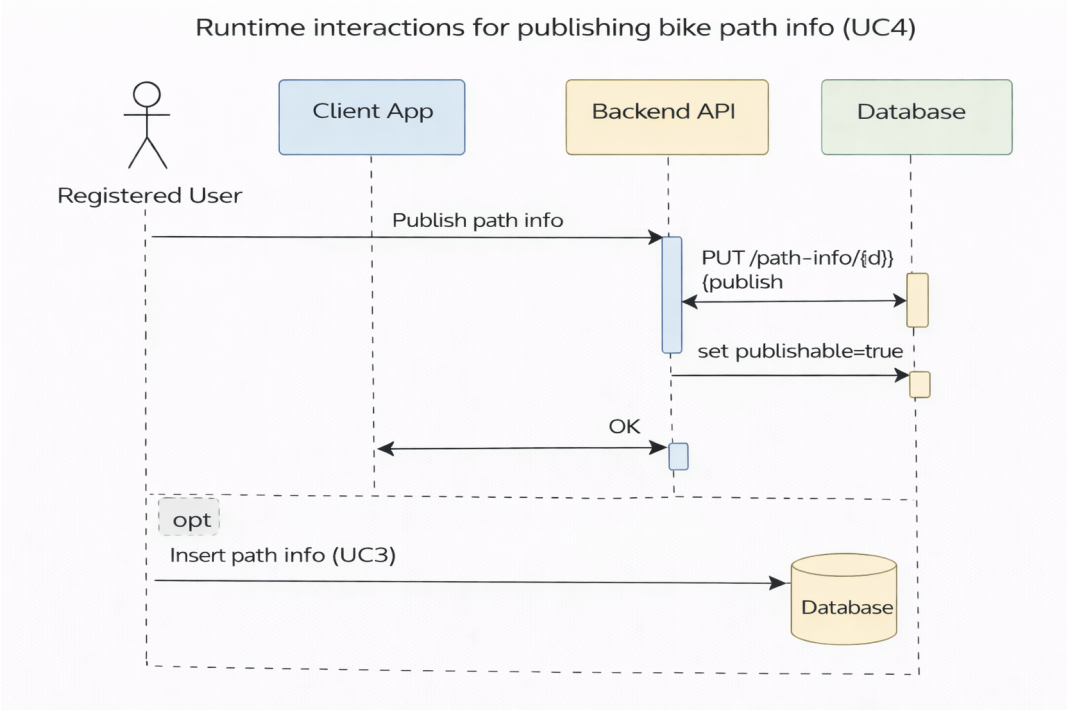

Figure 5: Runtime interactions for publishing bike path information (UC4)

Domain	**Method**	**Endpoint**	**Description**
Authentication	POST	/auth/register	Registers a new user account.
Authentication	POST	/auth/login	Authenticates a user and starts a session (or returns an access token).
Trips	POST	/trips	Creates a new trip and starts recording (returns *tripId*).
Trips	POST	/trips/{tripId}/points	Uploads one or more location points for an ongoing trip.
Trips	POST	/trips/{tripId}/finish	Ends the trip recording and stores final trip information.
Trips	GET	/trips	Retrieves the list of trips recorded by the authenticated user.
Trips	GET	/trips/{tripId}	Retrieves details of a specific trip (including its path).
Path Information	POST	/path-info	Inserts a new bike path information entry (default: not publishable).
Path Information	PUT	/path-info/{id}/publish	Sets the publishable flag of a path information entry.
Path Information	GET	/path-info	Retrieves publishable community information (optionally filtered).
Path Information	GET	/path-info/{id}	Retrieves details of a specific path information entry.
Routes	GET	/routes	Retrieves candidate routes between origin and destination for visualization.

Table 1: Main REST interfaces exposed by the BBP backend

3 User Interface Design

3.1 Overview

The user interface of the Best Bike Paths system is designed to support the main use cases defined in the RASD. The user interface is part of the client application and is responsible for collecting user inputs and presenting system outputs.

The detailed design of the user interfaces, including page descriptions and screenshots, is provided in the RASD. This section does not redefine the user interfaces, but summarizes their role within the system and their alignment with the architectural design described in Section 2.

3.2 User Interfaces Defined in the RASD

The BBP system provides the following user interfaces, which are fully described in the RASD:

- **Home Page**, which represents the entry point of the system and provides access to the main functionalities.
- **Explore Routes Page**, which allows users to specify an origin and a destination and visualize possible bike paths (UC5).
- **Community Page**, which allows users to browse publishable bike path information provided by registered users (UC4).
- **Login Page**, which enables user authentication and access to functionalities reserved to registered users.
- **My Trips Page**, which allows registered users to view their recorded bike trips and related information (UC1, UC2).
- **Report Path Page**, which allows registered users to manually insert bike path information and manage its publishable status (UC3).

