POLITECNICO MILANO 1863

COMPUTER SCIENCE AND ENGINEERING
SOFTWARE ENGINEERING II
2025 - 2026

DD

Design Document

Authors:	Cristhian Mejia and Sravan Yerranagu
Version:	1.0
Date:	January-2025
Download page:	`https://github.com/ReGaL24/MejiaYerranagu`
Copyright:	Copyright © 2025, Cristhian Mejia, Sravan Yerranagu - All rights reserved

Contents

1 Introduction

1.1 Purpose

Urban mobility is increasingly moving toward sustainable transportation solutions, with cycling playing a central role in reducing environmental impact. Despite this trend, cyclists often lack reliable information about the quality, safety and suitability of bike routes, as such data is typically scattered, outdated, or missing. At the same time, cyclists continuously produce valuable data through their daily trips, which can be used to improve up-to-date knowledge about cycling infrastructure.

The purpose of Best Bike Paths (BBP) is to provide a software system that supports cyclists to record and analyze their personal trips, and enabling the creation, management, and exploration of an inventory of bike paths enriched with community-provided information. The system supports both manual and sensor-based acquisition of path data, ensuring that automatically collected information is reviewed and confirmed by users before being shared. Additionally, BBP allows any user to identify and visualize suitable bike paths between a given origin and destination, ranking alternatives based on route effectiveness and path conditions.

1.1.1 Goals

G1: Allow registered users to log personal rides and view summary stats (distance, duration, average speed, and key performance metrics).

G2: Let registered users manually create and maintain bike path data by defining route segments and tagging conditions and obstacles.

G3: Let registered users automatically record bike path data during rides via GPS-based path reconstruction and sensor-based anomaly detection.

G4: Require user review before publishing automatically collected path data, and let contributors choose whether their submissions are shared with the community.

G5: Let any user view and compare bike paths between an origin and destination on a map, ranking options by a score combining route effectiveness and path conditions.

1.2 Scope

Best Bike Paths (BBP) is a mobile-centered software system that supports cyclists in recording personal biking trips and in managing an inventory of bike paths enriched with user-contributed information. The system enables registered users to record trips and access related statistics, optionally including meteorological data retrieved from external services.

Registered users can also publish bike path information through two distinct modes: manual mode, where users explicitly define the path segments and associate route status and obstacles, and automated mode, where the system reconstructs the followed path through GPS data and detects potential obstacles through signals acquired from the device motion sensors. Since automatically acquired information may be inaccurate, the system requires user review and confirmation prior to publication, allowing contributors to control the visibility of their data. Both registered and unregistered users can search in the system by specifying an origin and a destination and view one or more possible bike paths on a map, ranked according to a path score reflecting both path conditions and route effectiveness.

BBP includes the functionalities for trip storage and statistics computation, acquisition and management of bike path information, user confirmation, publication control, map-based visualization and ranking of paths. BBP relies on external services and mobile device sensors as data sources, while maintaining clear boundaries with respect to their control and availability.

1.2.1 World Phenomena (WP)

The system operates within a world where:

WP1: Cyclists physically traverse bike paths in urban and suburban environments.

WP2: Mobile devices generate raw positioning data (e.g., GPS coordinates) with inherent accuracy and coverage limitations.

WP3: Mobile devices generate sensor data (e.g., accelerometer and gyroscope signals) reflecting physical movements during biking.

WP4: Road infrastructure elements (e.g., bike lanes, potholes, obstacles) exist as physical entities and may change over time.

WP5: Weather conditions vary over time and location, and external meteorological services maintain authoritative datasets.

WP6: Different cyclists may traverse the same paths at different times and under different conditions.

WP7: Users form subjective assessments of bike path quality based on personal experience.

WP8: Network connectivity may be intermittent during outdoor biking activities.

1.2.2 Shared Phenomena (SP)

SP1: The system acquires location data transmitted by a user's mobile device during biking activities.

SP2: The system infers biking activity based on observed movement characteristics (e.g., speed).

SP3: The system computes and stores statistics related to recorded biking trips.

SP4: The system retrieves meteorological information from an external weather service associated with a recorded trip.

SP5: The system presents recorded trip data enriched with contextual information to registered users.

SP6: Registered users manually insert information about bike paths, including involved segments, qualitative status, and obstacles.

SP7: The system acquires potential bike path information through automated sensing during biking activities.

SP8: Registered users review, confirm, or correct automatically acquired bike path information.

SP9: Registered users decide whether their contributed bike path information is made publishable.

SP10: Users specify an origin and a destination to query bike paths.

SP11: The system visualizes one or more candidate bike paths on a map, ordered according to a computed path score.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Bike path**: A route suitable for cycling, either characterized by the presence of a dedicated bike track or by low vehicular traffic and speed limits compatible with average cycling speed.
- **Path status**: A qualitative assessment of the condition of a bike path, expressed using predefined categories such as Optimal, Medium, Sufficient, or Requires Maintenance.
- **Path score**: A numeric value used to rank alternative bike paths between a given origin and destination, reflecting both path condition and route effectiveness.
- **Publishable information**: User-contributed bike path information that has been explicitly marked by its owner as available for consultation by other users.
- **Trip**: A complete cycling journey recorded by the system, defined by a start time, an end time, and associated measured and computed data.
- **Trip statistics**: A set of computed metrics derived from a recorded trip, including distance, duration, average speed, maximum speed, and elevation gain.
- **Obstacle**: A physical condition or element along a bike path that may negatively affect cycling safety or comfort, such as potholes or surface irregularities.
- **Weather enrichment**: The association of meteorological information (e.g., temperature, wind speed, weather conditions) retrieved from an external service with a recorded trip.
- **Manual mode**: A mode of interaction in which a user explicitly inserts bike path information without relying on automated sensing or inference.
- **Automated mode**: A mode of interaction in which the system acquires bike path information during a biking activity by analyzing data collected from the user's mobile device, such as GPS and motion sensors.
- **Registered user**: A user who has created an account in the system and is authorized to record trips and contribute bike path information.
- **Unregistered user**: A user who accesses the system without authentication and is limited to querying and visualizing bike paths.

1.3.2 Acronyms

- **BBP**: Best Bike Paths
- **RASD**: Requirements Analysis and Specification Document
- **OD**: Origin-Destination

1.3.3 Abbreviations

- **G**: Goal
- **WP**: World Phenomenon
- **SP**: Shared Phenomenon
- **GPS**: Global Positioning System
- **API**: Application Programming Interface
- **HTTPS**: Secure HTTP protocol for encrypted communications
- **REST**: REpresentational State Transfer

- **CRUD**: Create, Read, Update, Delete
- **TTL**: Time To Live
- **UI**: User Interface
- **UX**: User Experience
- **DBMS**: DataBase Management System
- **PostGIS**: A spatial database extender for PostgreSQL - Post(greSQL) + GIS
- **GIS**: Geographic Information System
- **JSON**: JavaScript Object Notation
- **GeoJSON**: Geographical extension of JSON format
- **JWT**: JSON Web Token
- **SSL**: Secure Sockets Layer
- **TLS**: Transport Layer Security
- **TTL**: Time To Live
- **API Gateway**: Application Programming Interface Gateway

1.4 Reference Documents

The assignment for this document and all the information included refer to the following documentation:

- The specification for the 2025/26 Requirement Engineering and Design Project for the Software Engineering II course.
- The slides on the WeBeep page of the Software Engineering II course.

