POLITECNICO MILANO 1863

COMPUTER SCIENCE AND ENGINEERING
SOFTWARE ENGINEERING II
2025 - 2026

DD
Design Document

Authors:

- Dai Run Jie Simone - 10766478
- Yang Jiaxin - 10719178
- Zheng Jiayi - 10811585

Contents

1| Introduction

1.1 Purpose

The **Best Bike Paths (BBP)** system aims to support cyclists in recording their riding activities and in discovering, evaluating, and sharing information about bike paths.
The system is designed to improve the overall cycling experience by combining personal trip tracking with community-driven knowledge about path conditions, obstacles, and route quality.

BBP enables registered users to monitor their cycling performance while contributing reliable and up-to-date information about bike paths. At the same time, the system allows any user, including unregistered ones, to explore available routes between a chosen origin and destination and to compare alternative paths based on quality and effectiveness.

1.2 Scope

The BBP system covers the full lifecycle of cycling activities and bike path information management.

From an individual perspective, the system allows registered users to:

- record cycling trips using mobile devices,
- collect location and sensor data during rides,
- compute and visualize trip statistics,
- optionally enrich trip data with meteorological information.

From a collaborative perspective, BBP supports the collection and sharing of bike path information through two complementary approaches:

- **manual mode**, where users explicitly insert information about path segments and their conditions;
- **automated mode**, where the system detects biking activity, reconstructs followed paths, and identifies potential obstacles using sensor data, requiring user validation to handle possible false positives.

The system aggregates and merges publishable bike path reports provided by multiple users by considering their freshness and level of agreement. This process ensures that the information presented to users reflects the most reliable and recent assessment of bike path conditions.

Finally, BBP allows users to search for bike paths between an origin and a destination, visualize them on a map, and compare alternatives based on computed scores that account for both path quality and route effectiveness.

1.3 Definitions, Acronyms, AbbreviationsDefinitions

- **Bike Path**: A path suitable for cycling, either explicitly dedicated to bicycles or characterized by low traffic and compatible speed limits.
- **Trip**: A cycling activity recorded by a registered user, from start to end.
- **Bike Path Segment**: A continuous portion of a bike path used as the basic unit for reporting conditions.
- **Bike Path Report**: Information provided by a user describing the condition of a bike path segment and possible obstacles.
- **Publishable Report**: A validated report that can be shared with the community and used for aggregation.
- **Merging**: The process of combining multiple publishable reports related to the same bike path segment into a consolidated representation.

Acronyms

- **BBP**: Best Bike Paths
- **GPS**: Global Positioning System
- **UML**: Unified Modeling Language

1.4 Revision History

- Version 1.0: 04/01/2026

1.5 Reference Documents

- Requirements Analysis and Specification Document (RASD) – Best Bike Paths
- Software Engineering II – R&DD Project specification
- IEEE Standards for Software Architecture and Design Documentation

1.6 Document Structure

This document is structured to describe the BBP system from an architectural and design perspective:

1. Introduction presents the goals, scope, and terminology of the BBP system.

2. Architectural Design describes the overall architecture, main components, deployment choices, and runtime behavior.

3. User Interface Design outlines the main characteristics of the user interfaces.

4. Requirements Traceability explains how system requirements are mapped to design elements.

5. Implementation, Integration and Test Plan presents the planned development and testing strategy.

6. Effort Spent reports the effort contributed by each group member.
7. References lists the referenced material.

2| Architectural Design

2.1 Overview

This section consists of a high level description of the architecture of the system, of the single components and the interaction between the architectural layers. It also includes the modeling choices that have been taken to fulfill the functional and nonfunctional requirements defined in the RASD document.

Automated data acquisition is supported through **client-side access to device sensors** (e.g., GPS, accelerometer, gyroscope) when available and authorized by the user. Sensor data is collected by the presentation tier and transmitted to backend services for processing, validation, and aggregation.

2.1.1 System View

The Best Bike Paths (BBP) system adopts a **4-tier architectural style**, designed to ensure modularity, scalability, and high availability.

The **Presentation Layer** is client-side and delivers the user interface via a web browser. In addition to handling user interactions and visualization, it is responsible for **acquiring raw data from the user's device**, such as GPS location and motion sensor data, during cycling activities when supported by the device and explicitly authorized by the user.

The **Application Layer** is split into two tiers residing on the server side:

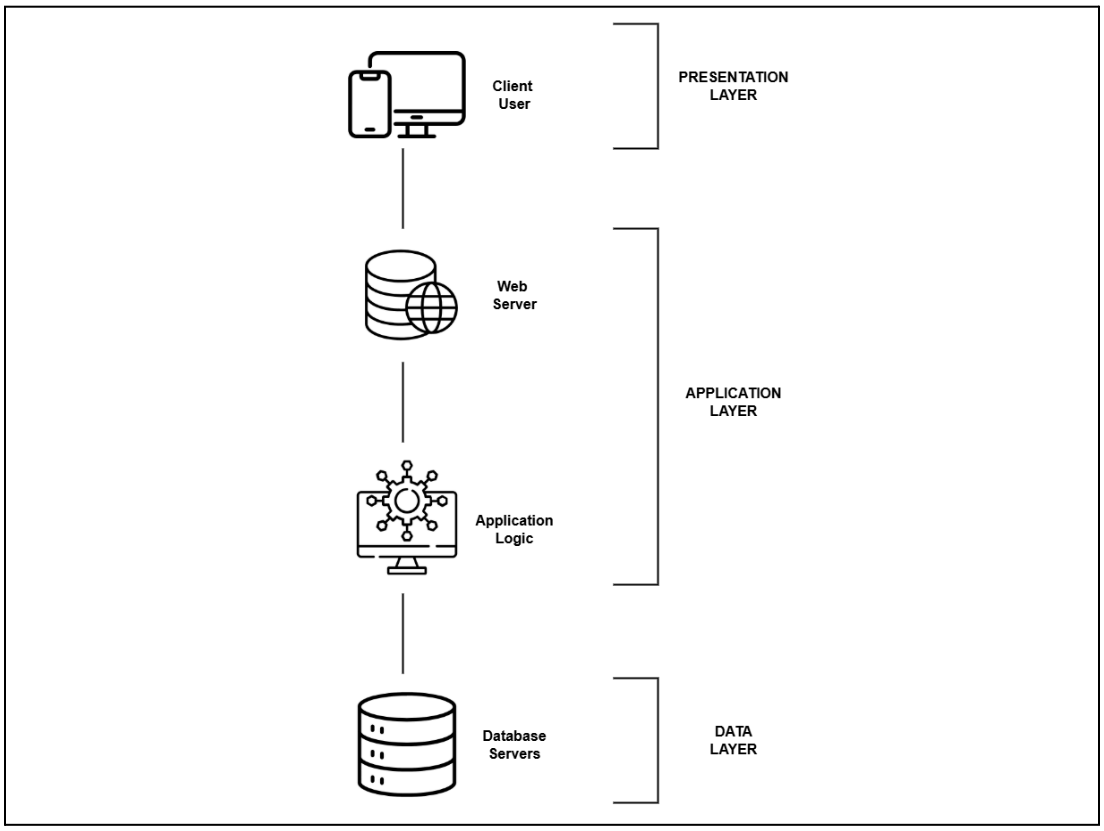
- The **Web Server and API Gateway Tier** manages incoming client requests, routing, security, and load balancing. It acts as a proxy, forwarding requests to appropriate backend services.

- The **Microservices Tier** implements the core system functionality through multiple loosely coupled services, including user management, trip processing, bike path reporting, aggregation, and route ranking. These microservices communicate via REST APIs and encapsulate specific domain responsibilities.

The **Data Layer** consists of databases that store user profiles, trip data, bike path information, and reports.

Sensor data acquisition does not imply local processing or interpretation on the client side. All collected data is forwarded to backend services, where path reconstruction, activity detection, obstacle identification, and validation are performed.

This layered separation promotes independent development, deployment, and scaling of components. The microservices-based application layer aligns with the system's need to manage diverse functionalities and high user concurrency effectively, as outlined in the RASD. Overall, this architecture supports efficient data flow from the user's browser to the backend services and databases, ensuring reliable processing and delivery of information back to the client.



2.1.2 Detailed View

Presentation Tier

Users interact with the Best Bike Paths (BBP) system through a web browser, which constitutes the presentation tier. This tier is responsible for providing the graphical user interface and handling all user interactions, including trip recording, bike path exploration, report confirmation, and visualization of routes and statistics.

The browser communicates with the backend by issuing HTTP requests to the platform's web server and receives responses containing formatted data to be displayed to the user. When supported by the browser and explicitly authorized by the user, the presentation tier also acquires **raw sensor data from the client device**, including GPS location data and motion-related information (e.g., accelerometer and gyroscope readings) during cycling activities.

The presentation tier does not perform any interpretation, validation, or inference on sensor data. All collected data is transmitted to backend services, which are responsible for detecting biking activity, reconstructing followed paths, identifying potential obstacles, and handling false positives through user validation.

Application Tier - Web Server and API Gateway

This tier includes all components that act as the interface between the external network and the core application services. It hosts the web server, the API gateway, and a proxy for external API calls.

The **web server** is responsible for receiving HTTP requests from client browsers and forwarding them to the API gateway. It also serves static content related to the web application when required.
The **API gateway** acts as the central coordination component for request handling. It is responsible for request routing, access control, and request validation. The gateway distinguishes between public and protected endpoints, allowing unauthenticated access to publicly available bike path information for guest users, while enforcing authentication and authorization for registered users accessing personal or write-enabled functionalities. Request distribution across replicated backend services is supported through load balancing mechanisms, improving system responsiveness and availability.

The **proxy component** provides a controlled access point for invoking external APIs required by the system, such as mapping and geolocation services and meteorological data providers. By isolating external API calls within this tier, the system improves security and maintainability while reducing coupling between internal services and third-party providers. This tier, together with network-level security mechanisms such as firewalls, contributes to protecting the system by filtering malicious traffic and preventing unauthorized access before requests reach the core application services.

Application Tier - Microservices Tier

This tier hosts the core services of the BBP system and is responsible for processing user requests that require data retrieval, computation, or enforcement of business rules. This tier is structured according to a **microservices architecture**, in which each service encapsulates a specific system functionality.

Key backend services include user management, trip processing, bike path report management, data aggregation and merging, and route search and ranking. Each microservice exposes a set of RESTful APIs and communicates with other services through HTTP-based REST calls, enabling loose coupling and independent evolution.

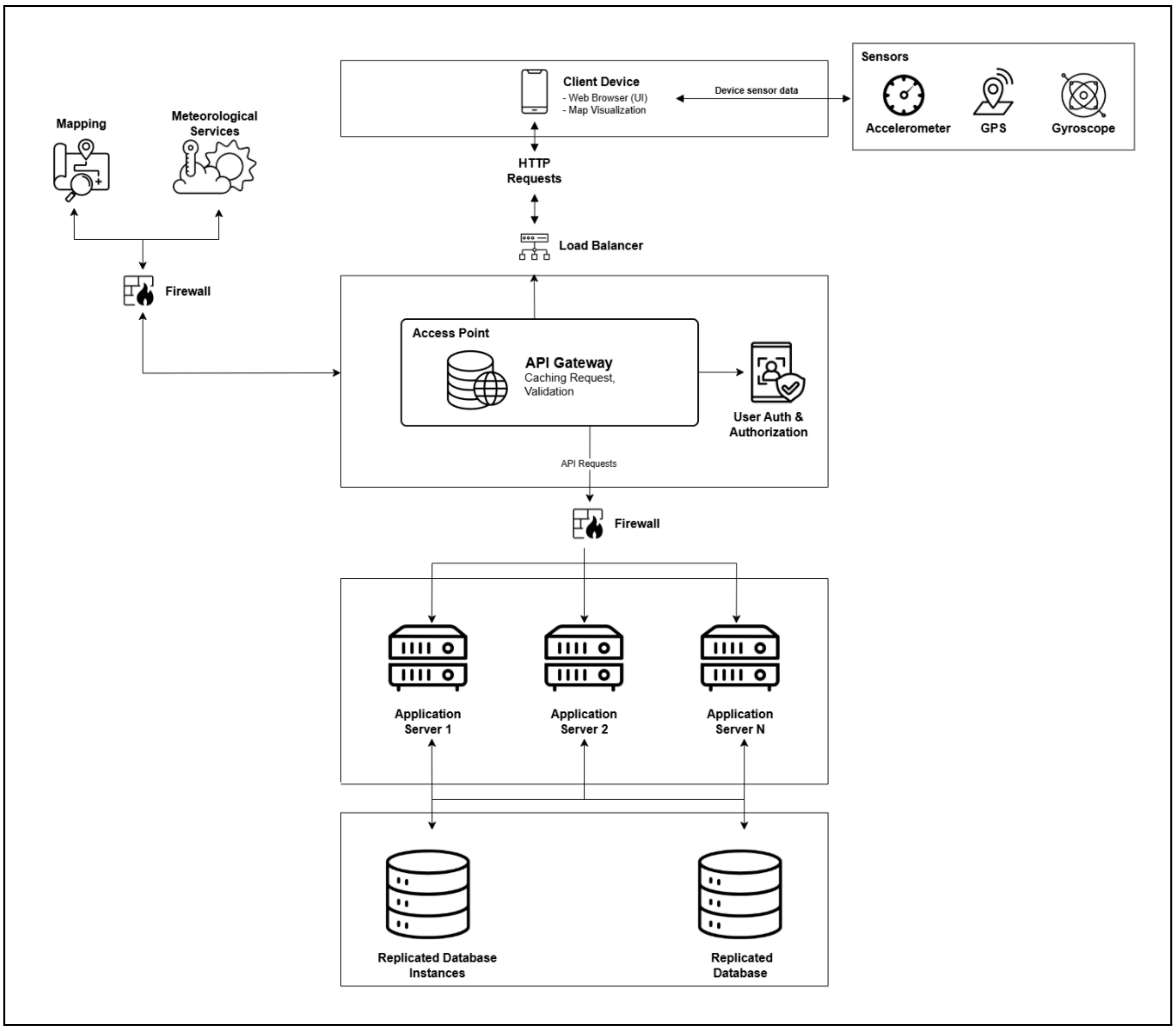
To ensure availability and fault tolerance, backend services are replicated and deployed across multiple machines. This organization allows the system to scale individual services based on workload demands and to isolate failures, preventing issues in one service from propagating to others.

Data Tier

The data tier is responsible for the persistent storage and management of all system data. It consists of a set of replicated databases managed by appropriate DBMS solutions to guarantee consistency, availability, and durability.

Data is organized in a way that supports the responsibilities of individual microservices, allowing services to access and manage the information relevant to their functionalities.

Stored information includes user profiles, recorded trips, bike path segments, bike path reports, and aggregated path conditions. This organization reinforces the decoupling introduced at the application layer and improves system reliability and performance by reducing unnecessary data dependencies across services.



Some Clarifications

- In the diagrams, the term "Access Point" represents the entry point to the backend infrastructure.

- Authentication and authorization mechanisms are implemented within the Web Server and API Gateway tier and supported by the User Management Service. They answer questions like "Who is the user?" and "What is this user allowed to do?", this means checking whether the user is registered or not, and if the action is permitted.

2.2 Component View

The system consists of various components, including some for the front end, some for the back end, and each component performs a specific service. Additionally, it will require interaction with some external components.

The BBP system is composed of frontend components, backend infrastructure components, domain-specific microservices, data storage components, and external services. Each component is designed with a clear responsibility and interacts with others through well-defined interfaces, promoting modularity, maintainability, and scalability.

Front-end Components

➔ **Web Client**

The Web Client is the frontend component executed in the user's web browser. It implements the graphical user interface of the BBP system and supports all user interactions, including:

- handling user interactions, such as starting and stopping trip recording, searching bike paths, and confirming reports;
- displaying maps, routes, trip statistics, and bike path information;
- managing user sessions and interface state;
- acquiring raw sensor data from the device (GPS, accelerometer, gyroscope) during automated mode, when supported and authorized;
- acquiring raw device data, such as geolocation information during cycling activities, when supported by the browser.

The Web Client does not implement business logic or data persistence. All processing and data management are delegated to backend components. Communication with the backend occurs through secure HTTP(S) requests, using RESTful APIs exposed by the backend services.

Back-end Components

➔ **User Management Service**

This service manages registered user profiles and authentication-related data. It supports operations such as user registration, profile retrieval, and authorization checks required by other services when accessing protected resources.

➔ **Trip Management Service**

The Trip Management Service handles the lifecycle of bike trips recorded by registered users. Its responsibilities include:

- storing trip metadata and GPS traces,
- computing trip statistics (distance, duration, average speed),
- associating meteorological information with trips,
- providing access to users' trip history and detailed trip views.