No additional user interfaces beyond those defined in the RASD are introduced in this Design Document.

3.3 Interaction Flow and Architectural Consistency

User interactions through the interfaces defined above follow a request–response communication model consistent with the client–server architecture described in Section 2. The client application handles user interaction and visualization concerns, while the backend application is responsible for processing requests, enforcing access control, and managing persistent data.

This separation of responsibilities ensures consistency between the user interface design and the architectural choices adopted for the BBP system, and supports the functional requirements defined in the RASD.

4 Requirements Traceability

4.1 Traceability Approach

This section describes how the goals and functional requirements defined in the Requirement Analysis and Specification Document (RASD) are reflected in the design choices presented in this Design Document.

The traceability analysis is organized around the three system goals (G1–G3) and covers all related functional requirements (R1–R21). Design assumptions and constraints are used to describe the context in which the system operates and are not treated as implementable requirements.

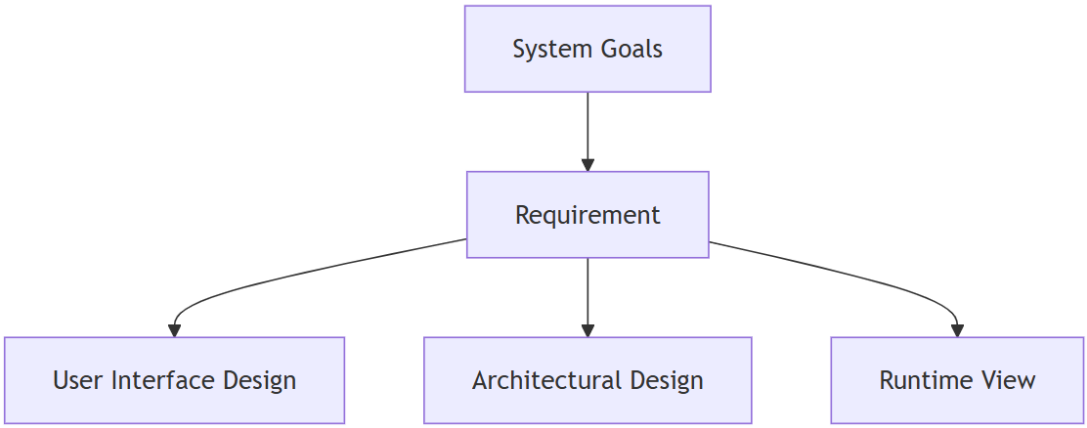

Figure 6: Overview of requirements traceability from RASD to design elements

4.2 Traceability from System Goals to Design Elements

- G1 – Personal bike trip recording and management
- G2 – Manual insertion and publication control of bike path information
- G3 – Exploration and visualization of bike paths

Goal ID	G1
Goal Description	Cyclists are supported in recording their personal bike trips, allowing them to keep track of their cycling activities and access basic statistics related to their rides.
Related Requirements	R6–R12 (Trip recording, data collection, storage, visualization, and browsing of trip history)
Design Elements	**User Interface Design**: My Trips Page, Home Page **Architectural Design**: Separation of responsibilities among Client Application, Backend Application, and Data Storage (Section 2.2) **Runtime View**: Interaction flows for UC1 and UC2 described in Section 2.4

Table 2: Mapping between Goal G1 and Design Elements

Goal ID	G2
Goal Description	Users are enabled to manually insert information about bike paths, including path conditions and relevant observations, and to decide whether such information can be made publishable to the community.
Related Requirements	R13–R18 (Manual insertion of bike path information, qualitative evaluations, and the distinction between private and published data)
Design Elements	**User Interface Design**: Report Path Page, Community Page **Architectural Design**: Management of bike path information and separation between private user data and community data (Section 2.2) **Runtime View**: Interaction flows for UC3 and UC4 described in Section 2.4

Table 3: Mapping between Goal G2 and Design Elements

Goal ID	G3
Goal Description	The system supports the visualization of possible bike paths between a user-specified origin and destination, helping cyclists explore available routes and select suitable paths.
Related Requirements	R19–R21 (Specifying origin and destination points and visualizing available bike paths)
Design Elements	**User Interface Design**: Explore Routes Page **Architectural Design**: Optional interaction between the Backend Application and external mapping services (Section 2.2) **Runtime View**: Interaction flow for UC5 described in Section 2.4

Table 4: Mapping between Goal G3 and Design Elements

4.3 Traceability of Supporting Functional Requirements (R1–R5)

Functional requirements R1–R5 describe user registration, authentication, and access control. While these requirements do not directly define the core system goals, they constrain how core functionalities can be accessed.