1.5 Document Structure

- **Section 1 (Introduction)**: Problem context, scope boundaries, terminology, document metadata.
- **Section 2 (Architectural Design)**: High-level system architecture decisions, deployment, runtime and component views and interfaces.
- **Section 3 (User Interface Design)**: Overview of the user interface structure and interaction patterns for users.
- **Section 4 (Requirements Traceability)**: Mapping between functional requirements and the architectural components responsible for their realization.
- **Section 5 (Implementation, Integration and Test Plan)**: Planned development order of system components, integration strategy, and testing approach.
- **Section 6 (Effort Spent)**: Time allocation and effort tracking by team member.
- **Section 7 (References)**: Source materials and documentation.

2 Architectural Design

2.1 Overview

This section aims to provide a high-level description of the architecture of the BestBikePaths (BBP) system and outlines the main architectural layers, the core components and their interactions. This layered organization facilitates a clear allocation of responsibilities and interactions across the system components. This section also validates the modeling decisions made to satisfy both the functional and non-functional requirements of the system.

2.1.1 System View

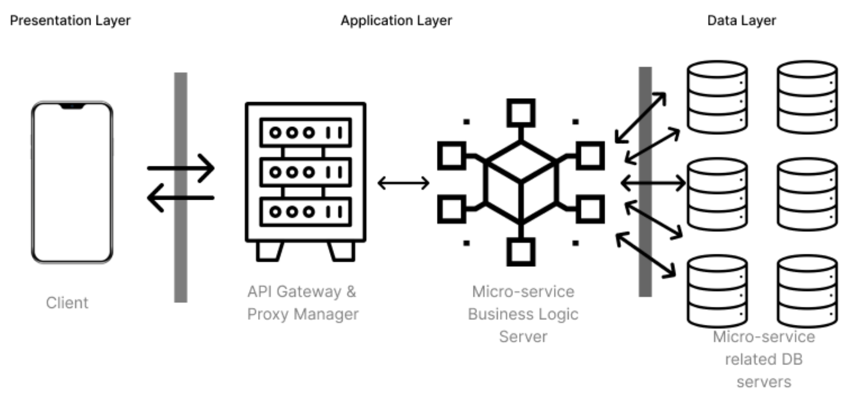
The chosen architectural solution for the BBP system is a 4-tier with microservice architecture. The architecture consists of a clearly separated presentation layer, two application layers - a server side layer and a second microservice layer, and a data layer. This structure allows for a modular development with easier maintenance, and scalability.

The presentation layer resides on the client side, implemented as a native mobile application for iOS and Android platforms, that will be responsible for all system functionality such as user interaction, trip recording, and visualization of bike paths and trip statistics. This mobile first approach prioritizes user experience and accessibility, where users can interact with the smartphone while cycling.

The server side application layer has been divided into 2 parts where the first part is the API Gateway and proxy management, which serves as the main entry point for all client requests. It handles authentication, routing, and rate limiting, and more importantly insulates the backend services from direct exposure to the internet, applying consistent security policies and request routing. The second part of the application layer consists of multiple microservices which make up the core functioning of the BBP system, each responsible for a specific functionality such as trip recording services, path services, user authentication, weather integration, and notifications services. This microservice architecture allows for independent development, deployment, and scaling of each service on a rapid scale.

The data layer consists of a multiple relational database management system (PostgreSQL), a single DB per microservice for persistent storage of user data, trips, paths, and other relevant information.

The flow of information between the layers is as follows: the client-side mobile application collects GPS data, user inputs and path information and then sends the requests to the API Gateway which enforces security and rate-limiting policies, which then routes them to the appropriate backend microservice based on the request type. Each microservice then processes the request calling on any external systems if required, interacts with the database as needed, and returns the formatted response back through the API Gateway to the client application. These microservices indeed do communicate with each other for processing with the help of message broker which supports event-driven interaction and loose coupling.


Presentation Layer
Application Layer
Data Layer
Client
API Gateway & Proxy Manager
Micro-service Business Logic Server
Micro-service related DB servers

2.1.2 Detailed View

Presentation Tier

The user interacts with the BBP system exclusively through a mobile application (iOS and Android) installed on their smartphone. The mobile application is responsible for collecting user inputs through a touch friendly interface, optimized for on the go use by the cyclists using it. The application acquires the GPS data from the device location services during bike trips, allowing users to enter bike paths, and also displays trip statistics and path information. The app communicates with the backend services via HTTPS REST request through the API Gateway to transmit collected data, query for bike paths, and retrieve user information and also displays the responses from the backend in a user-friendly manner: real-time trip metrics, path maps, notifications, etc. The mobile application also facilitates offline capabilities by locally buffering GPS data during trips when network connectivity is unavailable, and then synchronizing the data with the backend once connectivity is restored ensuring data integrity. This feature is critical for those who may cycle through areas with poor network coverage.

Application Tier - Frontend

This application tier serves as the main entry point for all client requests and acts as a buffer between the mobile client and the core backend services. This mainly handles authentication, routing, and rate limiting, and more importantly insulates the backend services from direct exposure to the internet, applying consistent security policies and request routing. The Front End of the application layer mainly consists of Proxy manager, API Gateway and Proxy for External API calls.

Proxy Manager manages incoming requests from the mobile application, enforcing security policies such as handling SSL/TLS certificate management and its renewal, decryption of incoming HTTPS requests often acting as a first security layer. It ensures that only authorized requests reach the backend services.

API Gateway serves as the main entry point for all client requests. It handles authentication and authorization by extracting the authentication token identifying the user identity and role - `RegisteredUser` or `UnregisteredUser`. It routes requests to the appropriate backend microservice based on the request type, ensuring that each service only receives relevant requests. Rate limiting is implemented to prevent abuse and ensure the protection of the backend services from denial-of-service attacks. Further Load balancing is also implemented to distribute incoming requests evenly across multiple instances of backend services, ensuring optimal resource utilization and responsiveness. Caching of frequently requested data is also implemented to reduce database load, latency and improve responsive performance.

Proxy for External API Calls manages outbound requests to third-party services such as OpenWeather API for weather data, OpenStreetMap for map data and OAuth providers for user authentication. It handles the communication protocols incase any external API are offline by implementing timeout policies and retry logic with backoff for failure to ensure that these external service unavailable doesn't degrade BBP system functionality.