➔ **Bike Path Report Service**

This service manages bike path reports created manually or automatically by users. It supports the creation, validation, confirmation, and publication of reports, enforcing the report lifecycle described in the RASD and formalized in the Alloy model. Only reports marked as publishable are exposed to other services for aggregation.

➔ **Aggregation and Merging Service**

The Aggregation Service is responsible for merging publishable bike path reports provided by multiple users. It evaluates report freshness and user agreement to produce consolidated bike path segment conditions, ensuring consistency with the most recent and reliable information, as required by goals G9 and G10.

➔ **Route Search and Ranking Service**

This service supports origin–destination queries and computes alternative bike paths between specified locations. It retrieves aggregated path data, computes path scores based on quality and effectiveness, and ranks available routes before returning them to the client for visualization.

➔ **DBMS**

The DBMS is responsible for managing persistent data storage and providing controlled access to the database. Backend services interact with the DBMS to store and retrieve system data.

External Components

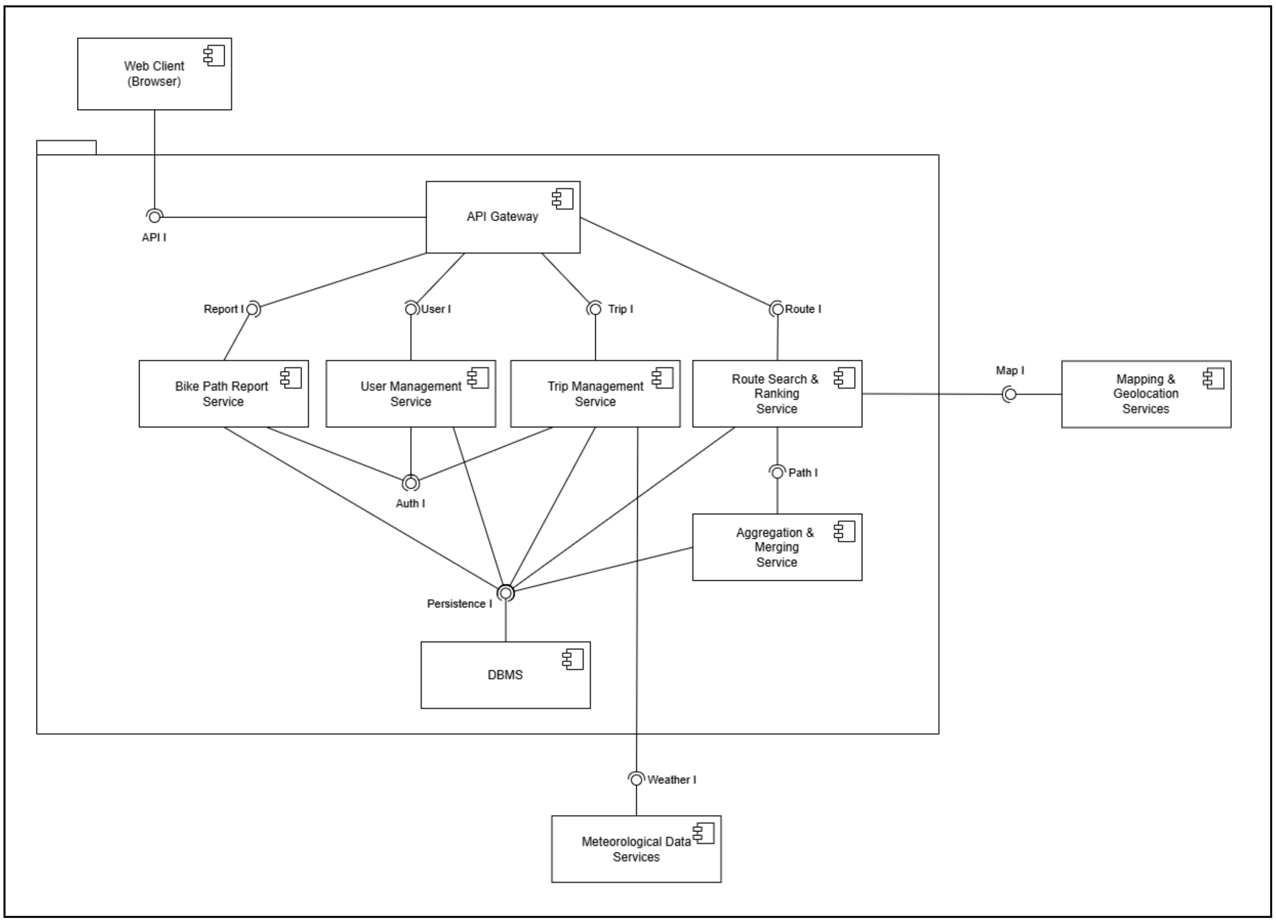
➔ **Mapping and Geolocation Services**

External mapping services provide geographic data, map visualization support, and path reconstruction capabilities. These services are accessed through the External API Proxy.

➔ **Meteorological Data Services**

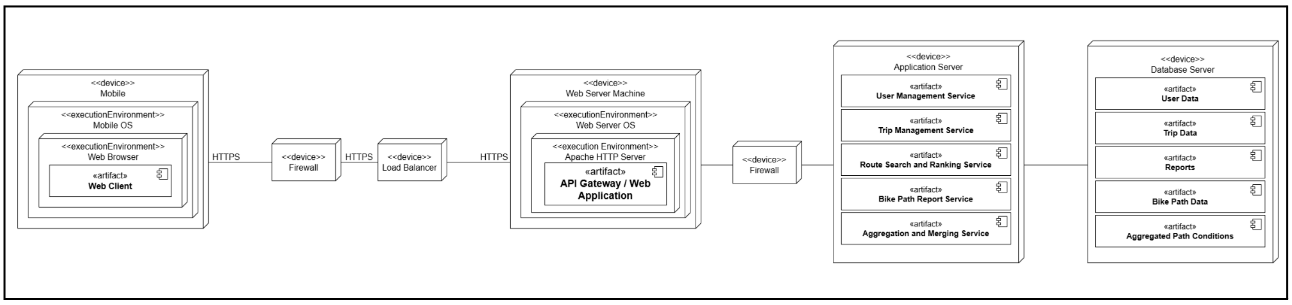
Weather services provide meteorological information associated with recorded trips. These services are queried by backend components when available, in accordance with the RASD assumptions.

N.B: Device sensors are not modeled as external components, as they do not represent software systems interacting with BBP, but rather physical data sources accessed by the web client.



2.3 Deploiment View

This chapter describes the deployment view of the BBP system. It outlines the execution environment of the system and details the geographical distribution of the hardware components that host the software making up the application.


<<device>> Mobile
<<executionEnvironment>> Mobile OS
<<executionEnvironment>> Web Browser
«artifact» Web Client
HTTPS
<<device>> Firewall
HTTPS
<<device>> Load Balancer
HTTPS
<<device>> Web Server Machine
<<executionEnvironment>> Web Server OS
<<execution Environment>> Apache HTTP Server
«artifact» API Gateway / Web Application
<<device>> Firewall
<<device>> Application Server
«artifact» User Management Service
«artifact» Trip Management Service
«artifact» Route Search and Ranking Service
«artifact» Bike Path Report Service
«artifact» Aggregation and Merging Service
<<device>> Database Server
«artifact» User Data
«artifact» Trip Data
«artifact» Reports
«artifact» Bike Path Data
«artifact» Aggregated Path Conditions

The application server hosts the microservices composing the application tier described in Section 2.1.

Client Device

A client device, such as a desktop computer, laptop, tablet, or smartphone, is used by users (guest or registered) to access the Best Bike Paths (BBP) application. The client only requires a standard web browser to interact with the system. Communication with the backend occurs over secure HTTPS connections. When supported by the device, the browser may access local capabilities such as geolocation sensors during trip recording activities.

Device sensors (such as GPS, accelerometer, and gyroscope) are considered **external to the BBP system** and are accessed exclusively through the web browser using standard client-side APIs. No sensor-related components are deployed on backend nodes.

Firewall

The firewall monitors all incoming network traffic directed to the BBP backend infrastructure. It filters requests based on predefined security rules, blocking potentially malicious packets before they reach internal components. Positioned at the network perimeter, the firewall contributes to protecting backend services and data from unauthorized access and common network-based attacks.

Load Balancer

The load balancer distributes incoming HTTPS requests among multiple backend entry-point instances. By evenly balancing traffic, it prevents overload of individual servers, improves system responsiveness, and increases availability. The load balancer also supports fault tolerance by redirecting traffic away from unavailable or failing backend nodes.

Web Server / API Gateway

The web server and API gateway are deployed on publicly accessible backend nodes. This component receives requests forwarded by the load balancer and acts as the entry point to the application logic.

Its responsibilities include:
- routing requests to the appropriate backend services,
- enforcing authentication and authorization policies,
- filtering and validating incoming requests,
- serving static frontend resources when applicable.

Requests requiring application logic are forwarded to the core application services using RESTful HTTP communication.

Application Server

The application services are deployed on one or more backend nodes hosting the BBP core functionality. These nodes run the backend microservices responsible for user management, trip processing, bike path reporting, aggregation, and route search and ranking.

Each service processes requests received from the API gateway, performs business logic and computations, and accesses persistent data when required. Services may be replicated across multiple nodes to support scalability and fault isolation.

Database Server

The database servers are deployed on dedicated nodes and managed by appropriate DBMS solutions. They provide persistent storage for user profiles, trips, bike path segments, reports, and aggregated path conditions.

Database servers are organized in a replicated or clustered configuration to ensure high availability, data consistency, and durability. Only backend application services are allowed to directly access the databases via secure network connections, preventing direct exposure of stored data.

2.4 Runtime View

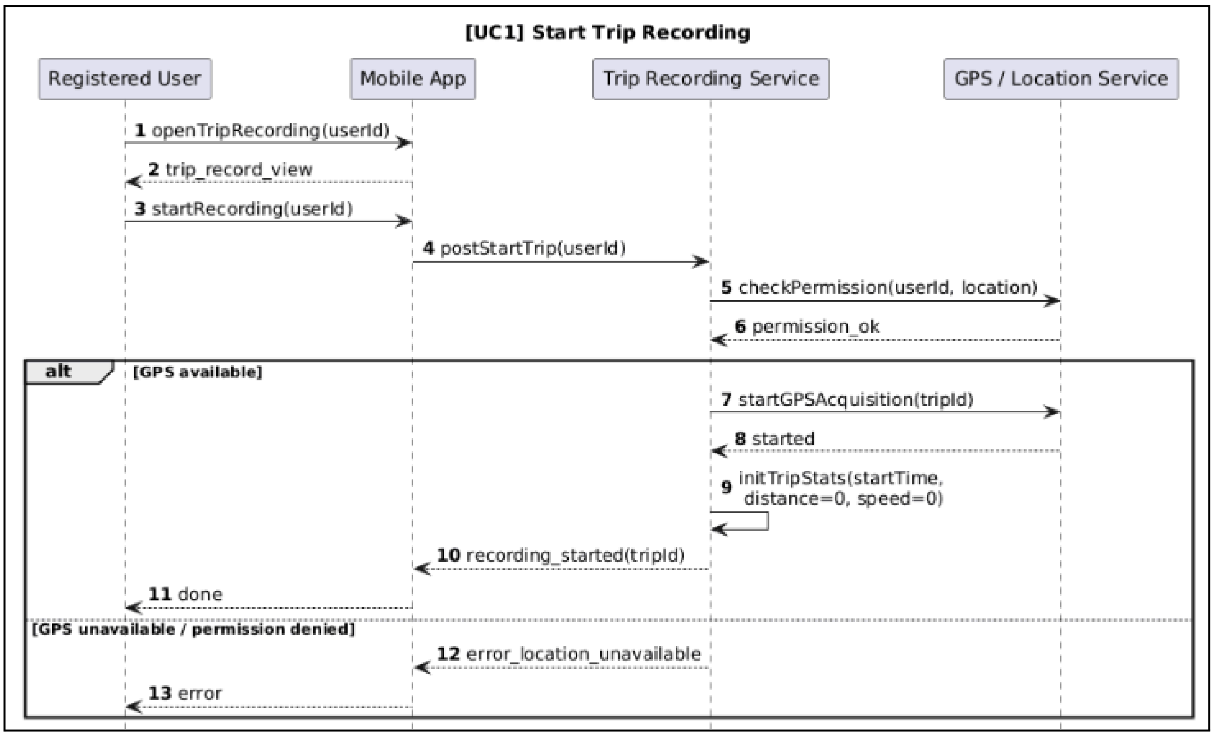
Sequence Diagrams Overview

In this section, we describe the flow of messages exchanged among the different system components by means of sequence diagrams, one for each use case presented in the previous section.
The diagrams illustrate how users, client applications, internal services, databases, and external services interact in order to support functionalities such as trip recording, automated data collection, bike path reporting, and route search.

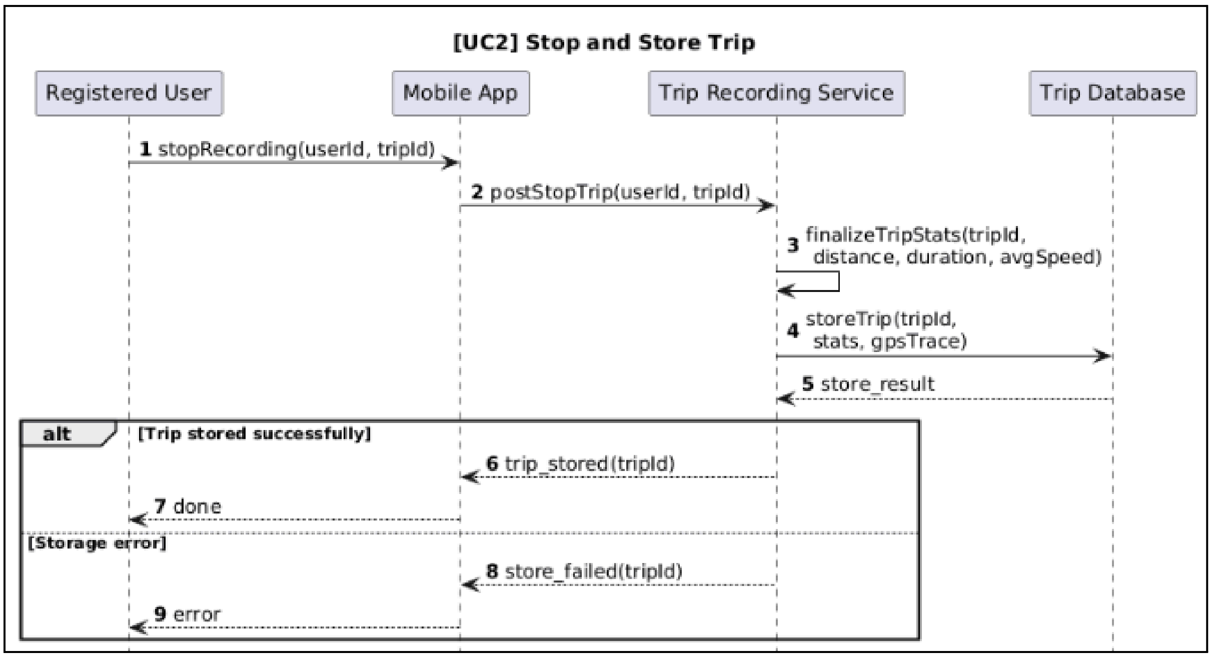
For the sake of clarity and readability, we deliberately omit some aspects that are not essential to understanding the main logical flow of the system. In particular, we do not explicitly represent the authorization and access control checks that ensure users can only perform actions permitted within their scope (e.g., accessing only their own trips or reports). Similarly, we exclude error handling related to invalid input parameters, temporary failures of external services (such as weather or map providers), and other exceptional conditions.

Moreover, although some components (e.g., mobile applications or backend services) may internally use caching mechanisms to avoid repeated database access, this behavior is not shown in the diagrams. Instead, we represent the general case in which each request interacts with the persistence layer, in order to keep the diagrams consistent and easier to interpret.

Despite these simplifications, the sequence diagrams accurately capture the core interactions and message exchanges required to execute each use case, providing a clear and structured view of the system's dynamic behavior.