Design elements in the DD

- User Interface Design: *Login Page* as the authentication entry point
- Access control: *My Trips Page* and *Report Path Page* are accessible only to registered users
- Architectural Design: the Backend Application distinguishes between registered and unregistered users when processing requests (Section 2.2)
- Runtime View: consistency with the preconditions of UC1 and UC3 described in Section 2.4

4.4 Design Assumptions and Constraints (D1–D7)

The design assumptions and constraints defined in the RASD describe the context in which the system operates, including device capabilities, network conditions, external service dependencies, and the subjective nature of user-provided information.

Although these elements do not correspond to specific functionalities, they influence several design decisions. For instance:

- trip recording assumes the availability of mobile devices with basic positioning capabilities;
- route exploration relies on external mapping services located outside the system boundary;
- the system presents possible routes without guaranteeing optimality, in line with the subjective nature of community-provided data.

5 Implementation, Integration And Test Plan

5.1 Overview

This chapter describes the implementation order, integration strategy, and testing approach adopted for the Best Bike Paths (BBP) system.

The system is developed incrementally. Core components are implemented first, and functionalities are added step by step. After each increment, the system is validated through unit tests and integration tests aligned with the main use cases (UC1–UC5). External services, such as mapping or weather providers, are considered system dependencies and are tested through controlled substitutes whenever required to ensure repeatability.

A key element of the testing strategy is the use of a GPS data simulator. Since live GPS data is inherently unpredictable and difficult to reproduce, system testing is based on a deterministic program that generates location points along predefined routes and provides them to BBP in place of a real GPS API.

5.2 Implementation Plan

The implementation follows a feature-oriented order, starting from the core data model and backend logic, and progressively introducing user-facing functionalities. This approach ensures that each development step results in a coherent and testable system state.

1. **Core domain model and persistence**
   - Definition of the main entities, including *User*, *Trip*, *TripPoint*, and *BikePathInfo*, as well as data structures for route queries.
   - Design and implementation of the database schema and persistence layer supporting basic CRUD operations.
2. **Authentication and access control**
   - Implementation of login and logout mechanisms and backend support for user identification.
   - Enforcement of access control rules to restrict trip recording and manual bike path insertion to authenticated users.
3. **Trip recording functionality (UC1)**
   - Implementation of backend endpoints to create a trip, receive batches of location points, and finalize a trip.
   - Computation and storage of basic trip metrics, such as duration, distance, and average speed.
4. **Trip visualization and browsing (UC2)**
   - Implementation of endpoints to retrieve the list of recorded trips and detailed information for a selected trip.
   - Provision of data formats suitable for map visualization and statistics presentation.
5. **Manual insertion of bike path information and publication control (UC3–UC4)**
   - Implementation of interfaces for inserting qualitative information about bike paths.
   - Management of publishable status, ensuring a clear separation between private user data and publicly available community information.
   - Implementation of consultation endpoints accessible to both registered and unregistered users.

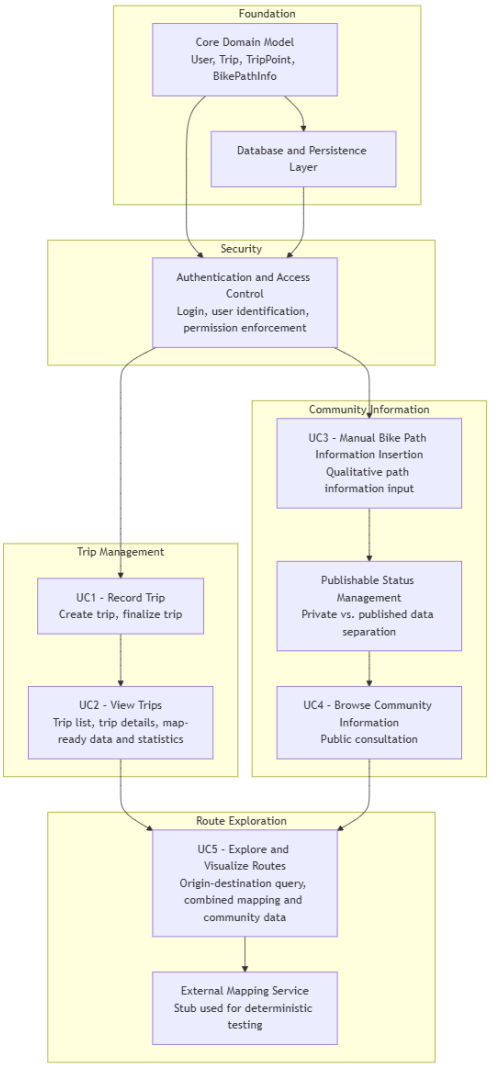

Figure 7: Layered implementation order of the BBP system.