Application Tier - Backend

The backend application tier consists of multiple microservices, each responsible for a specific functionality within the BBP system. The core microservices include:

- User Authentication Service: Manages user registration, login, OAuth federation, and token issuance for secure access to the system.
- Trip Recording Service: Responsible for acquiring GPS data during bike trips, computing trip statistics (distance, duration, speed, elevation gain), and storing trip data.
- Path Management Service: Manages the creation, and publishing of bike paths by `RegisteredUser`, including obstacle information.
- Path Scoring Service: Handles the computation of a path score value based on path conditions and obstacles and path ranking so it can order and effectively prioritize paths for visualization.
- Geocoding Integration Service: This component communicates directly with the external geocoding service, described in the External Services section below. It exposes a clean internal interface to uniformly resolve geographic locations and manage failures and eventual service unavailability.
- Weather Integration Service: Integrates with the OpenWeather API to enrich trip data with weather information at the time and location of the trip.
- Notification Service: Sends in-app and push notifications to users for important events such as trip completion, path publication, and weather alerts.

These microservices are independently deployable and scalable, allowing for rapid development and updates without affecting the entire system. They communicate with each other using RESTful APIs for event-driven interactions through message boards like RabbitMQ for events such as *trip.published*, *path.published*, *weather.enriched*, *weather.failed*.

Data Tier

The data layer is essential for overseeing the system's information all while offering efficient access to application elements. The database provides a primary persistent storage solution for user data, trips, paths, and other relevant information such as weather snapshot and geocoding cache. The database server is configured with multiple database setup with replication to ensure high availability and fault tolerance. Each microservice interacts with a dedicated database through well-defined schema for its entities, enabling data isolation.

2.2 Component View

The following component view organizes the BBP system into presentation (mobile client), application (frontend and backend), data and external components layers to clearly separate responsibilities. The core functionality is relayed to the application layer while the user interaction is described in the presentation layer. This also allows us to declare third party services as external dependencies with their corresponding considerations.

Presentation Layer

These components reside entirely on the user's mobile device.

- Mobile User Interface: The component that interacts directly with the users. It is responsible for rendering screens, maps and user interactions. It adapts functionality and navigation features depending on the type of user (registered/unregistered). It communicates with the backend through a direct connection to the API Gateway.

- Device Sensors Adapter: It interacts with platform-specific interfaces to access the device GPS, accelerometer and gyroscope APIs, both for Android and iOS systems. This component is also in charge of providing sensor data to the Path Management Service.
- Cache Server Adapter: This component manages local data buffering when the device is offline. It stores GPS data during trips and synchronizes it with the Trip Recording Service once connectivity is restored.

Application Layer

These components run on backend infrastructure and expose the system's core functionality.

- API Gateway: This component acts as the only entry point for all client requests by exposing an API to the mobile client. It routes each request to the appropriate application components and enforces the necessary security mechanisms (authentication, rate limits).
- User Authentication Service: This component oversees managing user accounts, credentials and sessions. It validates authentication tokens and controls access to restricted functionalities, specifically when regarding the distinction between registered and unregistered users. Thus, it supports both authenticated and guest sessions.
- Trip Recording Service: This component is responsible for managing a trip state during recording (play, pause, stop). It also handles the storage and retrieval of previously recorded trips. It can compute and persist relevant trip statistics to deliver them to the User Interface. It manages trip lifecycle states like recorded, processed, enriched, and communicates with Weather Integration Service to retrieve and store additional trip information. Since this service displays trips for the user, it retrieves coordinates data from the Geocoding Integration Service.
- Path Management Service: This component manages bike path data contributed by users. It handles both manual path insertion and automatic path data collection. It is also in charge of enforcing publication and visibility rules and managing users path reviews for the automatic case. This component also enables path discovery functionality with the support of the Geocoding Integration Service and Path Scoring Service, which can be accessed by registered and unregistered users.
- Path Scoring Service: This component handles the computation of a path score value based on path conditions and obstacles. It also handles path ranking so it can order and effectively prioritize paths for further visualization. It encapsulates scoring logic, so it is kept independent from path management and storage.
- Geocoding Integration Service: This component communicates directly with the external geocoding service, described in the External Services section below. It exposes a clean internal interface to uniformly resolve geographic locations and manage failures and eventual service unavailability.
- Weather Integration Service: This component communicates directly with the external weather service, described in the External Services section below. It communicates with the Trip Management Service to complement the stored trip information. It isolates the dependency with the third-party weather service from the core application logic.
- Notification Service: This component manages system notifications regarding trip completion, errors or review requests, to enrich the user's experience with relevant information. This component handles exclusively in-app and push notifications.

Data Layer

These components are responsible for providing data storage and access.

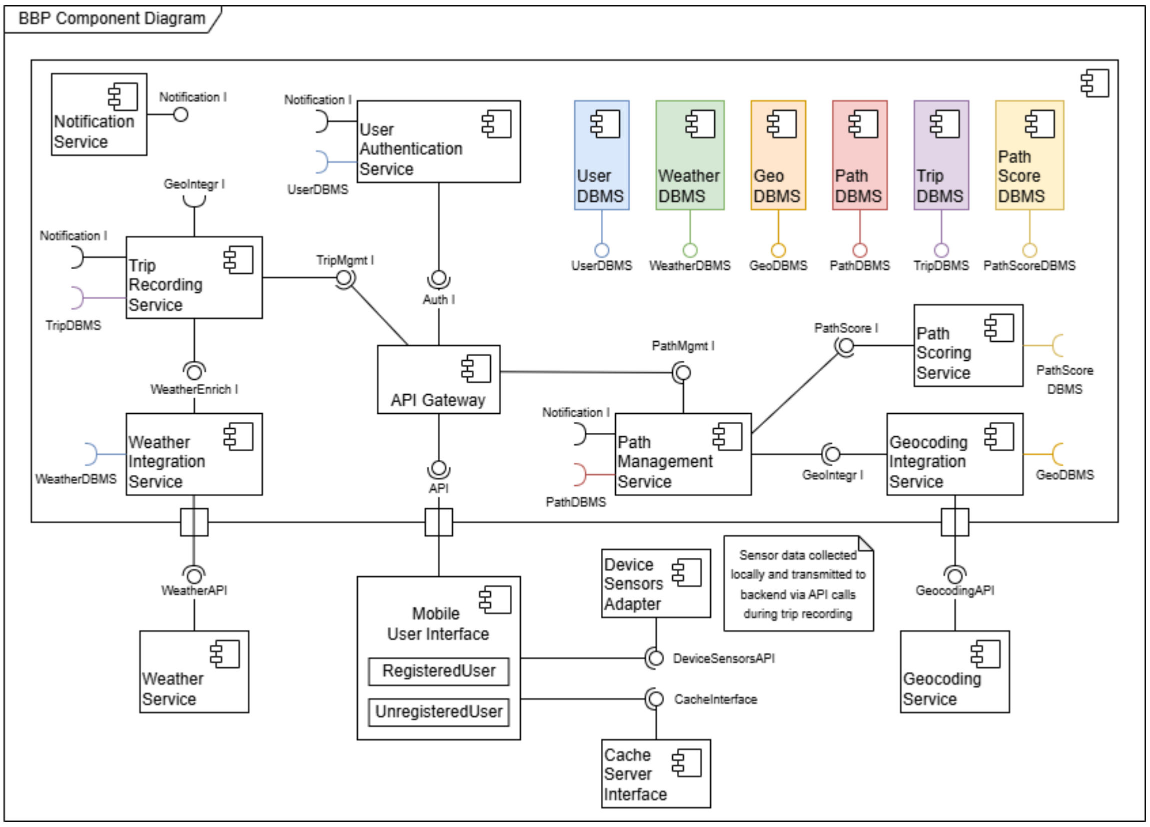
- Database Management System (DBMS): This component oversees the storage and persistence of users, trips, bike paths, obstacles and any relevant metadata. It ensures data integrity and consistency and supports querying and indexing for more efficient path discovery. Each service encapsulates its own persistence logic and interacts directly with the DBMS.