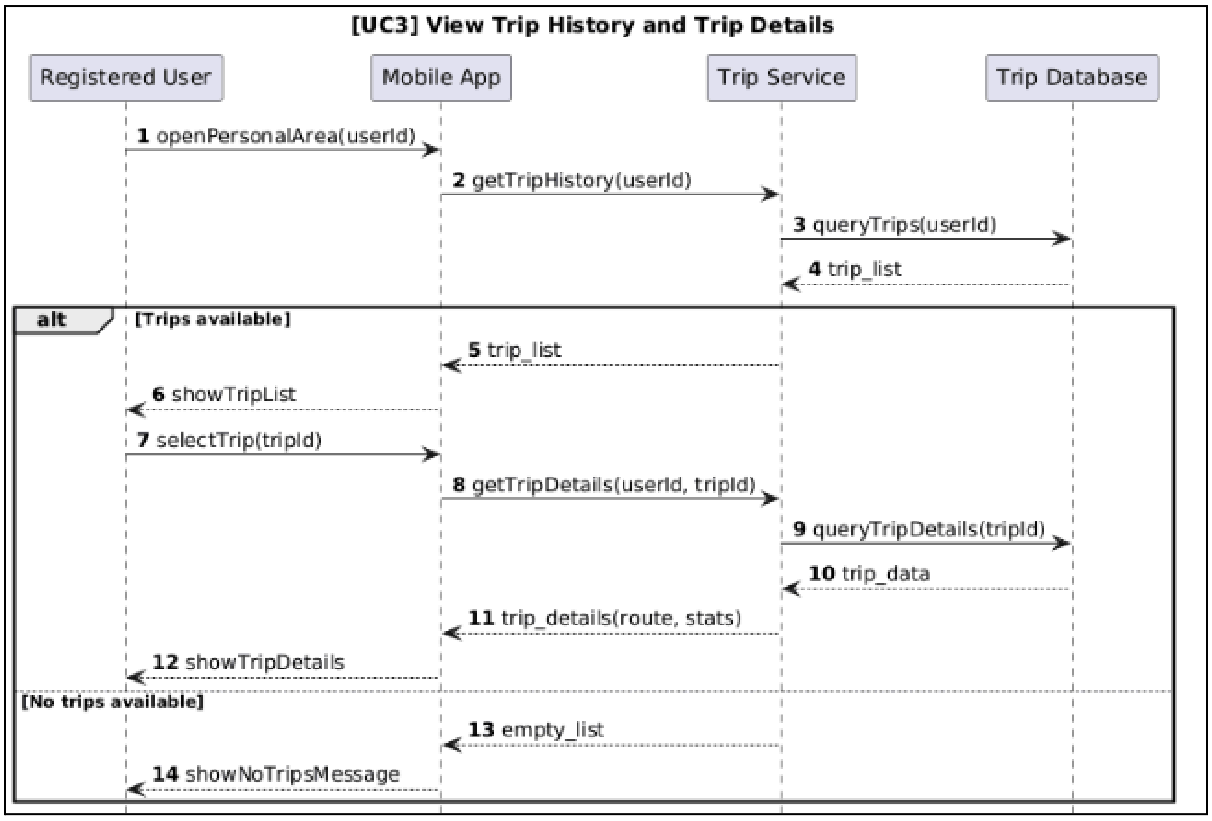

[UC1] Start Trip Recording
Registered User
Mobile App
Trip Recording Service
GPS / Location Service
1 openTripRecording(userId)
2 trip_record_view
3 startRecording(userId)
4 postStartTrip(userId)
5 checkPermission(userId, location)
6 permission_ok
alt
[GPS available]
7 startGPSAcquisition(tripId)
8 started
9 initTripStats(startTime, distance=0, speed=0)
10 recording_started(tripId)
11 done
[GPS unavailable / permission denied]
12 error_location_unavailable
13 error

This sequence diagram shows the flow of messages for the *Start Trip Recording* use case. The registered user initiates the trip recording process through the mobile application. The system verifies the availability of location permissions and GPS services and, if they are granted, starts acquiring location data. At the same time, the system initializes the main trip statistics, such as time, distance, and speed, and sets the trip state to *Recording*, allowing continuous data collection during the cycling activity.

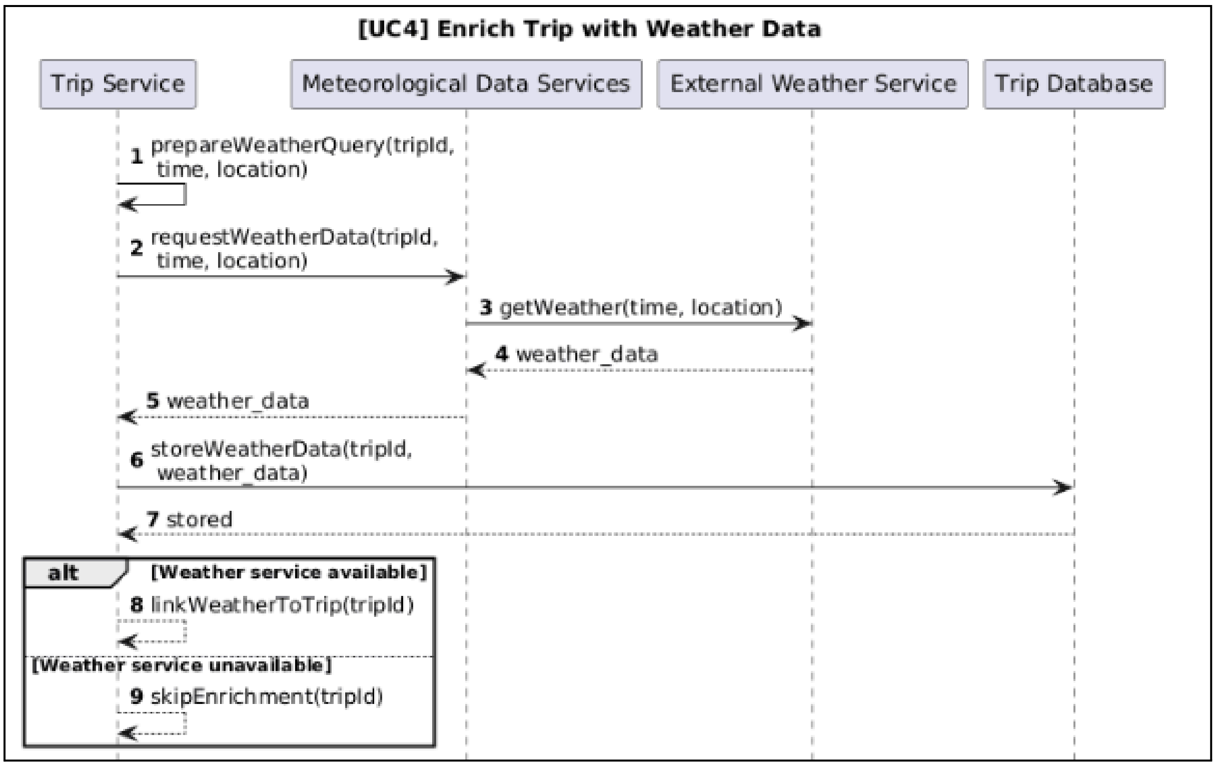

[UC2] Stop and Store Trip
Registered User
Mobile App
Trip Recording Service
Trip Database
1 stopRecording(userId, tripId)
2 postStopTrip(userId, tripId)
3 finalizeTripStats(tripId, distance, duration, avgSpeed)
4 storeTrip(tripId, stats, gpsTrace)
5 store_result
alt
[Trip stored successfully]
6 trip_stored(tripId)
7 done
[Storage error]
8 store_failed(tripId)
9 error

This sequence diagram illustrates the *Stop and Store Trip* use case.
When the user stops an ongoing recording, the system terminates GPS data acquisition and finalizes the trip statistics, including total distance, duration, and average speed. The

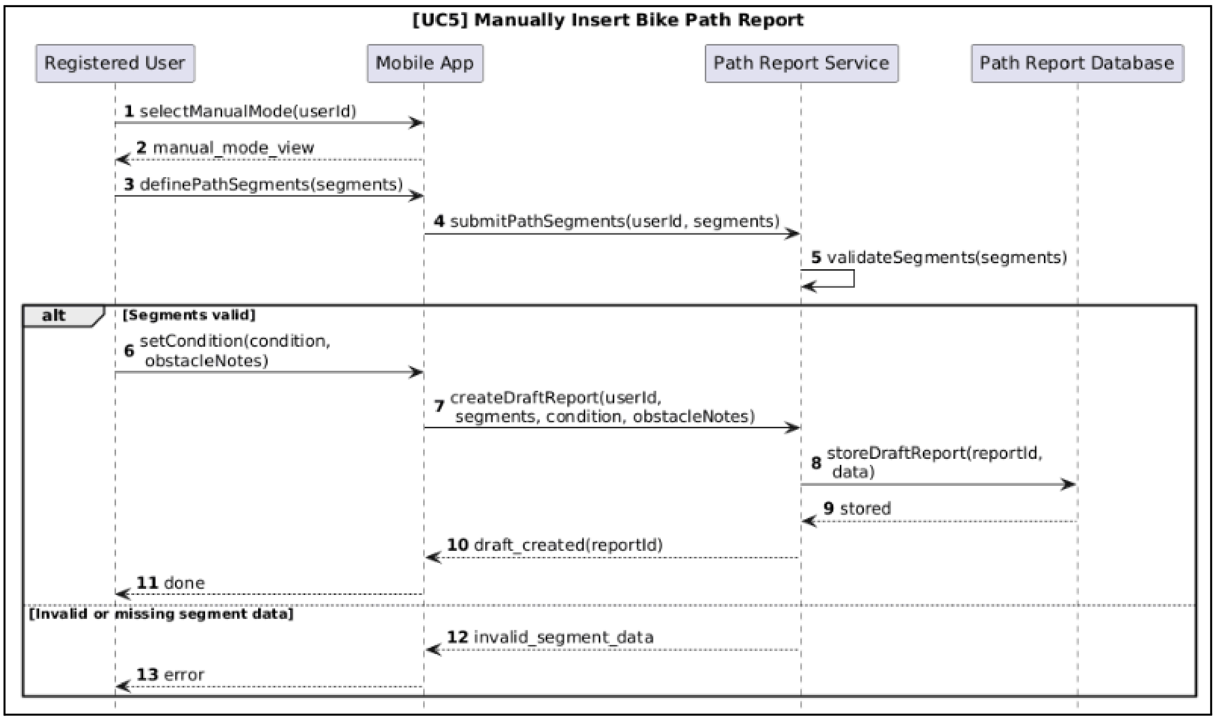
completed trip is then stored in the database and made available in the user's trip history. In case of storage issues, the trip remains unstored until the operation can be successfully completed.



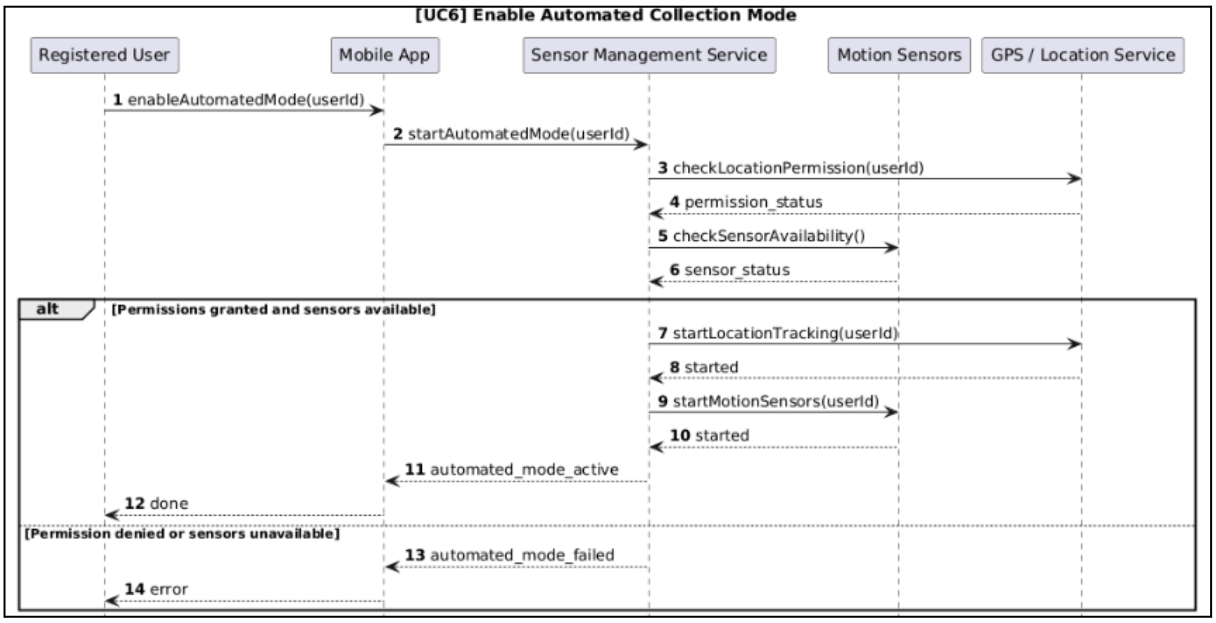
This sequence diagram represents the *View Trip History and Trip Details* use case.
The user accesses the personal area of the application to retrieve a list of previously recorded trips. After selecting a specific trip, the system loads and displays detailed information, including route visualization and statistical data. If no trips are available, the system informs the user accordingly.


[UC4] Enrich Trip with Weather Data
Trip Service
Meteorological Data Services
External Weather Service
Trip Database
1 prepareWeatherQuery(tripId, time, location)
2 requestWeatherData(tripId, time, location)
3 getWeather(time, location)
4 weather_data
5 weather_data
6 storeWeatherData(tripId, weather_data)
7 stored
alt
[Weather service available]
8 linkWeatherToTrip(tripId)
[Weather service unavailable]
9 skipEnrichment(tripId)

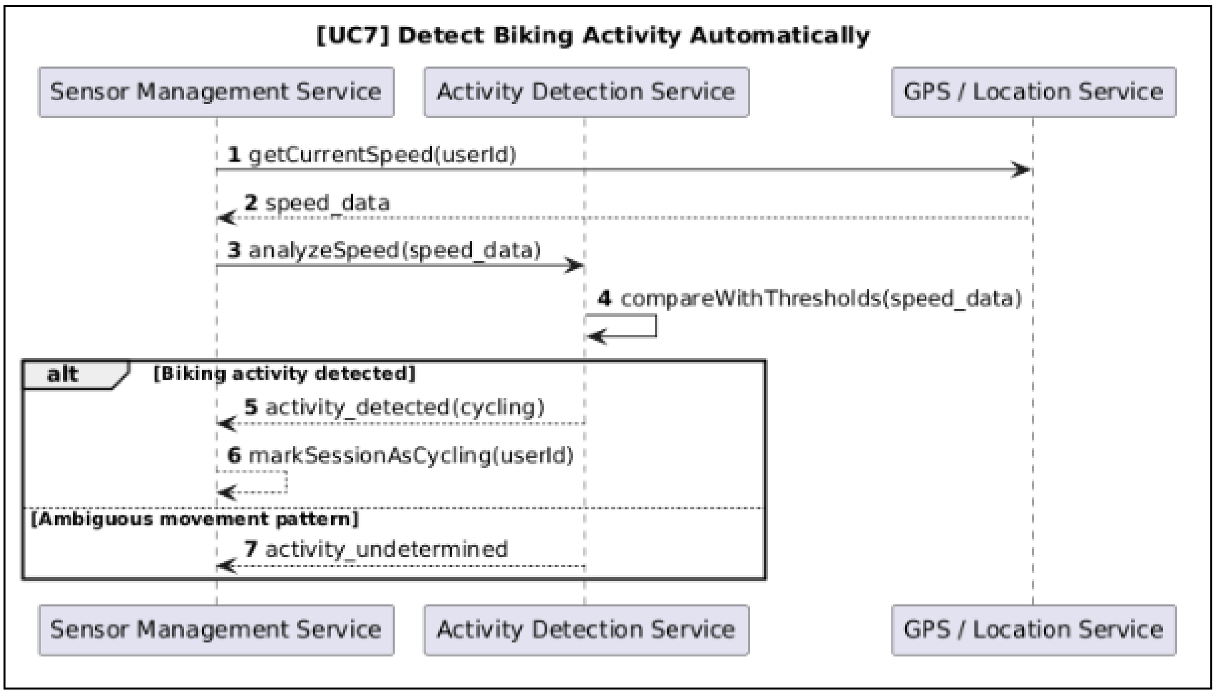
This sequence diagram describes the *Enrich Trip with Weather Data* use case.
Once a trip has been stored or finalized, the system prepares a query based on the trip's time and location and requests meteorological information from an external weather service. The retrieved data, such as weather conditions and temperature, is then associated with the trip and stored. If the external service is unavailable, the trip is saved without enrichment.