6. **Route exploration and visualization (UC5)**
    - Implementation of a route query interface accepting origin and destination points.
    - Integration with an external mapping service, with the possibility of replacing it with a stub during testing.
    - Combination of routing data with community-provided information before returning results to the client.

5.3 Integration Plan and Test Strategy

5.3.1 Integration approach

Integration is performed incrementally and follows a feature-driven strategy. The backend API and the data storage layer are integrated first, after which the client application is connected to the corresponding API endpoints.

Each integration step is validated through:

- **Unit tests**, targeting core domain logic such as distance computation, statistics calculation, access control checks, and publication rules;

- **API-level tests**, verifying request and response formats, status codes, and error handling;
- **End-to-end tests**, aligned with the main system use cases (UC1–UC5).

External dependencies are integrated last and replaced by controlled substitutes when variability or non-determinism would compromise test reproducibility.

5.3.2 Test data and GPS simulation

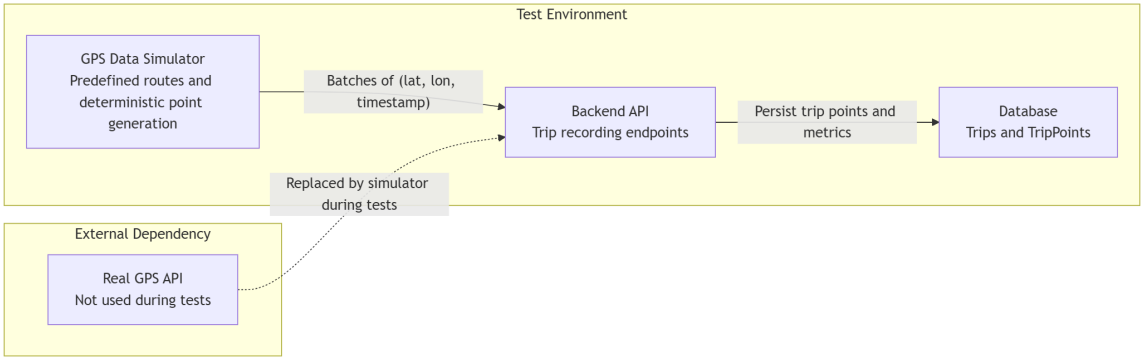

Figure 8: Test architecture based on a deterministic GPS data simulator.

To ensure repeatable testing of the trip recording functionality, BBP relies on a GPS data simulator. The simulator is implemented as a separate program or module that:

- contains a set of predefined routes representing different cycling scenarios (e.g., short urban trips, longer routes, stop-and-go segments);
- generates sequences of `(latitude, longitude, timestamp)` points by sampling the selected route at a configurable frequency;
- optionally introduces controlled noise or gaps to emulate realistic conditions, while keeping the overall sequence deterministic.

As shown in Figure 8, during tests, the simulator replaces a real GPS API by sending batches of generated points to the same backend endpoints used during normal trip recording. This allows the expected system behavior to be verified against known and stable inputs.

5.3.3 Incremental validation per use case

UC1 – Record Trip

- **Integration scope:** Client application, backend API, and database.
- **Test driver:** GPS simulator feeding batches of location points.
- **Validation:** correct trip state transitions, persistence of points, and correctness of computed trip metrics.

UC2 – View Trip

- **Integration scope:** trip retrieval endpoints and data storage.
- **Validation:** correct filtering by user, ordering of trip history, and integrity of returned trip geometries and statistics.