External Services

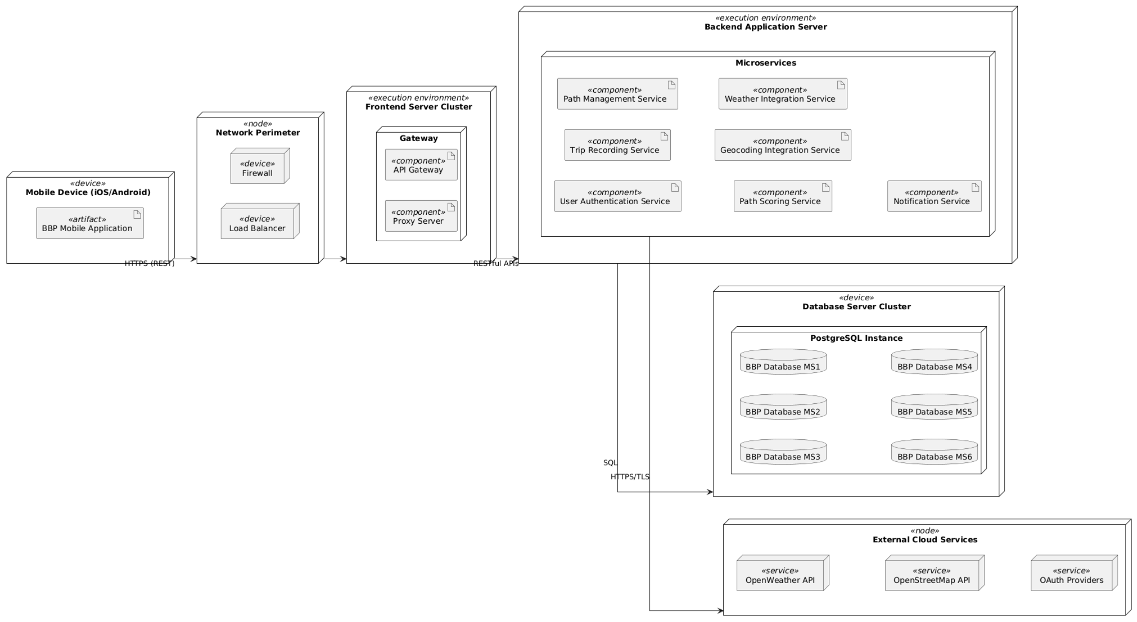
These external services are accessed through dedicated integration components to separate the core application logic from third-party dependencies. We assume that they are generally available and reliable, and that they support fault tolerance and graceful degradation policies. The application does not depend entirely on their availability and thus have fallback mechanisms when data cannot be retrieved.

- Weather Service: This external component retrieves meteorological data given specific location and time data. This service is exposed as an API that can be consumed by the Weather Integration Service on demand.

- Geocoding Service: This external component resolves addresses to coordinates and computes routes between two locations. This service is exposed as an API that is consumed by the Geocoding Integration Service. It can translate user-provided locations into geographic coordinates that can be later presented visually in a map.



2.3 Deployment View



This section describes the deployment view of the BBP system, outlining the hardware and software components, their configurations, and how their geographical distribution is in a production environment.

Mobile Device

A mobile device (smartphone) is used by cyclists to run the native BBP mobile application on iOS or Android platforms. The device's GPS capabilities are used to collect location data during bike trips. The mobile application then communicates with the backend services over HTTPS. Additionally, the application can also store GPS data when offline using local data buffering which synchronizes with the backend when connectivity is restored.

Firewall

A firewall is positioned at the network perimeter for monitoring of all network traffic from the mobile clients and also to protect the backend services from unauthorized access and potential threats. It filters incoming and outgoing traffic based on predefined security rules, ensuring that only legitimate requests reach the API Gateway. Should any anomalous traffic be detected, the firewall safeguards the backend infrastructure by blocking these unauthorized connections.

It is to be noted that the firewall is positioned before the Load Balancer to ensure that all incoming traffic is inspected and is deemed safe before being distributed to the backend services.

Load Balancer

The Load Balancer distributes the incoming requests from the mobile application evenly across multiple instances of the backend microservices. This ensures optimal resource utilization, responsiveness, and high availability. The load balancer periodically checks the health of Gateway instances automatically routing traffic only to healthy instances, preventing downtime in case of service failures.

API Gateway and Proxy Server

The API Gateway and Proxy Server layer handles all incoming requests from the mobile application, enforcing security policies such as authentication, rate limiting, and routing to the appropriate backend microservice. This layer deals with decryption of incoming HTTPS requests, SSL/TLS certificate management, and outbound requests to third-party services such as OpenWeather API, OpenStreetMap API and OAuth providers. Caching of frequently requested data is also implemented at this layer to reduce database load and improve response times where a request is delt without it ever being passed on to the Application Server.

Microservice Application Servers

The Microservice Application Servers host the core backend services of the BBP system. Each microservice is deployed on its own server instance, allowing for independent scaling and maintenance. The microservices communicate with each other using RESTful APIs and also interact with the database for data storage and retrieval. The microservices include:

- Trip Recording Service - for handling of GPS data, Trip statistics computation, and local buffering.
- Path Management Service - for handling of path data insertion by `RegisteredUser`, as well as enforcing publication states, visibility rules, and storage of path data in the database. It also enables path discovery functionality.
- Path Scoring Service - for computation of path scores and ranking based on path conditions and obstacles.
- Geocoding Integration Service - for integration with the external geocoding service API to resolve geographic locations.
- Weather Integration Service - for integration with weather APIs for trip weather data upon notification of trip completion event from the message brokers.
- User Authentication Service - for managing user login, registration, and session management.
- Notification Service - for sending notifications to users about events or updates.

Database Server

The Database Server in the BBP system is configured with a multiple seperate database setup along with replication to ensure high availability and failover where each microservice interacts with its own database through well-defined schema for its entities. Further PostGIS extension is used to handle geographical data.

External Services

The BBP system integrates with several external services like OpenWeather API, OpenStreetMap API at the microservice level to enhance the systems functionality.