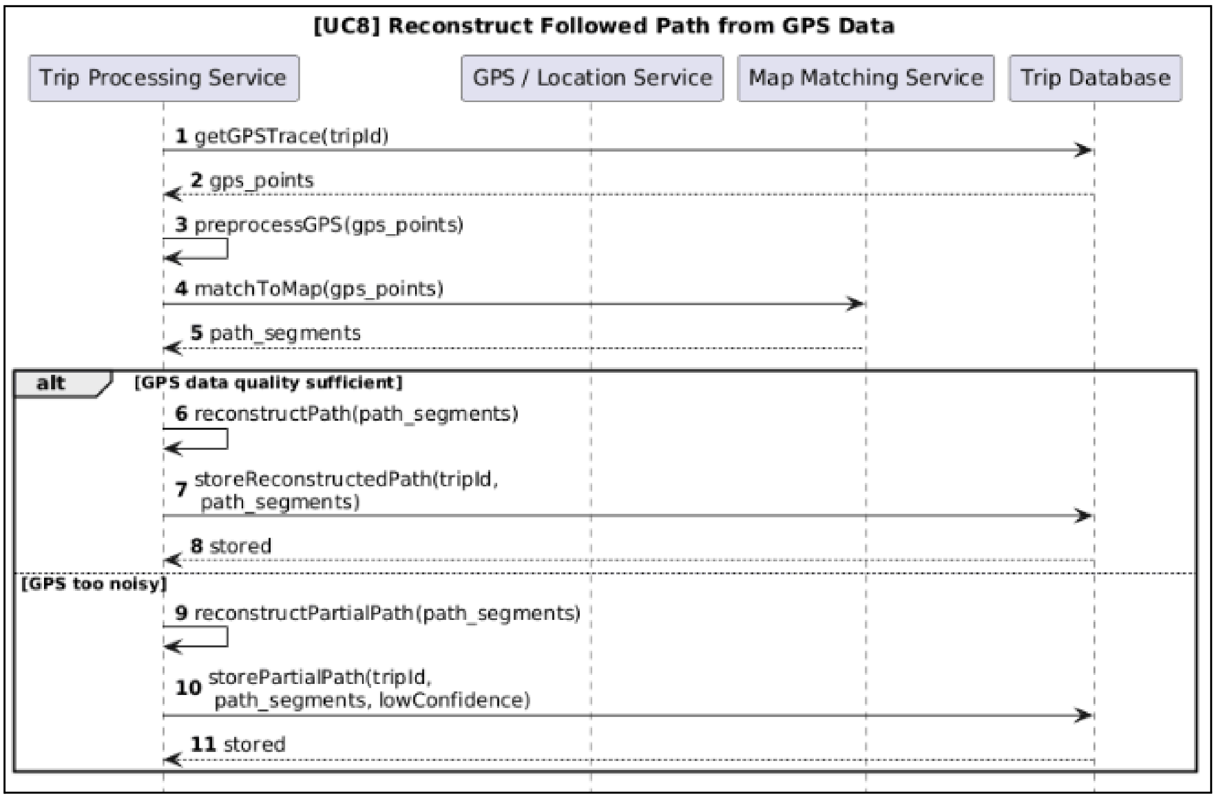

[UC5] Manually Insert Bike Path Report
Registered User
Mobile App
Path Report Service
Path Report Database
1 selectManualMode(userId)
2 manual_mode_view
3 definePathSegments(segments)
4 submitPathSegments(userId, segments)
5 validateSegments(segments)
alt
[Segments valid]
6 setCondition(condition, obstacleNotes)
7 createDraftReport(userId, segments, condition, obstacleNotes)
8 storeDraftReport(reportId, data)
9 stored
10 draft_created(reportId)
11 done
[Invalid or missing segment data]
12 invalid_segment_data
13 error

This sequence diagram shows the flow of the *Manually Insert Bike Path Report* use case. The user manually defines one or more bike path segments and assigns a condition level, optionally adding notes about obstacles. The system validates the provided data and stores the report in a *Draft* state, allowing further review or modification before confirmation.

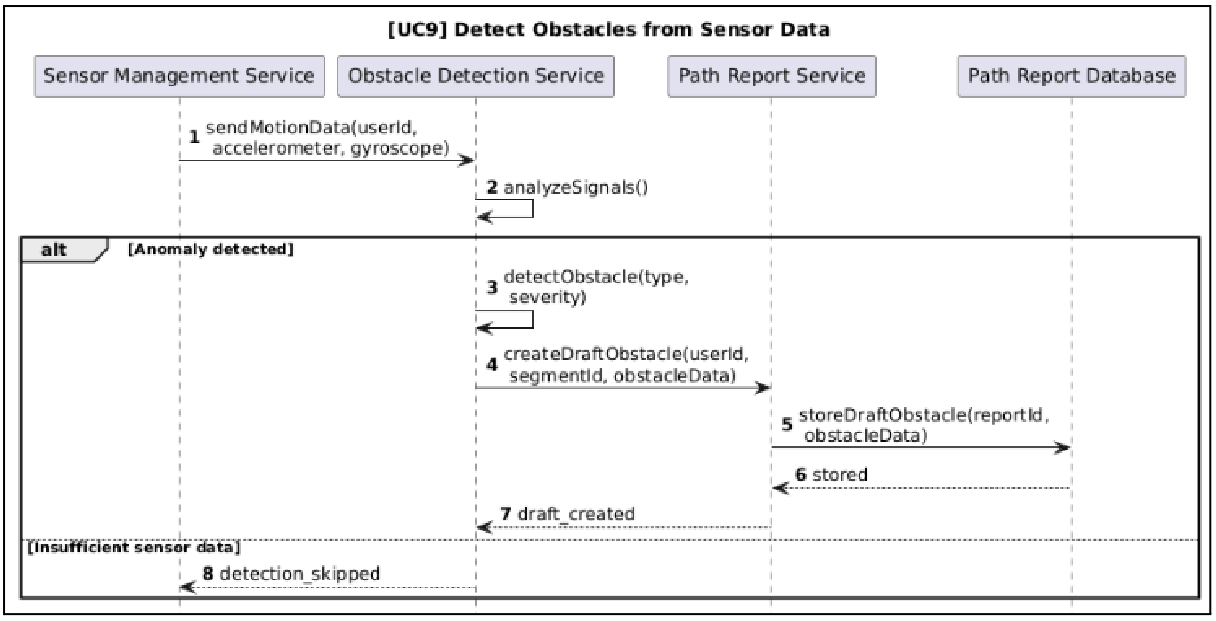

[UC6] Enable Automated Collection Mode
Registered User
Mobile App
Sensor Management Service
Motion Sensors
GPS / Location Service
1 enableAutomatedMode(userId)
2 startAutomatedMode(userId)
3 checkLocationPermission(userId)
4 permission_status
5 checkSensorAvailability()
6 sensor_status
alt
[Permissions granted and sensors available]
7 startLocationTracking(userId)
8 started
9 startMotionSensors(userId)
10 started
11 automated_mode_active
12 done
[Permission denied or sensors unavailable]
13 automated_mode_failed
14 error

This sequence diagram illustrates the *Enable Automated Collection Mode* use case. Before starting a cycling activity, the user enables the automated mode. The system checks the availability of location and motion sensors and, if all permissions are granted, starts collecting GPS and sensor data in the background. This mode allows the system to automatically detect cycling activity without requiring explicit user interaction.

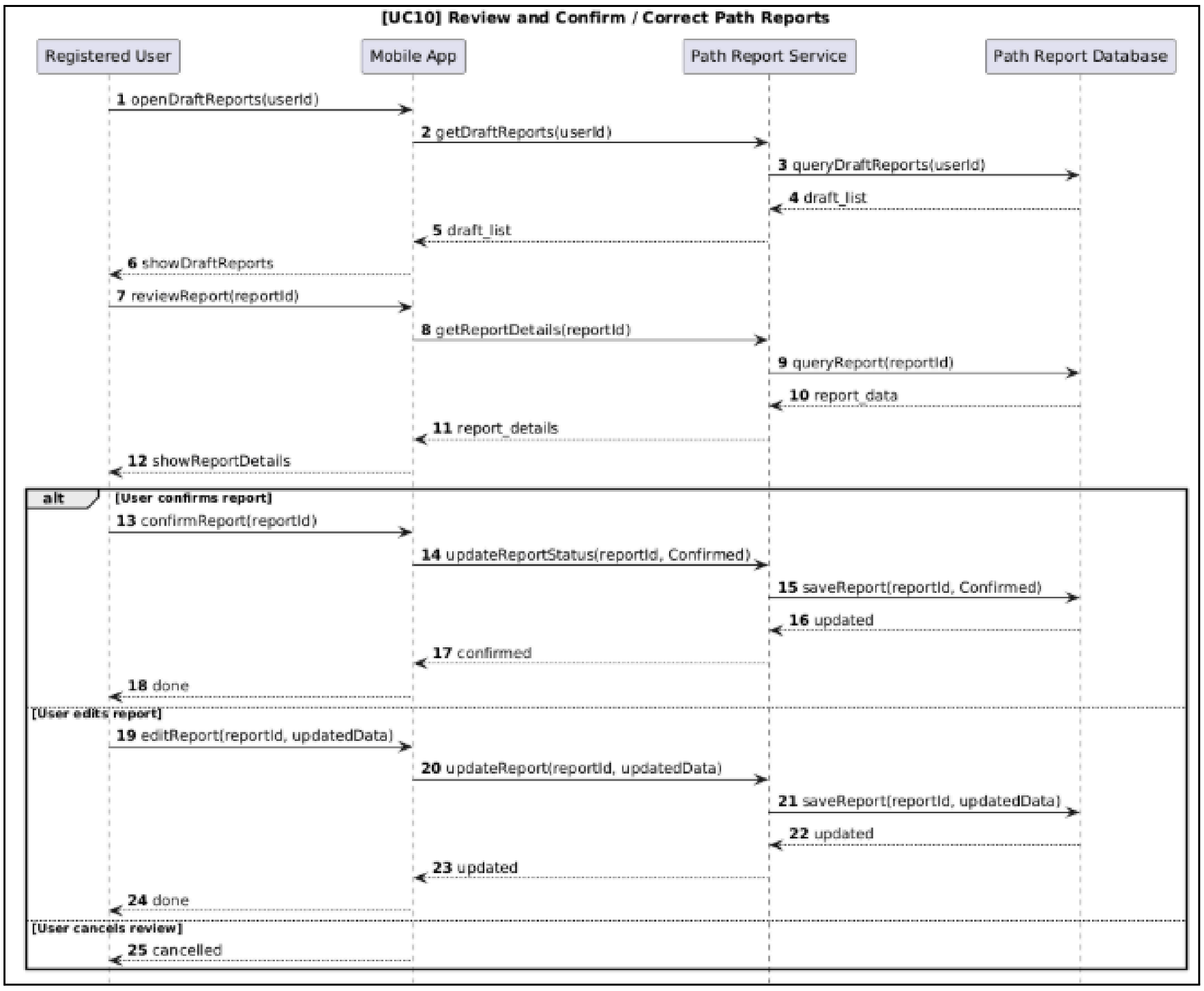

[UC7] Detect Biking Activity Automatically
Sensor Management Service
Activity Detection Service
GPS / Location Service
1 getCurrentSpeed(userId)
2 speed_data
3 analyzeSpeed(speed_data)
4 compareWithThresholds(speed_data)
alt
[Biking activity detected]
5 activity_detected(cycling)
6 markSessionAsCycling(userId)
[Ambiguous movement pattern]
7 activity_undetermined
This sequence diagram represents the *Detect Biking Activity Automatically* use case. While automated mode is active, the system continuously analyzes speed data derived from location information. By comparing the observed values with predefined thresholds, the system determines whether the user is cycling. If the movement pattern is ambiguous, the system continues monitoring without confirming the activity.


[UC8] Reconstruct Followed Path from GPS Data
Trip Processing Service
GPS / Location Service
Map Matching Service
Trip Database
1 getGPSTrace(tripId)
2 gps_points
3 preprocessGPS(gps_points)
4 matchToMap(gps_points)
5 path_segments
alt
[GPS data quality sufficient]
6 reconstructPath(path_segments)
7 storeReconstructedPath(tripId, path_segments)
8 stored
[GPS too noisy]
9 reconstructPartialPath(path_segments)
10 storePartialPath(tripId, path_segments, lowConfidence)
11 stored

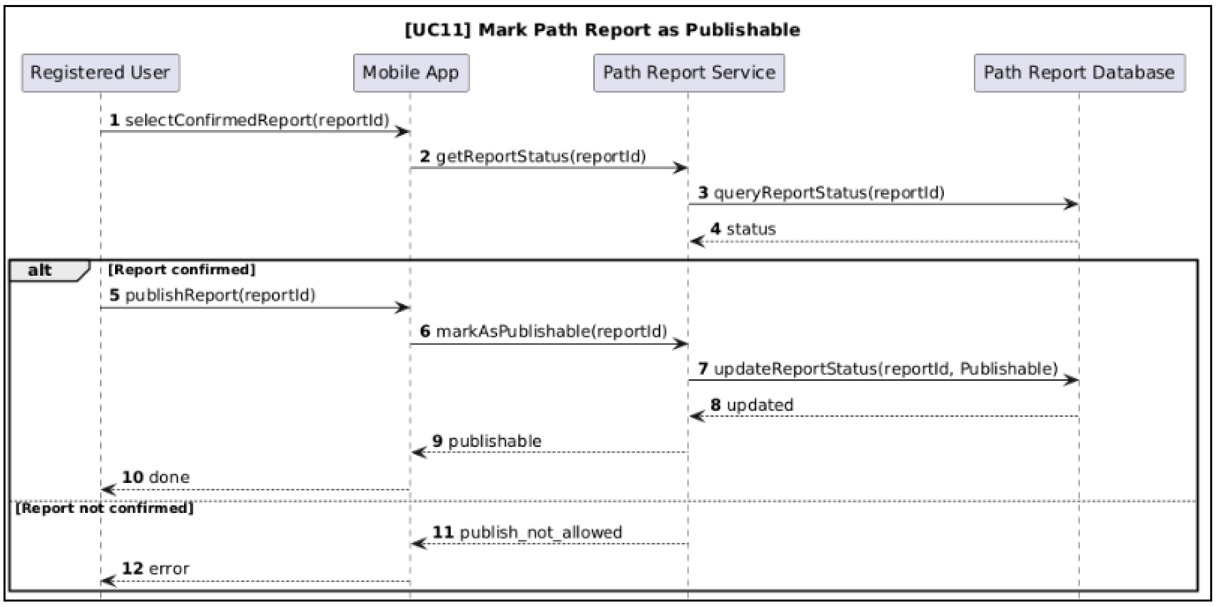
This sequence diagram describes the *Reconstruct Followed Path from GPS Data* use case. After a cycling session, the system processes the collected GPS points and reconstructs the traveled path. The reconstructed route is matched with map data to identify the corresponding path segments and is then stored for visualization and further analysis. In case of noisy data, a partial reconstruction is produced and marked with low confidence.


[UC9] Detect Obstacles from Sensor Data
Sensor Management Service
Obstacle Detection Service
Path Report Service
Path Report Database
1 sendMotionData(userId, accelerometer, gyroscope)
2 analyzeSignals()
alt
[Anomaly detected]
3 detectObstacle(type, severity)
4 createDraftObstacle(userId, segmentId, obstacleData)
5 storeDraftObstacle(reportId, obstacleData)
6 stored
7 draft_created
[Insufficient sensor data]
8 detection_skipped

This sequence diagram shows the flow of the *Detect Obstacles from Sensor Data* use case. During automated data collection, the system analyzes accelerometer and gyroscope signals to identify anomalies that may correspond to obstacles, such as potholes or uneven surfaces. Each detected issue is associated with a specific path segment and stored as a draft obstacle report for later review.