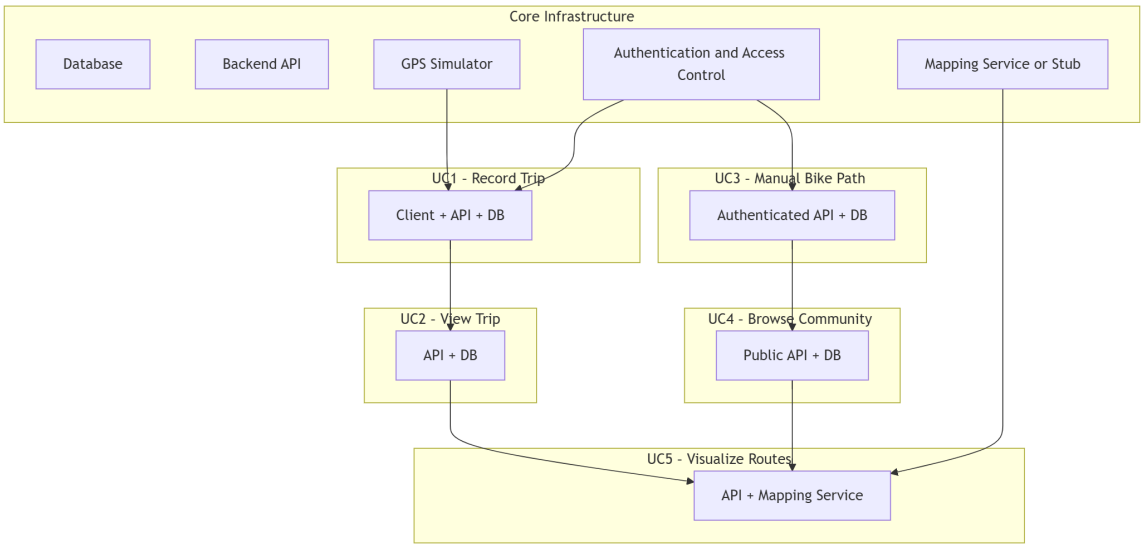

Figure 9: Incremental validation of system functionalities per use case.

UC3 – Manualinsertion of bike path information

- **Integration scope:** authenticated insertion of bike path information.
- **Validation:** enforcement of access restrictions, correct storage of qualitative data, and preservation of private information for non-published entries.

UC4 – Browse Community Information

- **Integration scope:** publishable status management and public consultation interfaces.
- **Validation:** correct visibility rules for published versus private data and access for unregistered users to public information.

UC5 – Visualize Routes between Origin and Destination

- **Integration scope:** route query interface and mapping service.
- **Validation:** input validation for origin and destination, correctness of returned route structures, and inclusion of community-related data.
- **Testing note:** for deterministic tests, the mapping service may be replaced by a stub returning predefined routes.

5.4 System Testing

System testing validates BBP as a whole against its functional requirements and main use cases.

Functional testing executes UC1–UC5 end-to-end, covering both registered and unregistered user scenarios. The GPS simulator enables repeatable verification of recorded trips and computed statistics.

Performance testing focuses on basic responsiveness under typical workloads, such as batch insertion of location points, trip history retrieval, and route queries. The objective is to ensure acceptable performance for realistic usage conditions.

Usability testing assesses the clarity of user interactions, the separation between public and restricted functionalities, and the readability of map-based visualizations and trip statistics.

Reliability testing evaluates system behavior under irregular conditions, such as incomplete point batches or simulated connectivity gaps, ensuring robust error handling and data consistency.

6 Effort Spent

The following table displays the number of hours each group member spent on the various sections of the document. Please note that this division is only approximate, and each section still required the collaboration of all members.

Table 5: Distribution of effort among group members

Member	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5
Hu Peng	2	3	3	3	4
Zhang Zihao	3	6	2	3	2

7 Use Of AI Tool

During the preparation of this Design Document, generative artificial intelligence tools were used to support limited writing and organization tasks. Their usage was mainly focused on improving the clarity of technical descriptions and assisting in the organization of the document structure. All architectural decisions, component responsibilities, interface definitions, runtime interactions, and testing strategies described in this document were independently analyzed and determined by the authors, who take full responsibility for their correctness.

During the DD writing phase, the generative artificial intelligence tool used was ChatGPT (OpenAI), employed as a conversational support tool.

Usage Description

The artificial intelligence tool was primarily used in the following contexts:

- improving the wording, clarity, and consistency of existing technical descriptions;
- assisting in organizing written explanations of established architectural designs and runtime behaviors;
- supporting the drafting of the implementation, integration, and test plan based on implementation ideas and testing strategies defined by the authors;
- checking terminology and naming consistency across different sections of the document.

Review and Final Validation

All text generated or modified with the support of artificial intelligence tools was manually reviewed and, when necessary, adjusted before being included in this document. The described system architecture, component responsibilities, interface semantics, and testing strategies were cross-checked for consistency across sections.

All diagrams were manually created by the authors based on the system design. In some cases, the artificial intelligence tool was used to assist in clarifying the meaning of structures and interactions represented in the diagrams and to improve the associated textual explanations, but it did not take part in the actual construction of the diagrams.

The outputs provided by artificial intelligence tools were used solely as supporting material during the writing process. The final content of this document reflects the authors' decisions and is intended to comply with the course requirements and the objectives of a design document.

References