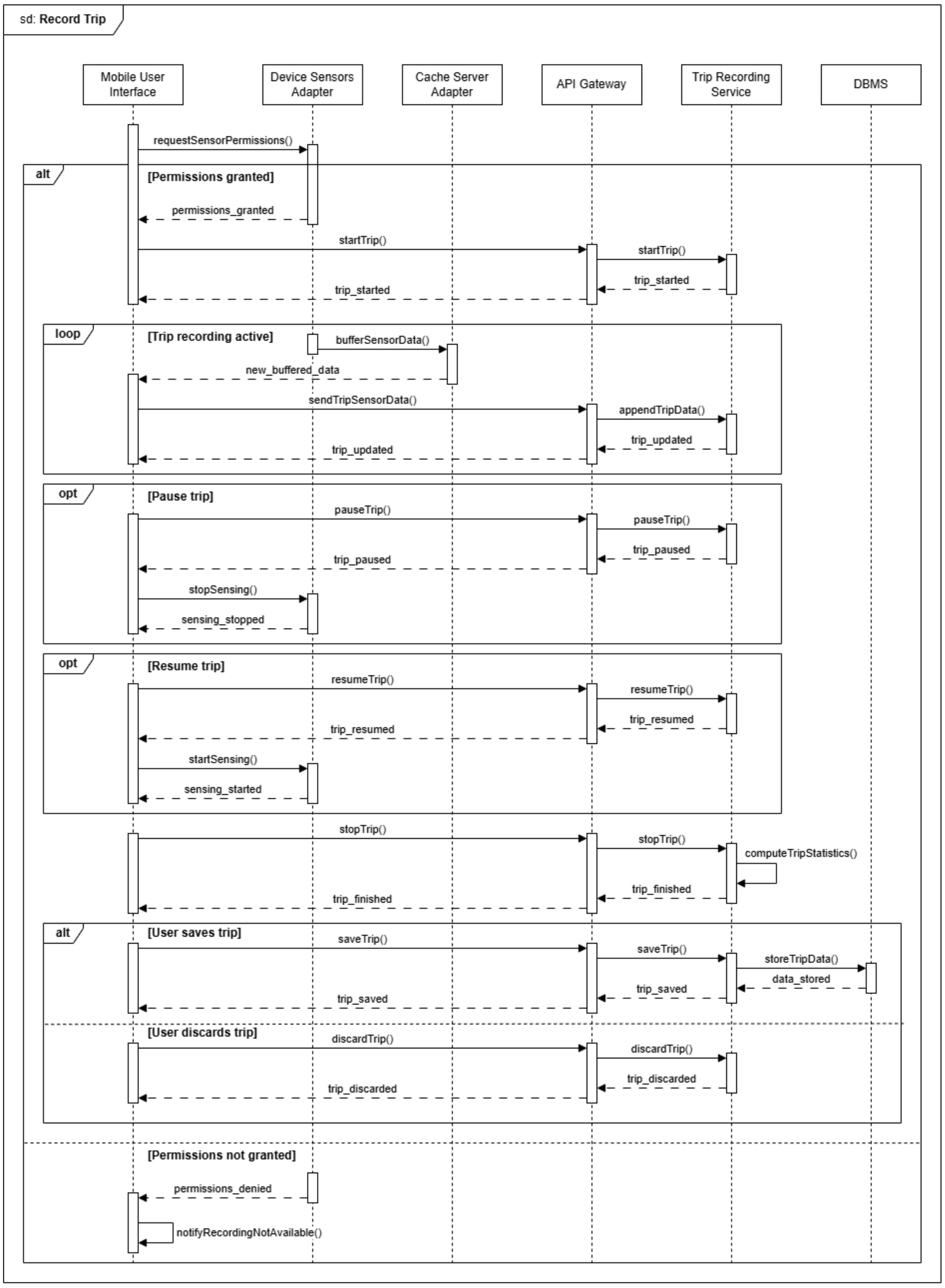
Geographical Distribution and Deployment

The BBP system is deployed using dockerized containers where each microservice, API Gateway and Proxy Manager Server runs in their own container. Further, the database is deployed in a multi-region setup where the primary region - Italy houses all the microservices, databases which handle 100% of the traffic under normal conditions with readily available backups of databases along with standby instances of microservices for failover in situations where the primary region becomes unavailable to ensure data redundancy and disaster recovery

2.4 Runtime View

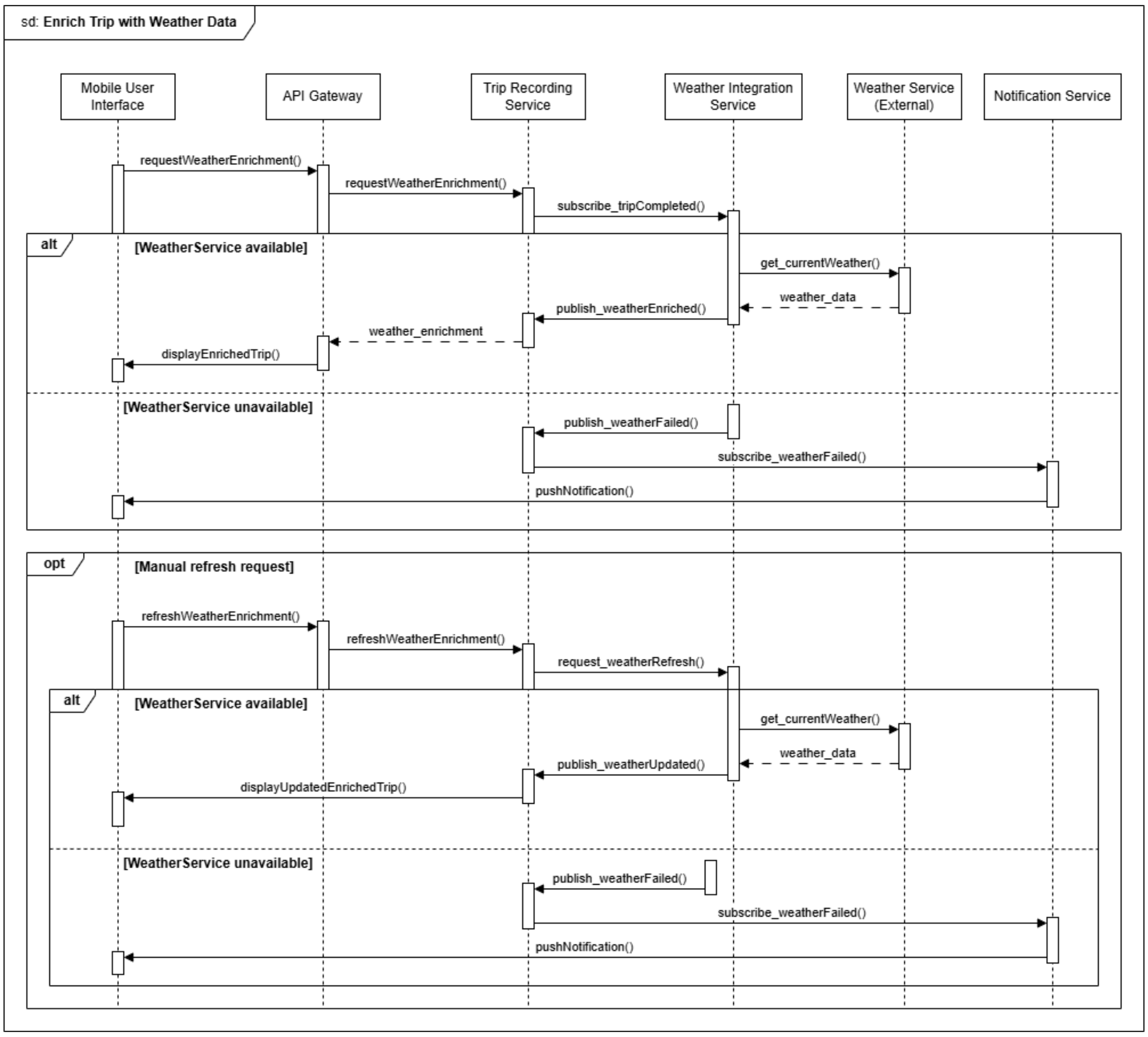
In the Runtime View we illustrate the dynamic interactions among the system's architectural components for a selected set of use cases. User interaction and device-specific events are abstracted by the Mobile User Interface, which represents the system boundary in all sequence diagrams. Only architecturally significant use cases are included, considering that trivial interactions do not contribute additional insight at the design level. The diagrams focus on the complete execution flow to clearly expose component responsibilities, so we intentionally omitted redundant interactions such as caching optimizations and notification handling when they do not affect the overall architectural behavior, even though they are present and play important roles in the production-level system. It is to be noted that the databases shown in the sequence diagrams are simplified to avoid ambiguity and clutter.