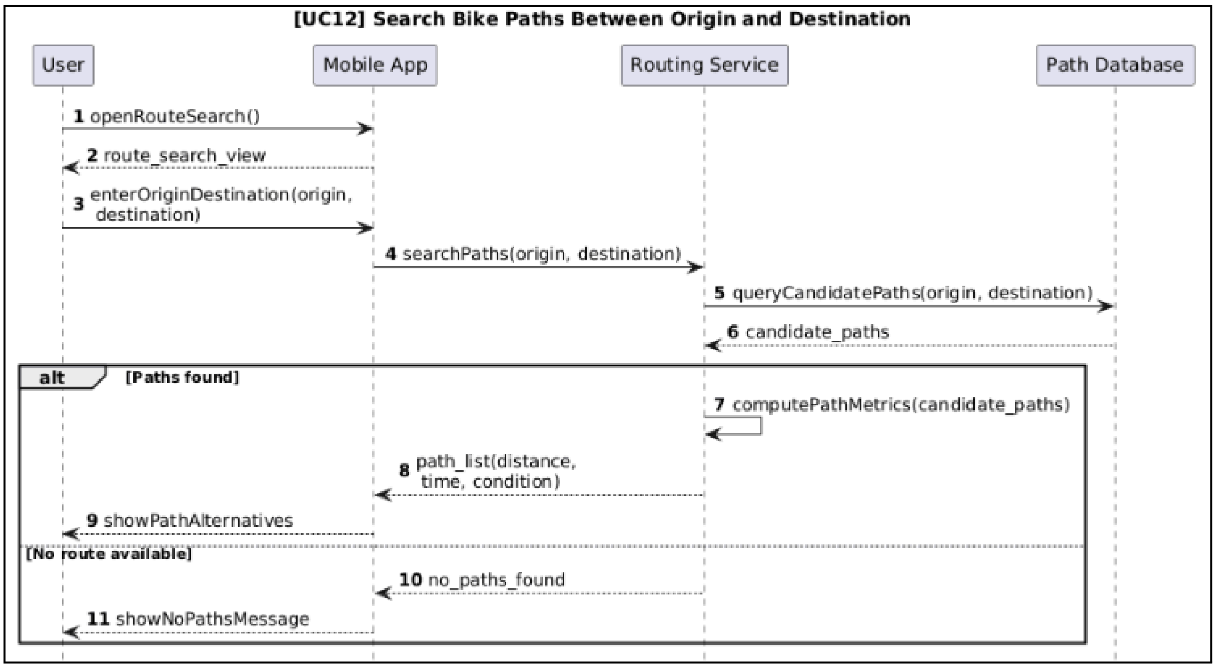

[UC10] Review and Confirm / Correct Path Reports
Registered User
Mobile App
Path Report Service
Path Report Database
1 openDraftReports(userId)
2 getDraftReports(userId)
3 queryDraftReports(userId)
4 draft_list
5 draft_list
6 showDraftReports
7 reviewReport(reportId)
8 getReportDetails(reportId)
9 queryReport(reportId)
10 report_data
11 report_details
12 showReportDetails
alt
[User confirms report]
13 confirmReport(reportId)
14 updateReportStatus(reportId, Confirmed)
15 saveReport(reportId, Confirmed)
16 updated
17 confirmed
18 done
[User edits report]
19 editReport(reportId, updatedData)
20 updateReport(reportId, updatedData)
21 saveReport(reportId, updatedData)
22 updated
23 updated
24 done
[User cancels review]
25 cancelled

This sequence diagram illustrates the *Review and Confirm/Correct Path Reports* use case. The user reviews draft reports generated either manually or automatically. The system displays the report details, allowing the user to confirm the information or edit incorrect data. Once confirmed, the report state is updated accordingly; if the user cancels the operation, the report remains in the draft state.


[UC11] Mark Path Report as Publishable
Registered User
Mobile App
Path Report Service
Path Report Database
1 selectConfirmedReport(reportId)
2 getReportStatus(reportId)
3 queryReportStatus(reportId)
4 status
alt
[Report confirmed]
5 publishReport(reportId)
6 markAsPublishable(reportId)
7 updateReportStatus(reportId, Publishable)
8 updated
9 publishable
10 done
[Report not confirmed]
11 publish_not_allowed
12 error

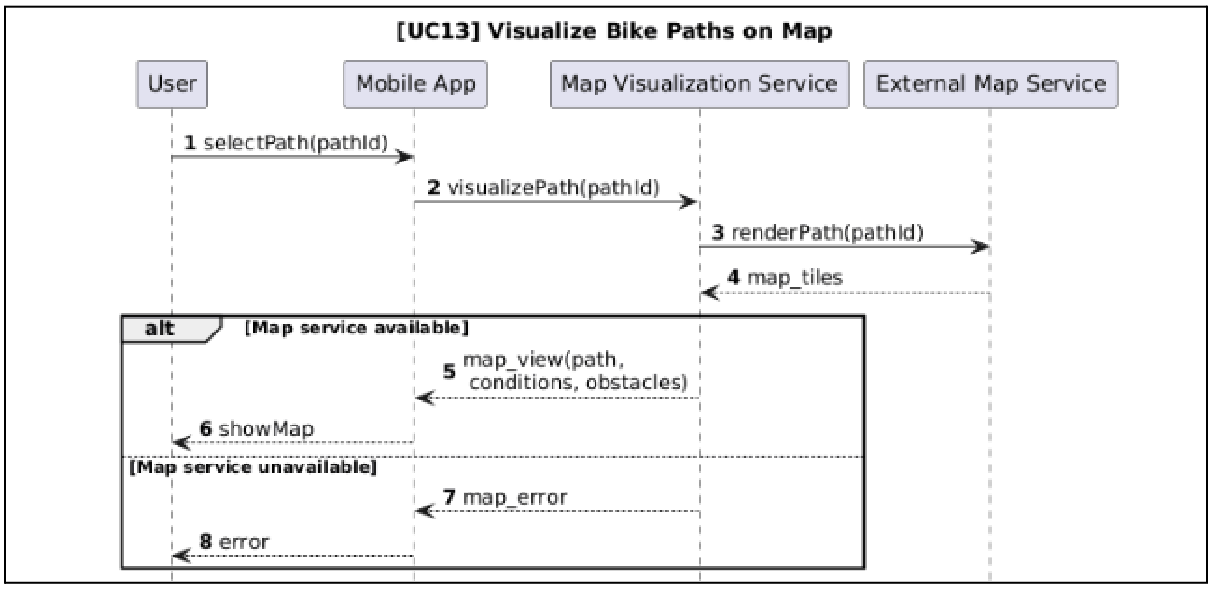
This sequence diagram represents the *Mark Path Report as Publishable* use case.

After a report has been confirmed, the user can mark it as publishable. The system verifies the report status and, if the conditions are satisfied, updates its state to *Publishable*, making it available for aggregation and use by the community. Reports that are not confirmed cannot be published.

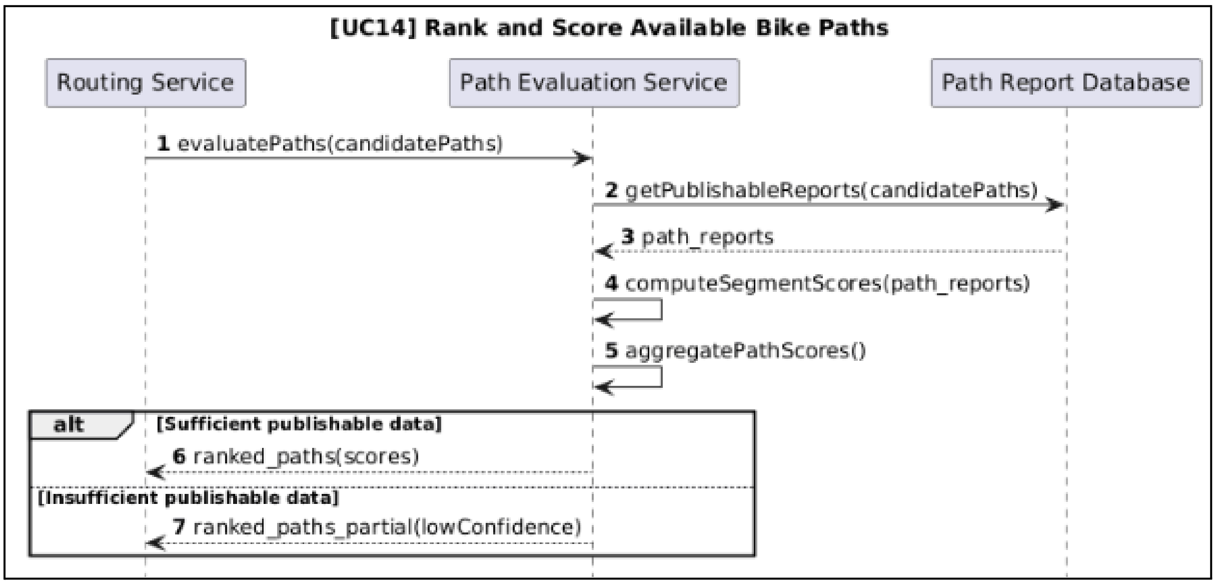

[UC12] Search Bike Paths Between Origin and Destination
User
Mobile App
Routing Service
Path Database
1 openRouteSearch()
2 route_search_view
3 enterOriginDestination(origin, destination)
4 searchPaths(origin, destination)
5 queryCandidatePaths(origin, destination)
6 candidate_paths
alt
[Paths found]
7 computePathMetrics(candidate_paths)
8 path_list(distance, time, condition)
9 showPathAlternatives
[No route available]
10 no_paths_found
11 showNoPathsMessage

This sequence diagram describes the *Search Bike Paths Between Origin and Destination* use case.
The user enters an origin and a destination, and the system retrieves candidate bike paths connecting the two locations. For each alternative, aggregated information such as distance,

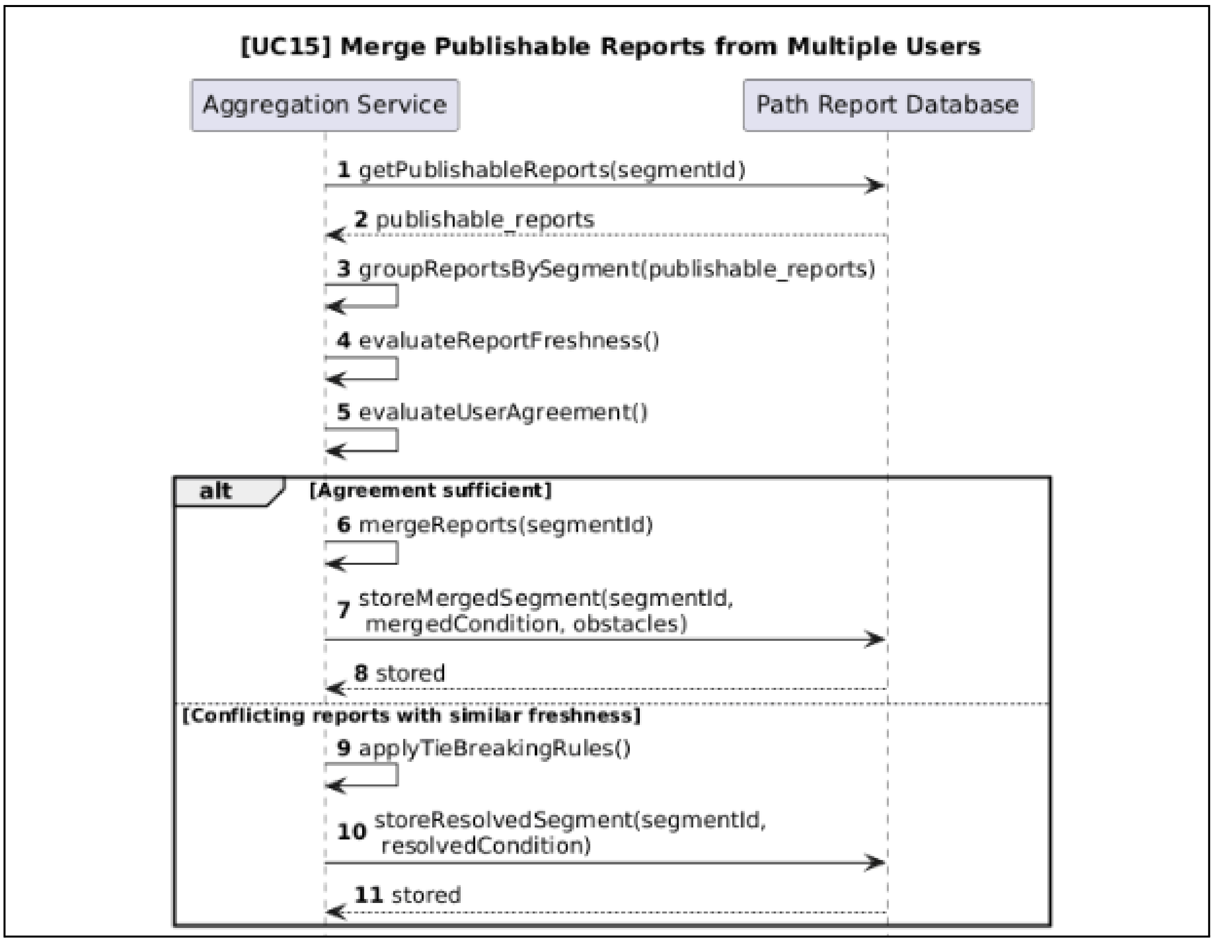
estimated travel time, and overall condition is computed and presented to the user. If no suitable paths are found, the system displays a corresponding message.


[UC13] Visualize Bike Paths on Map
User
Mobile App
Map Visualization Service
External Map Service
1 selectPath(pathId)
2 visualizePath(pathId)
3 renderPath(pathId)
4 map_tiles
alt [Map service available]
5 map_view(path, conditions, obstacles)
6 showMap
[Map service unavailable]
7 map_error
8 error

This sequence diagram shows the *Visualize Bike Paths on Map* use case.
After selecting one or more bike paths, the system renders the chosen routes on a map. Additional information, such as segment conditions and detected obstacles, is displayed to provide a comprehensive overview. If the map service is unavailable, an error message is shown to the user.


[UC14] Rank and Score Available Bike Paths
Routing Service
Path Evaluation Service
Path Report Database
1 evaluatePaths(candidatePaths)
2 getPublishableReports(candidatePaths)
3 path_reports
4 computeSegmentScores(path_reports)
5 aggregatePathScores()
alt [Sufficient publishable data]
6 ranked_paths(scores)
[Insufficient publishable data]
7 ranked_paths_partial(lowConfidence)

This sequence diagram illustrates the *Rank and Score Available Bike Paths* use case.
Based on publishable reports and aggregated segment data, the system evaluates and scores each candidate path. The paths are then ranked according to their overall quality and presented to the user. When the available data is insufficient, the ranking is still produced but marked with low confidence.


[UC15] Merge Publishable Reports from Multiple Users
Aggregation Service
Path Report Database
1 getPublishableReports(segmentId)
2 publishable_reports
3 groupReportsBySegment(publishable_reports)
4 evaluateReportFreshness()
5 evaluateUserAgreement()
alt
[Agreement sufficient]
6 mergeReports(segmentId)
7 storeMergedSegment(segmentId, mergedCondition, obstacles)
8 stored
[Conflicting reports with similar freshness]
9 applyTieBreakingRules()
10 storeResolvedSegment(segmentId, resolvedCondition)
11 stored

This sequence diagram represents the *Merge Publishable Reports from Multiple Users* use case.
The system groups publishable reports referring to the same path segment and evaluates their freshness and consistency. A consolidated representation of segment conditions and obstacles is then generated and stored. In the presence of conflicting reports with similar relevance, predefined tie-breaking rules are applied.

2.5 Component interfaces

This section provides a summary of the interfaces and methods exposed by the main components of the system. All internal components communicate through REST-based APIs. Method names follow a naming convention that includes an initial prefix indicating the type of operation performed: `get` for retrieval operations, `post` for creation or execution of actions, and `put` for updates of existing resources.

For some methods, overloading is assumed, meaning that the concrete implementation may slightly differ depending on the parameters provided. For external components, such as map and weather services, we assume the availability of a limited set of generic methods that are commonly offered by most providers, independently of the specific service used.

The class names used as method parameters were chosen to be as self-explanatory as possible. However, to avoid ambiguity, the following clarifications apply:

- **User** represents a registered user of the system.
- **Trip** represents a recorded cycling session, including GPS traces and aggregated statistics.
- **Path** represents a complete bike path composed of one or more segments.
- **PathSegment** represents an individual segment of a bike path.
- **PathReport** represents a user-generated report describing the condition of a path segment and possible obstacles.
- **SensorData** represents data collected from motion sensors (e.g., accelerometer and gyroscope).
- **WeatherData** represents meteorological information retrieved from an external weather service.

API Interfaces

The following interfaces are exposed to the client application and support trip management, path reporting, route search, and visualization functionalities.

- `get_tripHistory(User)`
- `get_tripDetails(Trip)`
- `post_startTrip(User)`
- `post_stopTrip(User, Trip)`
- `get_tripMap(Trip)`
- `post_enableAutomatedMode(User)`
- `post_disableAutomatedMode(User)`
- `post_manualPathReport(User, List<PathSegment>, Condition, Notes)`
- `get_draftPathReports(User)`
- `put_confirmPathReport(PathReport)`
- `put_publishPathReport(PathReport)`
- `get_searchPaths(Location, Location)`
- `get_rankedPaths(Location, Location)`
- `get_visualizePath(Path)`
- `get_pathDetails(Path)`

Trip Service Interface

This component manages all operations related to trip recording and storage.