[UC4] Record Trip



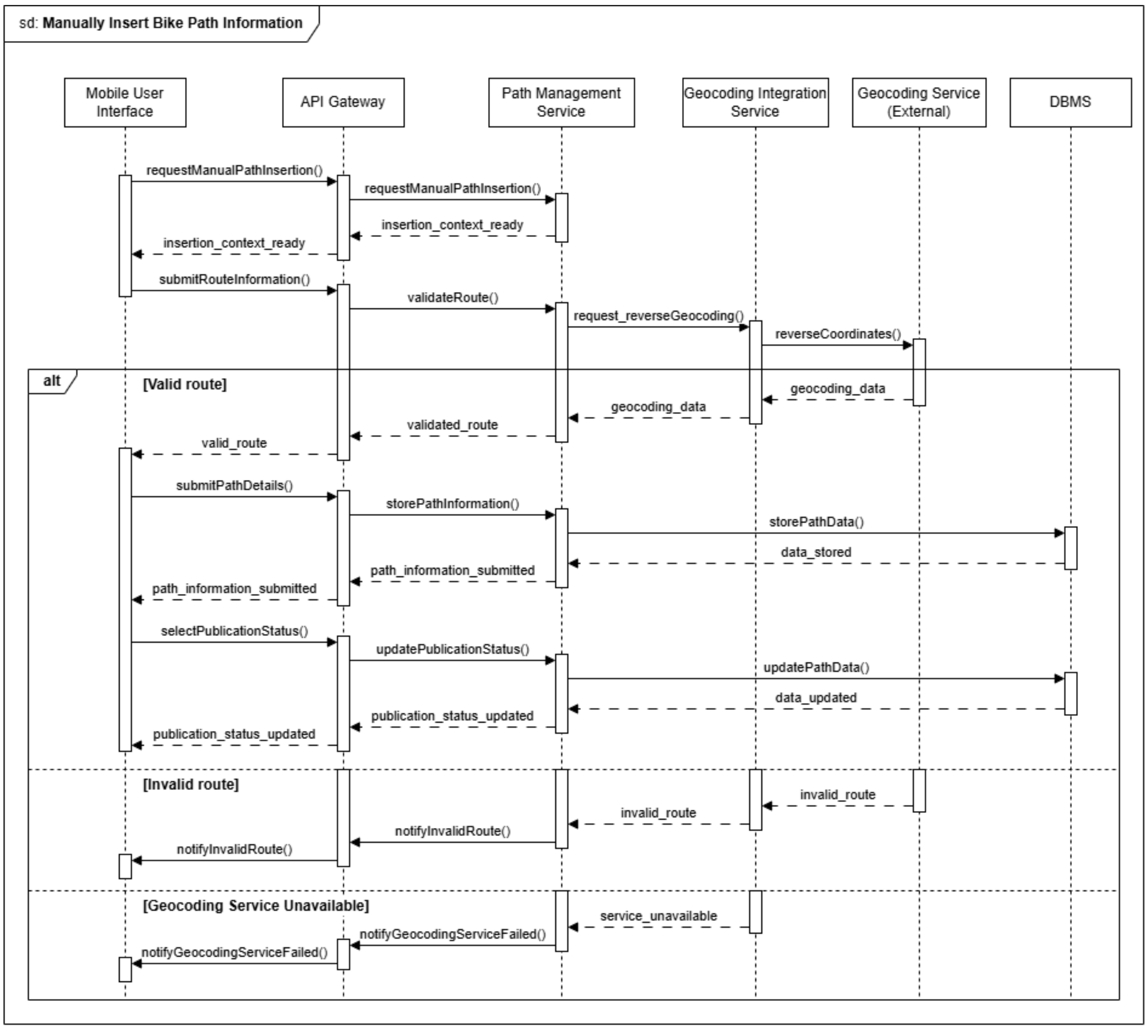
This sequence diagram illustrates the runtime interaction among the Mobile User Interface, device sensors, and backend services during trip recording. It shows permission handling, sensor data acquisition, cache buffering, trip lifecycle management, and persistence decisions. The diagram emphasizes backend responsibilities of trip state management while showing how sensor data is safely collected and transmitted during recording, pause, resume, and completion phases. Notice that the backend services do not communicate directly with the device sensors, as all interactions are mediated by the Mobile User Interface component, according to the defined layered architecture principles.

[UC6] Enrich Trip with Weather Data


sd: Enrich Trip with Weather Data
Mobile User Interface
API Gateway
Trip Recording Service
Weather Integration Service
Weather Service (External)
Notification Service
requestWeatherEnrichment()
requestWeatherEnrichment()
subscribe_tripCompleted()
alt
[WeatherService available]
get_currentWeather()
weather_data
publish_weatherEnriched()
weather_enrichment
displayEnrichedTrip()
[WeatherService unavailable]
publish_weatherFailed()
subscribe_weatherFailed()
pushNotification()
opt
[Manual refresh request]
refreshWeatherEnrichment()
refreshWeatherEnrichment()
request_weatherRefresh()
alt
[WeatherService available]
get_currentWeather()
weather_data
publish_weatherUpdated()
displayUpdatedEnrichedTrip()
[WeatherService unavailable]
publish_weatherFailed()
subscribe_weatherFailed()
pushNotification()

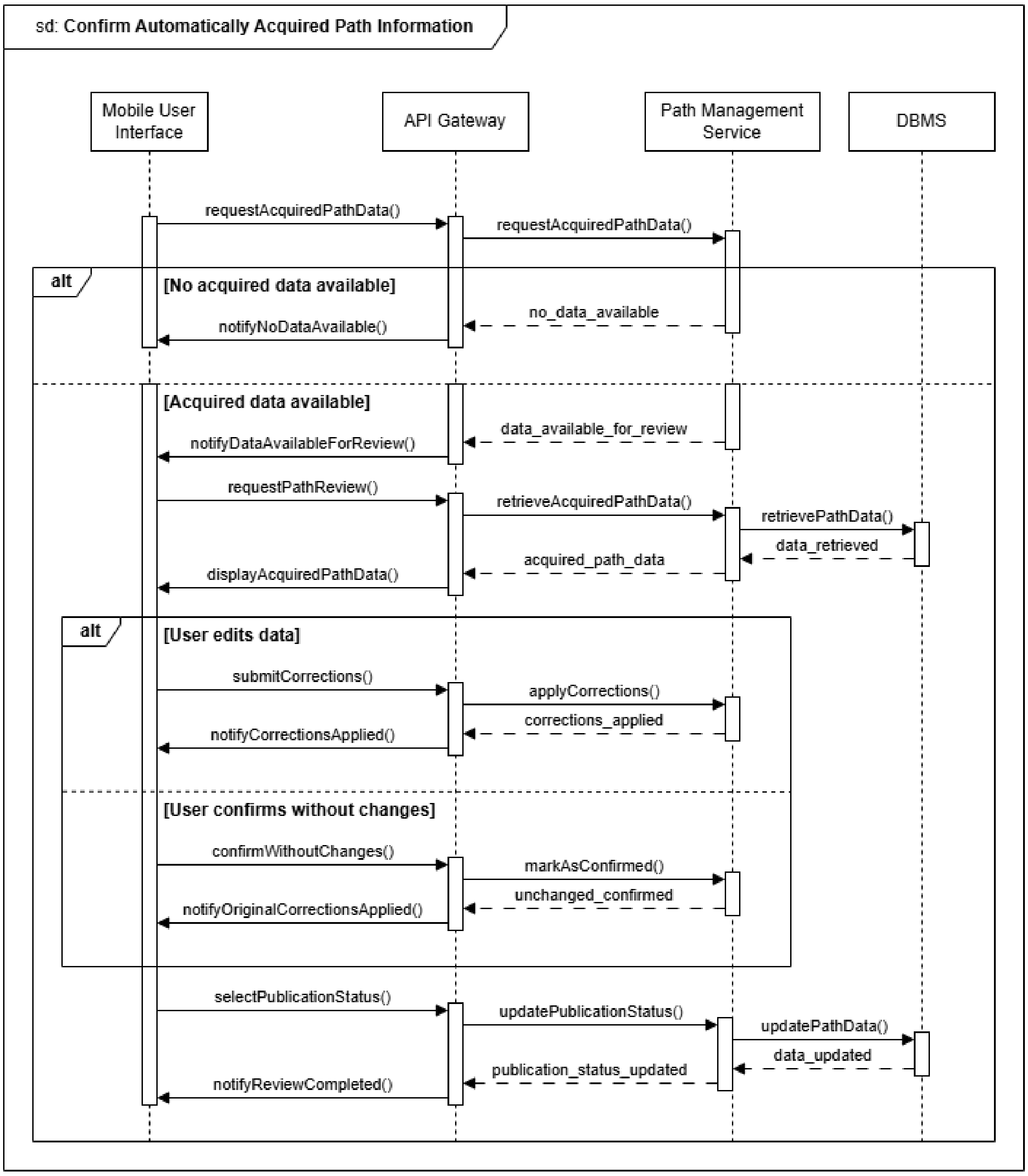
This sequence diagram illustrates the process through which completed trips are enriched with meteorological information obtained from an external weather service. It shows how the Trip Recording Service delegates external communication to the Weather Integration Service and how the system handles both successful enrichment and service unavailability. The use case also illustrates the notification flow for asynchronous enrichment events, and thus, will intentionally be omitted from other sequence diagrams to avoid redundancy. We assume that this same flow is followed whenever a service requires communication features from the Notification Service.

[UC7] Manually Insert Bike Path Information



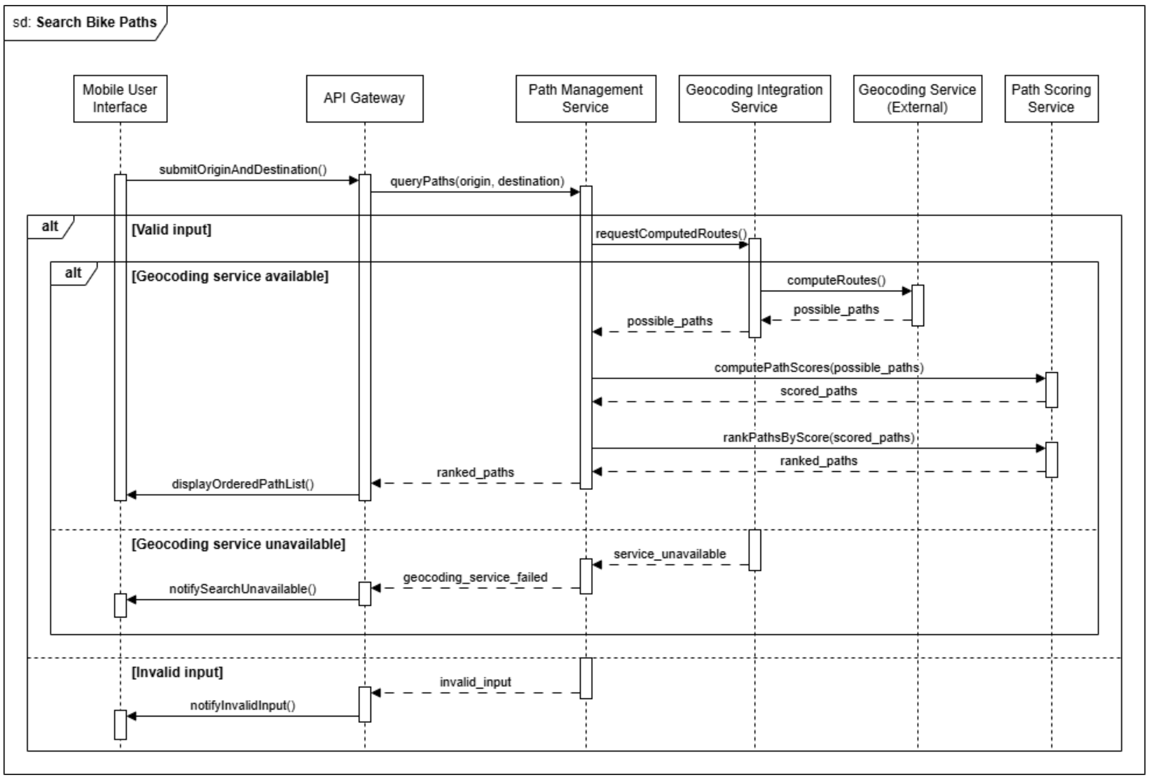
This sequence diagram illustrates the behavior associated with manual path data insertion by a `RegisteredUser`. It shows how the Path Management Service validates route information through the Geocoding Integration Service, manages user-provided path details, and persists the resulting data. Alternative flows illustrate invalid routes and external service unavailability, highlighting graceful degradation, which will be specifically emphasized in communication with external services.