- `startRecording(User)`
- `stopRecording(User, Trip)`
- `initializeTripStats(Trip)`
- `finalizeTripStats(Trip)`
- `storeTrip(Trip)`
- `queryTripHistory(User)`
- `queryTripDetails(Trip)`

Path Report Service Interface

This component handles the creation, review, confirmation, and publication of bike path reports.

- `createDraftReport(User, PathSegment, Condition, Notes)`
- `updateDraftReport(PathReport)`
- `confirmReport(PathReport)`
- `markAsPublishable(PathReport)`
- `queryDraftReports(User)`
- `queryPublishableReports(PathSegment)`

Routing and Evaluation Service Interface

This component is responsible for path search, evaluation, scoring, and ranking.

- `searchCandidatePaths(Location, Location)`
- `evaluatePathSegments(Path)`
- `computePathScore(Path)`
- `rankPaths(List<Path>)`

Sensor and Activity Detection Service Interface

This component processes sensor data collected during automated mode and detects cycling activity and obstacles.

- `startSensorCollection(User)`
- `stopSensorCollection(User)`
- `analyzeSpeed(SensorData)`
- `detectCyclingActivity(SensorData)`
- `detectObstacle(SensorData)`

External Services Interfaces

The system interacts with external services to enrich trip data and support visualization.

Weather Service

- `get_weather(Time, Location)`

Map Service

- `get_coordinates(List<Location>)`
- `render_path(Path)`

DBMS Interface

Each of the following methods corresponds to one or more parameterized database operations. Method prefixes indicate the type of operation performed: `query` for data retrieval, `store` for updates to existing objects, and `create` for the introduction of new entities. The actual implementation depends on the specific DBMS adopted.

- `create_trip(User, Trip)`
- `store_tripStats(Trip)`
- `store_gpsTrace(Trip, List<Coordinate>)`
- `query_tripHistory(User)`
- `query_tripDetails(Trip)`
- `create_pathReport(PathReport)`
- `store_pathReport(PathReport)`
- `query_pathReports(PathSegment)`
- `store_mergedSegment(PathSegment)`
- `query_candidatePaths(Location, Location)`

Despite the completeness of the interface list, authorization checks, error handling, and caching mechanisms are not explicitly represented in this section. These aspects are fundamental for a production-ready system but have been intentionally omitted to focus on the definition of component responsibilities and their interactions.

2.6 Selected Architectural Styles and Patterns

The Best Bike Paths (BBP) system adopts a set of well-established architectural styles and design patterns to satisfy the functional and non-functional requirements defined in the RASD, particularly those related to scalability, modularity, availability, and maintainability.

Layered Architecture

The overall system follows a **layered architectural style**, clearly separating concerns across different tiers: presentation, application, and data.
The presentation layer is responsible for user interaction and visualization, while the application layer encapsulates business logic and request processing. The data layer manages persistent storage and data consistency.

This separation improves maintainability by isolating changes within a single layer and enhances security by preventing direct access to backend data resources from the client side.

Microservices Architecture

The application tier is structured according to a microservices architecture, where each backend service encapsulates a specific domain responsibility, such as user management, trip processing, bike path reporting, data aggregation, and route search and ranking.

Each microservice exposes RESTful APIs and communicates with other services using HTTP-based protocols. This architectural choice allows:

- independent development and deployment of services,
- fine-grained scalability based on workload,
- fault isolation, preventing failures in one service from affecting the entire system.

This approach aligns with the need to support high user concurrency and evolving system functionalities.

API Gateway Pattern

An **API Gateway** is used as the single entry point to the backend system. It handles cross-cutting concerns such as request routing, authentication and authorization enforcement, input validation, and load distribution among backend services.

By centralizing these responsibilities, the system reduces duplication across microservices and improves security by limiting direct exposure of internal services to external clients.

Client–Server Pattern

The BBP system follows the client–server architectural pattern, where the Web Client acts as a thin client responsible only for presentation and interaction handling, while all business logic and data management are delegated to server-side components.

This pattern supports platform independence, as users only require a standard web browser to access the system.

RESTful Communication

Communication between components is based on RESTful architectural principles over HTTP(S). REST is used both for client-to-server interactions and for service-to-service communication within the application tier.

This choice ensures interoperability, simplicity, and ease of integration with external services, such as mapping and meteorological data providers.

2.7 Other Design Decisions

In addition to the selected architectural styles and patterns, several design decisions have been taken to improve the robustness, security, and extensibility of the BBP system.

Stateless Backend Services

Backend microservices are designed to be **stateless**, with all persistent information stored in the data tier. Statelessness simplifies horizontal scaling and supports efficient load balancing across replicated service instances.

Security-Oriented Design

Security considerations strongly influenced the system design. All communications between the client and backend components occur over secure HTTPS connections.
A firewall protects the backend infrastructure from unauthorized access, while the API gateway enforces authentication and authorization policies for protected resources.

Direct access to the data tier is restricted exclusively to backend application services, preventing data exposure and reinforcing separation of concerns.

External Services Isolation

Interactions with third-party services, such as mapping and geolocation providers and meteorological data services, are handled exclusively by backend application services.
This decision avoids coupling external APIs with the presentation tier or the API gateway and allows the system to manage failures, changes, or unavailability of external services without impacting client-side functionality.

Scalability and Fault Tolerance

To ensure high availability, key backend components are replicated and deployed on multiple nodes. Load balancing mechanisms distribute incoming requests across available instances, while service replication allows the system to tolerate partial failures without service interruption.
This design supports incremental growth in user base and data volume.

Consistency Across Architectural Views

The architectural decisions presented in this section are consistent across the system, component, and deployment views. Logical components described in the component view are realized as deployable artifacts in the deployment view, ensuring traceability and coherence throughout the design document.

3| User Interface Design

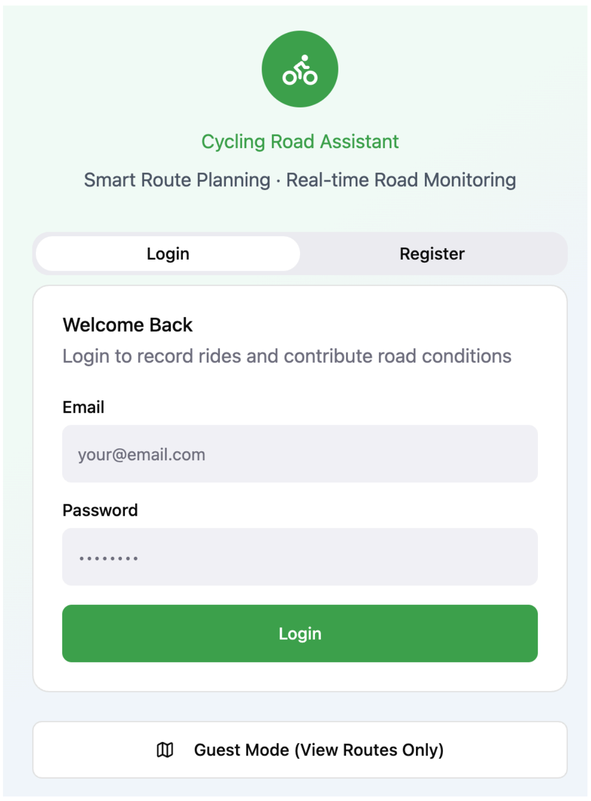
The user interface of the Best Bike Paths (BBP) system is designed to support cyclists in interacting with the system in a simple, intuitive, and efficient way, both during cycling activities and while browsing or managing information offline.
The system provides a responsive web-based interface, accessible from desktop and mobile devices. Special attention is given to mobile usability, as several core functionalities—such as trip recording and automated data collection—are performed while users are riding.
The user interface is organized around the different user roles and usage contexts, clearly separating functionalities available to registered users from those accessible to guest users.
The user interface design presented in this section is consistent with the interfaces described in the RASD and focuses on their role within the overall system architecture.

3.1 Access and Authentication Interface

BBP provides a unified access interface that allows users to:

- log in with existing credentials,
- register as new users,
- access public functionalities as guest users.

The authentication interface is designed to be minimal and straightforward, reducing friction for new users while ensuring secure access to personalized features.
Guest users can bypass authentication and directly access route exploration and bike path visualization functionalities.



Cycling Road Assistant

Smart Route Planning · Real-time Road Monitoring

Login | Register

Welcome Back

Login to record rides and contribute road conditions

Email

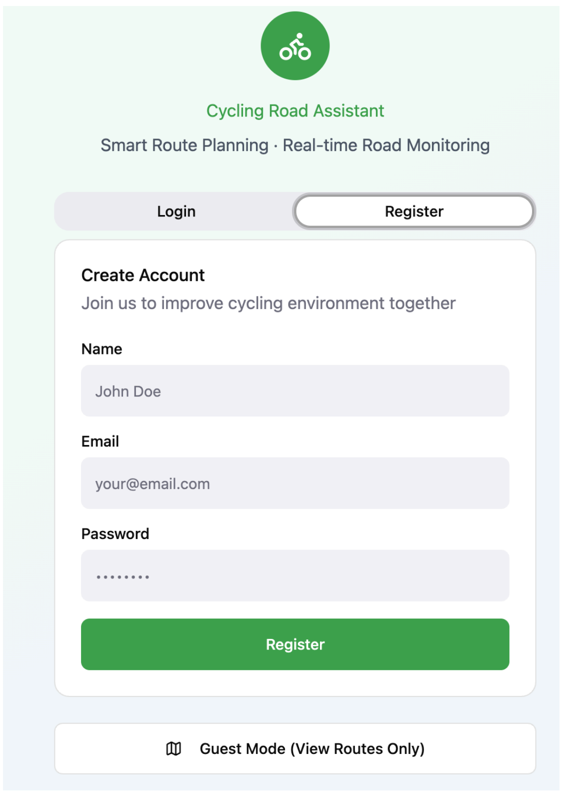
your@email.com

Password

••••••••

Login

Guest Mode (View Routes Only)



Cycling Road Assistant

Smart Route Planning · Real-time Road Monitoring

Login | Register

Create Account

Join us to improve cycling environment together

Name

John Doe

Email

your@email.com

Password

••••••••

Register

Guest Mode (View Routes Only)

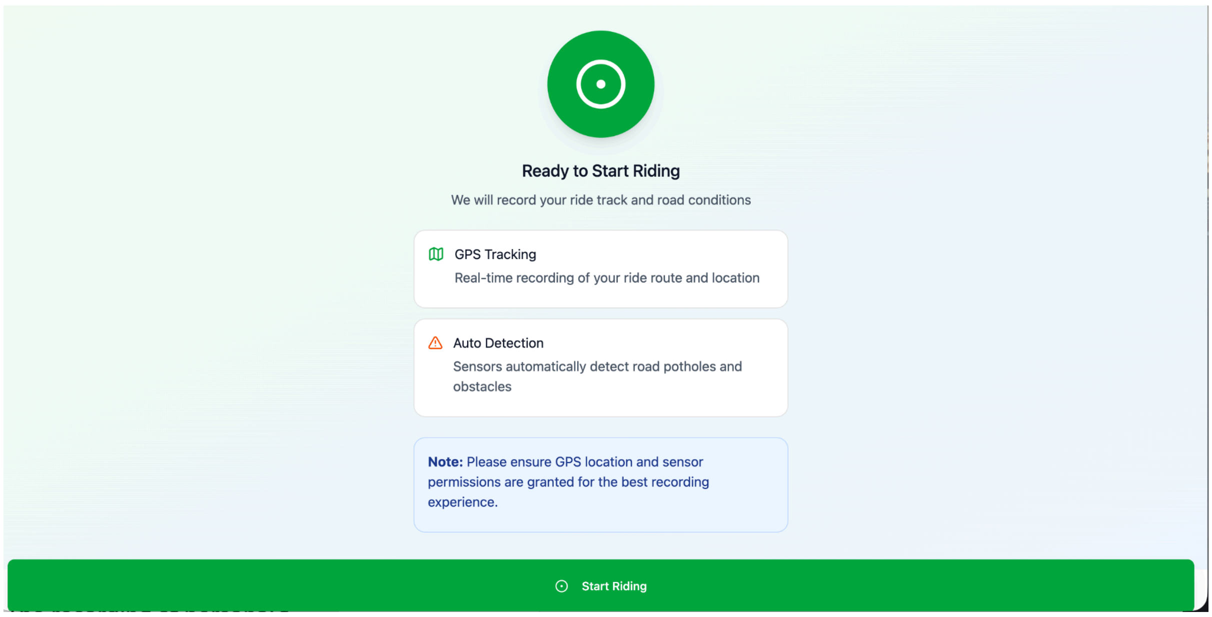
3.2 Trip Recording Interface

The trip recording interface is available to registered users and is optimized for real-time interaction during cycling activities.
Through this interface, users can:

- start and stop trip recording,

- monitor the recording status,

During an active recording session, the interface provides clear feedback on the recording state while minimizing on-screen interactions to avoid distracting the user. Detailed statistics and visualizations are presented only after the trip has been completed and stored.



Ready to Start Riding

We will record your ride track and road conditions

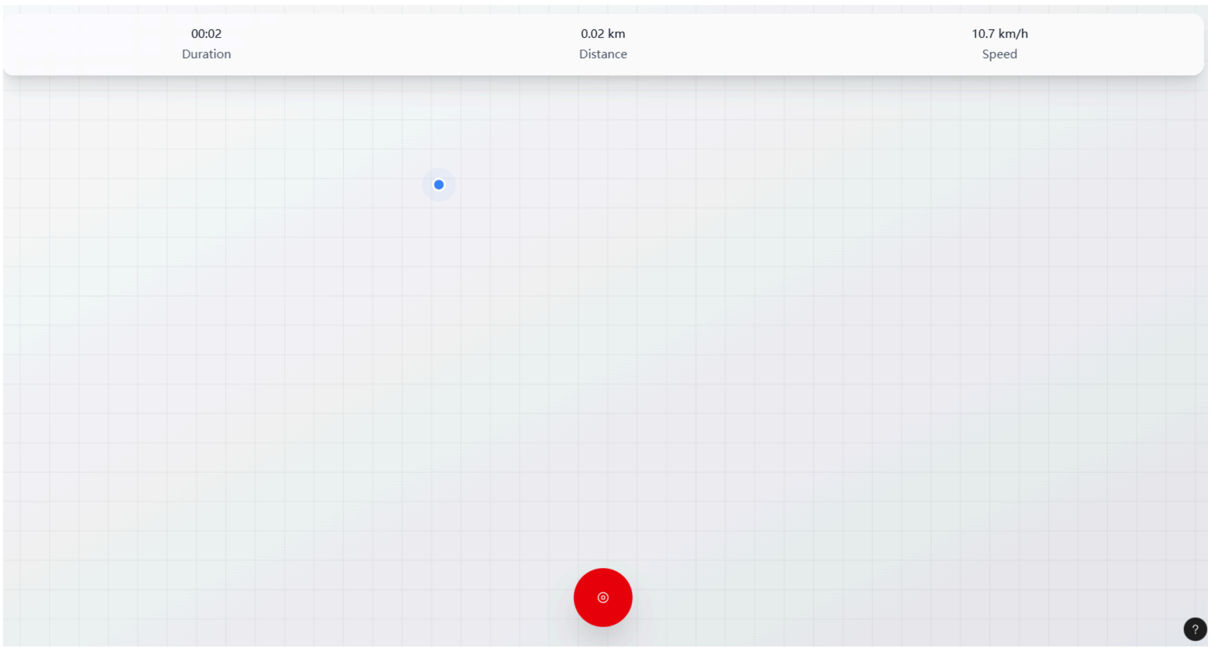
GPS Tracking

Real-time recording of your ride route and location

Auto Detection

Sensors automatically detect road potholes and obstacles

Note: Please ensure GPS location and sensor permissions are granted for the best recording experience.



00:02	0.02 km	10.7 km/h
Duration	Distance	Speed

3.3 Bike Path Reporting and Confirmation Interface

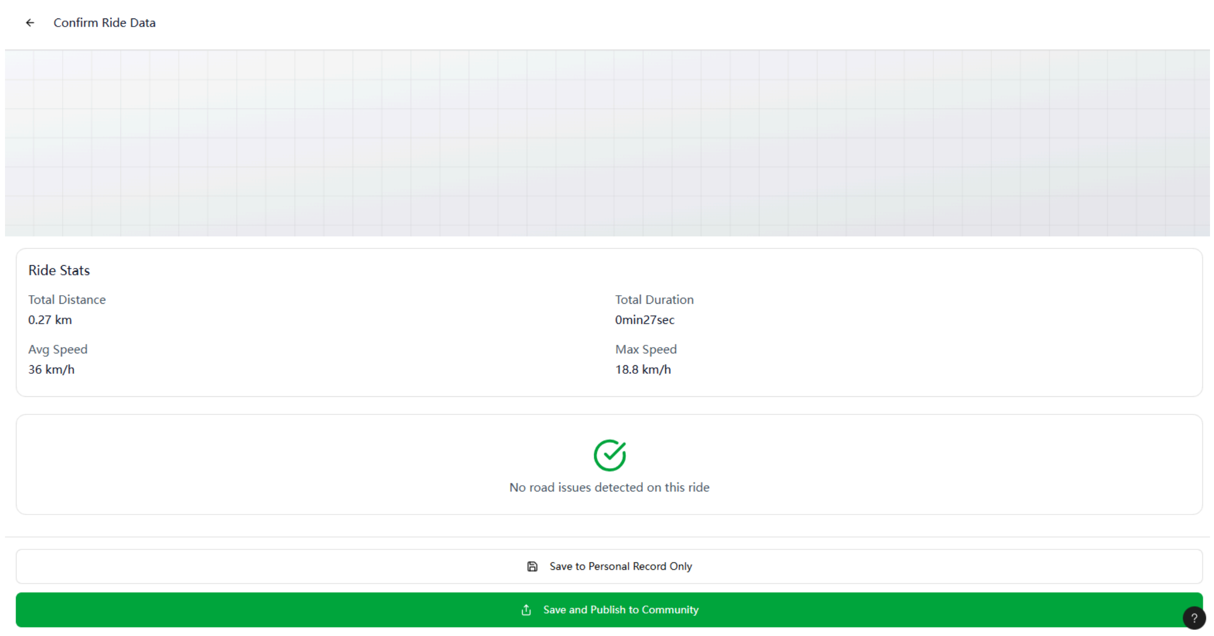
The system provides dedicated interfaces for managing bike path information, supporting both manual and automated reporting workflows.
In manual mode, users can:

- define bike path segments,
- assign conditions to each segment,
- optionally add notes about obstacles.

In automated mode, after a cycling session, the system presents the user with a list of automatically generated draft reports.
A confirmation interface allows users to:

- review detected path segments and issues,
- confirm correct information,

Only reports that successfully pass this validation step can be marked as publishable, ensuring data quality and reliability before information is shared with the community.



Confirm Ride Data

Ride Stats

Total Distance
0.27 km

Total Duration
0min27sec

Avg Speed
36 km/h

Max Speed
18.8 km/h

No road issues detected on this ride

3.4 Route Search and Visualization Interface

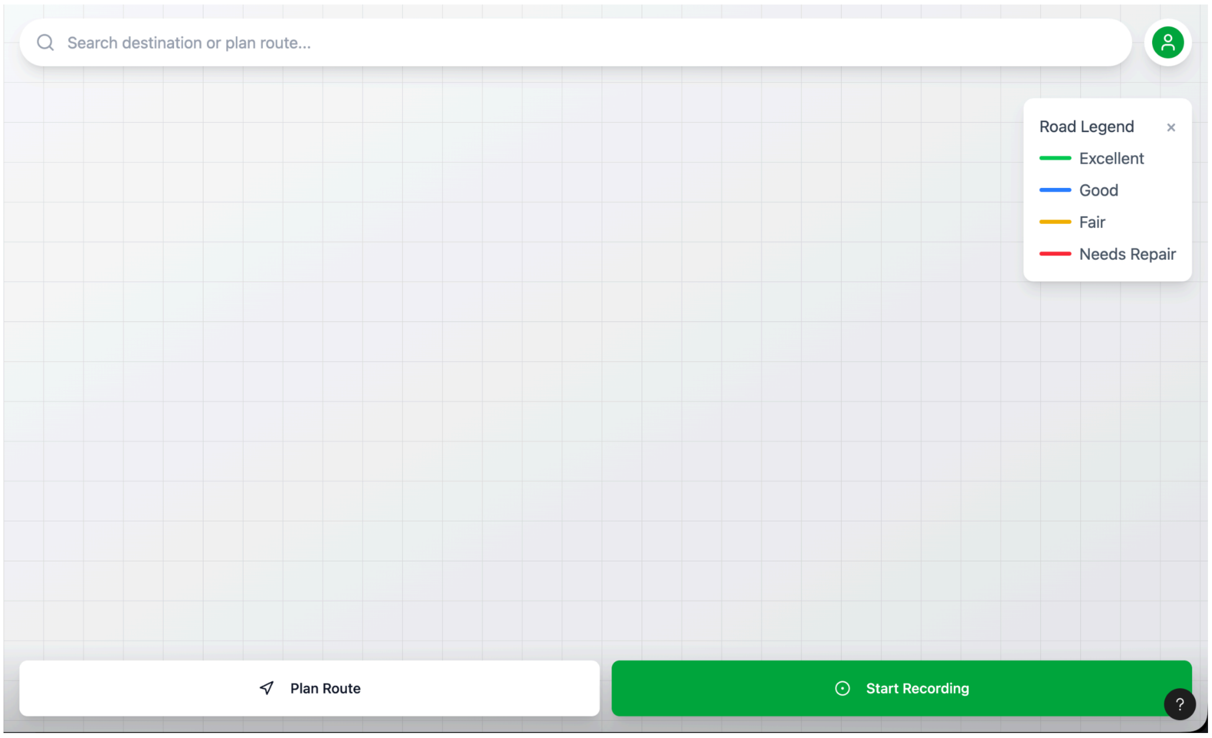
The route search interface allows both registered and guest users to:

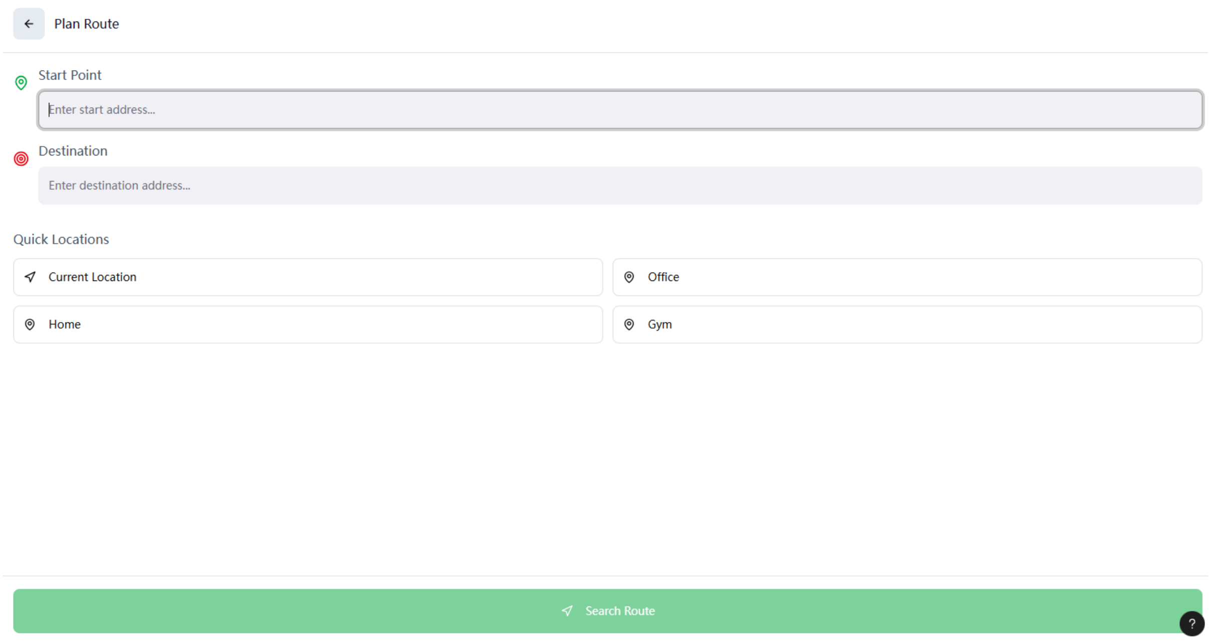
- specify an origin and a destination,
- retrieve available bike paths connecting the two locations,
- compare alternative routes.

Results are visualized on an interactive map, where users can:

- view the geometry of bike paths,
- inspect aggregated path conditions and obstacles,
- understand the relative quality of alternative routes through ranking or scoring indicators.

The interface supports the visualization of multiple paths simultaneously to facilitate comparison and informed route selection.





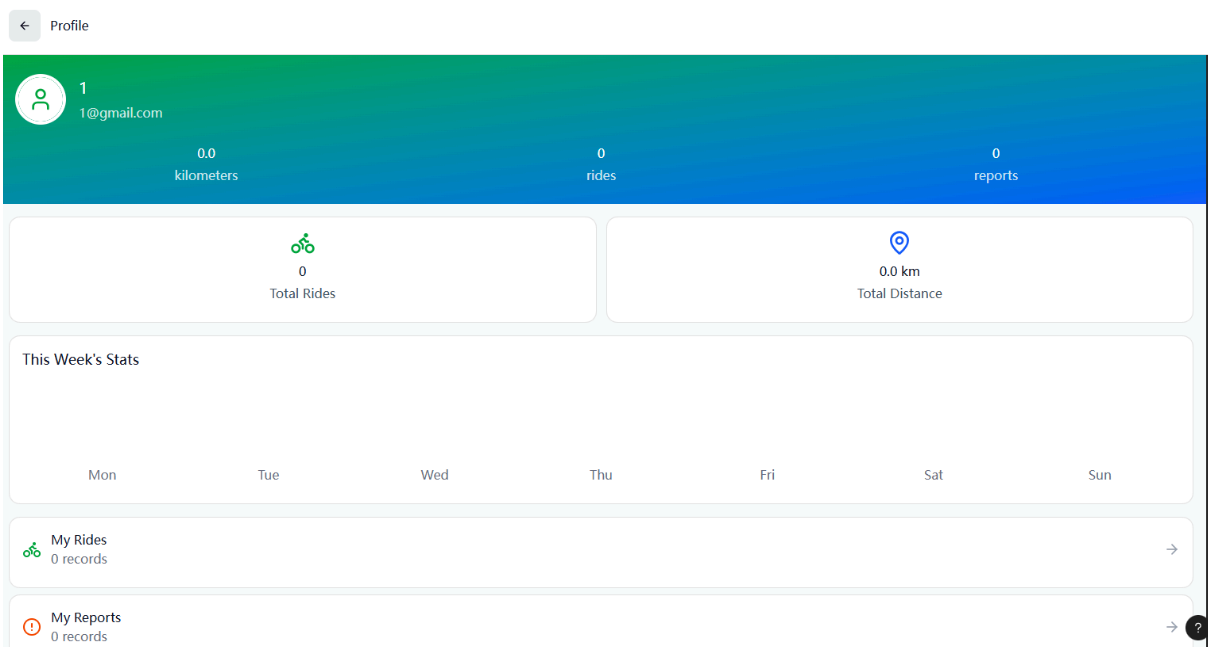
3.5 Personal Area Interface

Registered users have access to a personal area that provides an overview of their interaction with the system.
Through this interface, users can:

- view personal profile information,
- browse recorded trips and inspect detailed statistics,

- manage submitted bike path reports and their publication status.

The personal area is designed to clearly separate historical data (stored trips and reports) from ongoing activities, supporting both retrospective analysis and future planning.



Profile

1
1@gmail.com

0.0 kilometers | 0 rides | 0 reports

0 Total Rides

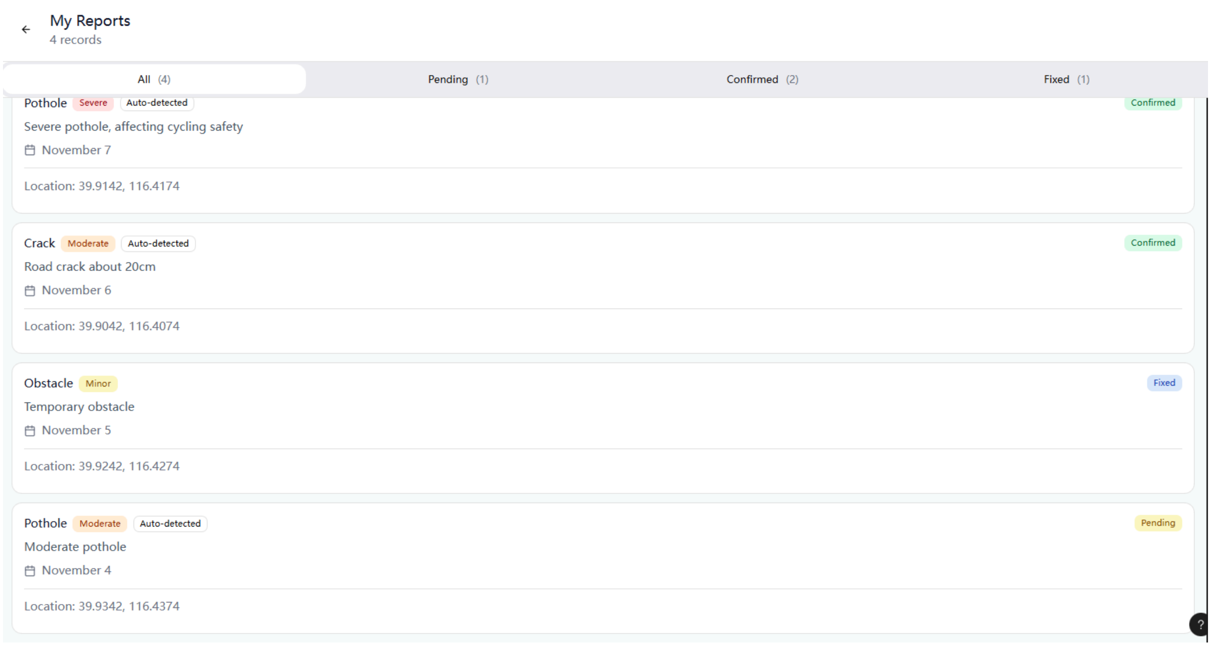
0.0 km Total Distance

This Week's Stats

Mon Tue Wed Thu Fri Sat Sun

My Rides
0 records

My Reports
0 records



My Reports
4 records

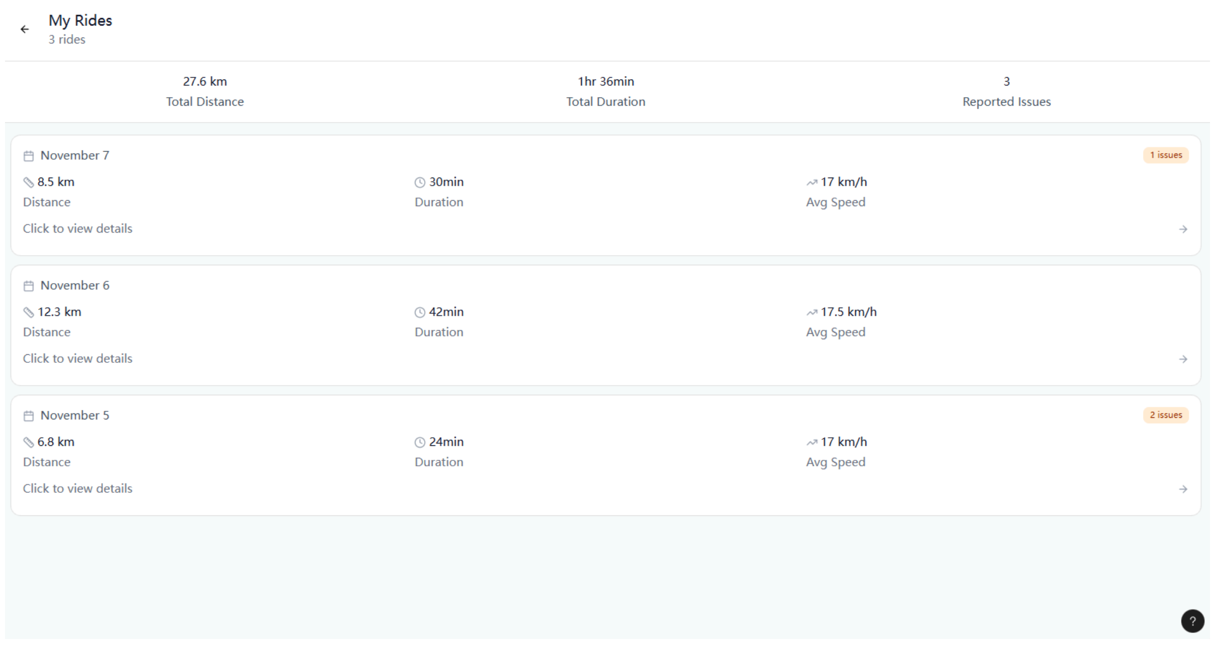
All (4) | Pending (1) | Confirmed (2) | Fixed (1)

Pothole Severe Auto-detected — Confirmed
Severe pothole, affecting cycling safety
November 7
Location: 39.9142, 116.4174

Crack Moderate Auto-detected — Confirmed
Road crack about 20cm
November 6
Location: 39.9042, 116.4074

Obstacle Minor — Fixed
Temporary obstacle
November 5
Location: 39.9242, 116.4274

Pothole Moderate Auto-detected — Pending
Moderate pothole
November 4
Location: 39.9342, 116.4374


My Rides
3 rides

27.6 km Total Distance | 1hr 36min Total Duration | 3 Reported Issues

November 7 — 1 issues
8.5 km Distance | 30min Duration | 17 km/h Avg Speed
Click to view details

November 6
12.3 km Distance | 42min Duration | 17.5 km/h Avg Speed
Click to view details

November 5 — 2 issues
6.8 km Distance | 24min Duration | 17 km/h Avg Speed
Click to view details

3.6 Design Principles

The user interface of BBP follows a set of common design principles:

- Clarity: information is presented in a clear and structured manner.
- Consistency: similar interactions and visual elements are used across different sections.
- Responsiveness: the interface adapts to different screen sizes and devices.
- Usability: interactions are kept simple, especially during cycling activities.

These principles aim to ensure that BBP remains accessible to users with different levels of technical expertise while supporting both real-time and exploratory usage scenarios.

4| Requirements Traceability

Record a Bike Trip

Requirements	[R1] The system allows registered users to start and stop the recording of a cycling trip. [R2] During trip recording, the system collects GPS data to reconstruct the traveled route. [R3] The system computes and stores statistics for each recorded trip, including total distance, duration, and average speed. [R4] The system allows registered users to enable an automated mode that collects bike path information while cycling. [R5] In automated mode, the system detects biking activity based on movement speed and collects sensor data from the user's mobile device.
Components	Web Client (Browser) API Gateway Trip Management Service User Management Service Sensor & Activity Detection Service DBMS

Automatically Collect Bike Path Condition and Weather Information

Requirements	[R6] The system acquires meteorological information related to a recorded trip when available. [R7] The system associates meteorological data, such as weather conditions and temperature, with recorded trips.
Components	Web Client (Browser) API Gateway User Management Service Trip Management Service DBMS Sensor & Activity Detection Service

View Personal Profile and Recorded Trips

Requirements	[R8] The system allows registered users to view their personal trip history and detailed statistics for each trip.
Components	Web Client (Browser) API Gateway Trip Path Report Service User Management Service DBMS

Manually Insert Bike Path Information

Requirements	[R9] The system allows registered users to manually insert information about bike paths by specifying path segments and their conditions.
Components	Web Client (Browser) API Gateway Trip Management Service Bike Path Report Service DBMS

Confirm and Publish Bike Path Reports

Requirements	[R10] The system allows registered users to review, confirm, or correct automatically or manually generated bike path reports. [R11] The system allows registered users to mark validated bike path reports as publishable. [R12] Only publishable bike path reports are used by the system for aggregation and route evaluation.
Components	Web Client (Browser) API Gateway Bike Path Report Service User Management Service DBMS

Browse Bike Paths Between an Origin and a Destination

Requirements	[R13] The system allows users to specify an origin and a destination to search for available bike paths. [R14] The system displays one or more available bike paths between the selected origin and destination. [R15] The system ranks available bike paths based on their overall condition and effectiveness in reaching the destination.
Components	Web Client (Browser) API Gateway Route Search & Ranking Service Aggregation & Merging Service DBMS Mapping & Geolocation Services (external)

Merge Bike Path Information

Requirements	[R16] The system merges publishable bike path reports provided by multiple registered users. [R17] The merging process considers the freshness of reports and the level of agreement among users.
Components	Web Client (Browser) API Gateway Route Search & Ranking Service Aggregation & Merging Service

	DBMS Mapping & Geolocation Services (external)

5| Implementation , Integration & Testing

5.1 Overview

This section describes the planned strategy for implementing, integrating, and testing the **Best Bike Paths (BBP)** system.
 The goal is to ensure a structured development process that progressively validates system functionalities and reduces integration risks.

The plan follows an incremental and modular approach, allowing early verification of core functionalities before introducing more complex interactions.

5.2 Implementation Plan

The implementation of the BBP system follows a **hybrid strategy combining bottom-up and thread-based approaches**, in order to benefit from the strengths of both methodologies.

The thread-based strategy supports the early realization of end-to-end system functionalities, enabling the delivery of intermediate results that can be evaluated by stakeholders. This approach is particularly useful for validating system behavior and ensuring alignment with user expectations at early stages of development.

At the same time, the bottom-up strategy facilitates **incremental integration** by progressively assembling subsystems from smaller, well-tested components. This allows defects to be identified and isolated more effectively, as each intermediate subsystem can be tested before additional modules are integrated.

Within the thread-based approach, system features are decomposed into the specific portions of components—referred to as subcomponents—that contribute to their realization, even when such subdivisions do not strictly correspond to architectural components. Since the implementation of a single feature typically requires cooperation among multiple subcomponents, defining a clear implementation order becomes essential. The bottom-up strategy supports this need by ensuring that lower-level functionalities are implemented and validated before higher-level integrations occur.

This combined implementation strategy enables parallel development across multiple teams working on different system features. To support effective coordination and avoid redundant development efforts, shared components and responsibilities are identified early and reused consistently throughout the implementation process.

5.2.1 Features identification

The features to be implemented are derived from the requirements defined in the RASD. Some of these features require the introduction of dedicated components, while others are realized through the extension or coordination of existing ones.

The following list summarizes the main features of the BBP system that guide the implementation process.

[F1] Record a Bike Trip
This feature allows registered users to record their cycling activities. The system supports starting and stopping trip recording, collecting GPS data during the ride, and computing basic trip statistics such as distance, duration, and average speed. Recorded trips are stored and associated with the corresponding user, forming a personal trip history.

[F2] View Personal Profile and Recorded Trips
This feature enables registered users to access their personal area and inspect previously recorded trips. Users can view trip details, including reconstructed routes and computed statistics. The feature relies on existing trip management components and does not require additional data acquisition.

[F3] Manually Insert Bike Path Information
This feature allows registered users to manually provide information about bike paths. Users can define one or more bike path segments and specify their conditions, optionally including notes about obstacles. The inserted information is initially stored as draft reports and remains private until validated and published.

[F4] Automatically Collect Bike Path Condition and Weather Information
This feature supports the automated acquisition of bike path information during cycling activities. The system detects biking activity based on user speed, reconstructs followed paths using GPS data, and identifies potential obstacles through sensor data. When available, meteorological information is also collected and associated with the trip. All automatically collected data is stored as draft information pending user validation.

[F5] Confirm and Publish Bike Path Reports
This feature enables registered users to review, confirm, or correct both manually and automatically generated bike path reports. Only reports that successfully pass this validation step can be marked as publishable. This mechanism ensures data reliability before information is shared with the community.

[F6] Merge Bike Path Information
This feature allows the system to aggregate publishable bike path reports provided by multiple users. The merging process considers the freshness of reports and the level of agreement among users to produce a consolidated representation of bike path conditions. This feature is entirely system-driven and supports consistency across shared data.

[F7] Browse Bike Paths Between an Origin and a Destination
This feature enables users, both registered and guests, to search for bike paths between a selected origin and destination. The system retrieves available routes, computes aggregated information and scores, and presents ranked alternatives through map-based visualization. This functionality supports route comparison and informed decision-making.

5.3 Component Integration and Testing

The integration and testing of the BBP system follow a **bottom-up strategy**, where individual components are first tested and then progressively integrated into larger subsystems. This approach allows early validation of core services and simplifies fault isolation, as errors can be detected before higher-level components are introduced.

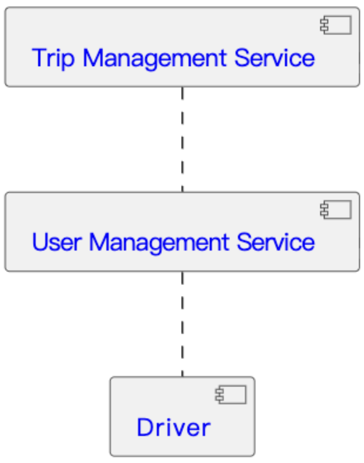
Integration testing starts from the fundamental services that provide basic system functionalities, such as user and trip management. Additional services are incrementally integrated on top of the previously validated ones. At each step, already integrated components are re-tested to ensure that new integrations do not introduce regressions.

For each feature, integration tests are triggered by a **Driver**, which simulates requests to the system. Databases and external services are assumed to be available and correctly functioning and are therefore not explicitly represented in the diagrams.

Each diagram represents the set of components involved in the integration test of a specific feature, highlighting in **blue** the components that are newly introduced at that stage.

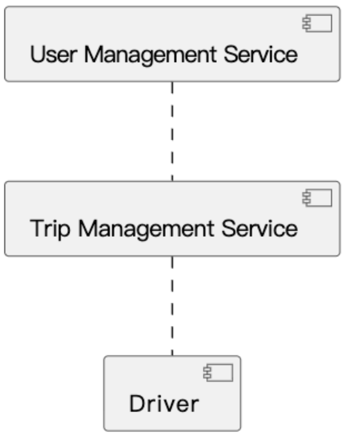
[F1] Record a Bike Trip

Integration testing starts from the core services responsible for managing trips and users. At this stage, the **Trip Management Service** is introduced and integrated with the **User Management Service**.



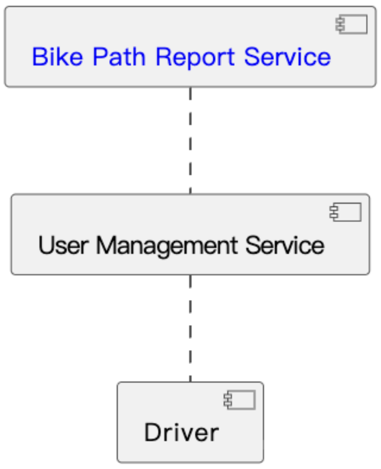
[F2] View Personal Profile and Recorded Trips

After validating trip recording, the visualization of personal profiles and recorded trips can be tested.



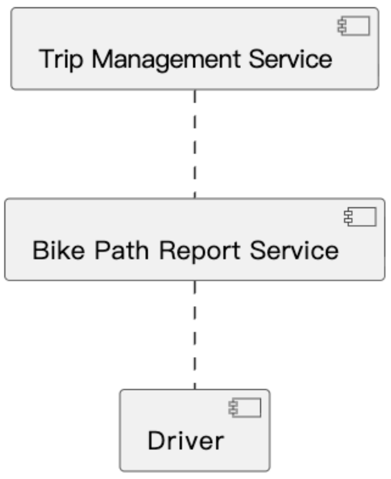
[F3] Manually Insert Bike Path Information

This feature introduces the manual insertion of bike path information by registered users.
Bike path reports are initially stored as draft information.



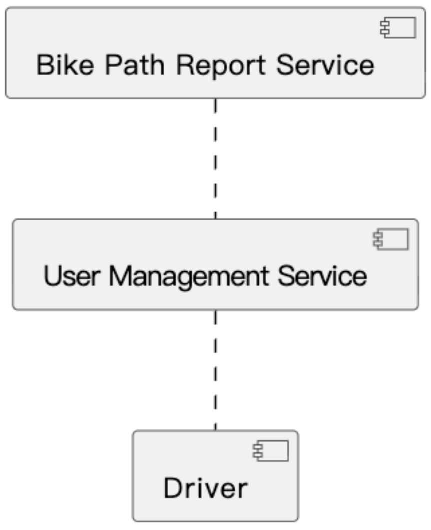
[F4] Automatically Collect Bike Path Condition and Weather Information

This feature extends the system by enabling the automatic collection of bike path conditions during trips.
It relies on data analysis performed on GPS and sensor data.



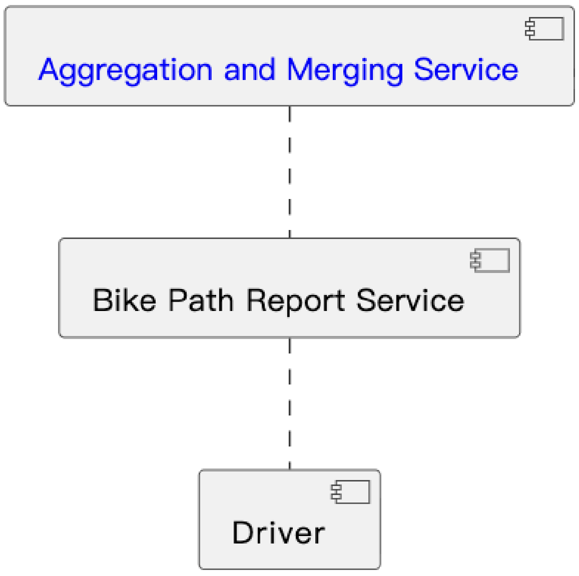
[F5] Confirm and Publish Bike Path Reports

Once draft reports are available, users can review and validate them before publication. This feature does not introduce new components but tests additional interactions among existing ones.



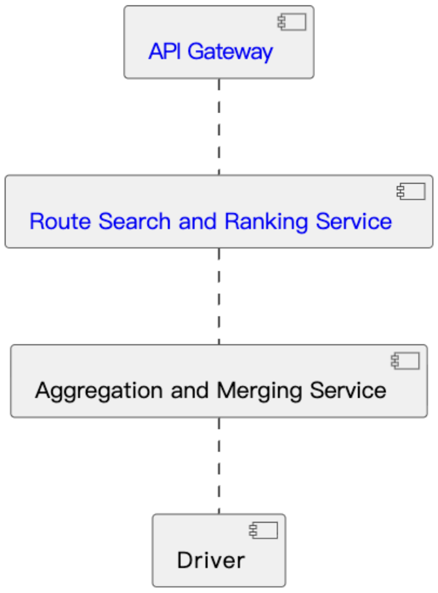
[F6] Merge Bike Path Information

This feature allows the system to aggregate bike path reports provided by multiple users. The merging process is system-driven and focuses on data consistency.



[F7] Browse Bike Paths Between an Origin and a Destination

The final feature enables users to browse bike paths between an origin and a destination. This step completes the end-to-end integration of the system.


API Gateway
Route Search and Ranking Service
Aggregation and Merging Service
Driver

Final Integration Step

After all features have been individually integrated and tested, the system is tested as a whole by connecting the client applications to the API layer, ensuring correct cooperation among all components.

5.4 System Testing

After completing the integration testing activities described in Section 5.3, system testing is performed to validate the BBP system as a whole.
 The objective of this phase is to ensure that all components interact correctly through the system interfaces and that the functional and non-functional requirements defined in the RASD are satisfied.

System testing is based on end-to-end execution of system functionalities, simulating realistic user interactions. As during integration testing, components are progressively integrated and already available features are re-tested to detect possible regressions introduced by newly added components.

Functional testing verifies the correct behavior of all system features, ensuring that the provided functionalities produce consistent and correct results.
 In addition, non-functional testing is conducted at system level, including performance, load, and stress testing, in order to evaluate the system's efficiency, scalability, and robustness under different operating conditions.

Once all system tests have been successfully completed and identified issues have been resolved, the BBP system can be considered validated and ready for deployment.

5.5 Additional Specifications on Testing

During the entire implementation period of the BBP system, user acceptance testing should be carried out to ensure that the application satisfies the needs and expectations of stakeholders and end-users. Developers should adopt both alpha and beta testing phases. During the alpha testing phase, the system is tested internally by the development team in order to identify and resolve critical issues at an early stage. This is followed by a beta testing phase, during which a wider group of users interacts with the system in realistic usage scenarios.

Security testing must be considered a priority, as the system manages sensitive user data and relies on multiple interacting services. Identifying potential vulnerabilities as early as possible is therefore essential to ensure system reliability. For future releases, a phased deployment strategy can be adopted, initially releasing new features to a limited subset of users before extending them to the entire user base. This approach allows developers to monitor system behavior and quickly address possible issues before they affect all users. All testing activities should follow established industry best practices and be properly documented in order to track detected issues and their corresponding resolutions.

6| Effort Spent

The following table summarizes the approximate number of hours each group member devoted to the different sections of the document. The reported distribution is indicative, as the preparation of each section required collaboration and joint discussion among all group members.

	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5
Run Jie Simone Dai	2	16	1	5	7
Jiaxin Yang	4	2	6	8	13
Jiayi Zheng	3	18	1	4	5

7| References

- Diagrams are made with draw.io.

- UI mockups are made with Figma.