[UC8] Confirm Automatically Acquired Path Information


sd: Confirm Automatically Acquired Path Information
Mobile User Interface
API Gateway
Path Management Service
DBMS
requestAcquiredPathData()
requestAcquiredPathData()
alt
[No acquired data available]
no_data_available
notifyNoDataAvailable()
[Acquired data available]
data_available_for_review
notifyDataAvailableForReview()
requestPathReview()
retrieveAcquiredPathData()
retrievePathData()
data_retrieved
acquired_path_data
displayAcquiredPathData()
alt
[User edits data]
submitCorrections()
applyCorrections()
corrections_applied
notifyCorrectionsApplied()
[User confirms without changes]
confirmWithoutChanges()
markAsConfirmed()
unchanged_confirmed
notifyOriginalCorrectionsApplied()
selectPublicationStatus()
updatePublicationStatus()
updatePathData()
data_updated
publication_status_updated
notifyReviewCompleted()

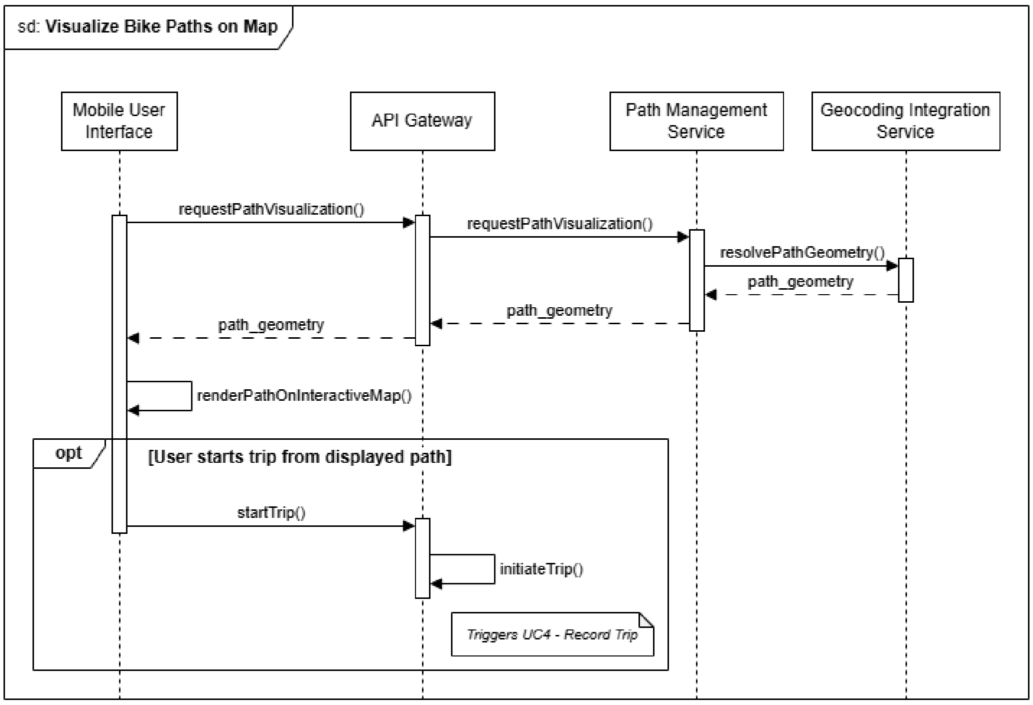
This sequence diagram illustrates the review and confirmation process for automatically acquired path data. It shows how the system retrieves pending path information, allows users to confirm or modify it, and handles publication decisions. The sequence emphasizes backend control of review state and publication status without involving external services.

[UC9] Search Bike Paths



This sequence diagram illustrates the collaboration among components during path querying and discovery. It shows how origin and destination queries are processed by the Path Management Service, how external geocoding is handled by an integration component, and how path scoring and ranking are delegated to the Path Scoring Service. The diagram also illustrates graceful handling of geocoding service unavailability.

[UC10] Visualize Bike Paths on Map


sd: Visualize Bike Paths on Map
Mobile User Interface
API Gateway
Path Management Service
Geocoding Integration Service
requestPathVisualization()
requestPathVisualization()
resolvePathGeometry()
path_geometry
path_geometry
path_geometry
renderPathOnInteractiveMap()
opt
[User starts trip from displayed path]
startTrip()
initiateTrip()
Triggers UC4 - Record Trip

This sequence diagram illustrates the visualization of existing paths on an interactive map. It focuses on retrieving path geometry from the backend and rendering it at the presentation layer. The use case also shows the optional transition to trip recording, explicitly linking visualization with the trip recording workflow (UC4) without duplicating its logic.

2.5 Component Interfaces

This section provides a summary of all the methods within the interfaces of the BBP system. All internal system components will rely on REST API interfaces. Note that the methods have been named with an initial prefix indicating the corresponding operation type (get/put/post/delete). Regardless of the provider, we have assumed that there are only a few methods available for external components.

The names of the classes used as method parameters were chosen to be as self-explanatory as possible. However, to avoid ambiguity, note the following:

- "User" represents a generic user, while "RegisteredUser" and "UnregisteredUser" are subclasses of User and are used when the parameter needs to specify a particular type of user.
- "Trip" represents a recorded bike trip with GPS data and computed statistics.
- "Path" represents a bike path published by users, with geometry (LineString - courtesy of PostGIS) and status information.
- "PathScore" is a computed metric (0-10) based on path status quality and routing effectiveness.
- "Obstacle" represents a hazard or challenge on a path (Pothole , Construction , Debris or Infrastructure).
- "TripData" includes metadata about a trip (distance_km, duration_seconds, avg_speed_kmh, max_speed_kmh, elevation_gain_m).
- "WeatherData" includes conditions at trip location (temperature_c, humidity, wind_speed_kmh, precipitation_mm, condition).

API Gateway Interface

- get_tripHistory(UserID, DateRange)
- get_tripDetails(TripID)
- get_pathInfo(PathID)
- get_userProfile(UserID)
- post_login(Email, Password)
- post_register(Email, Password, UserType)
- post_oauthLogin(Provider, Token)
- post_startTrip(UserID)
- post_stopTrip(UserID, TripID)
- post_submitPath(UserID, PathData, Geometry, ObstacleList)
- post_queryPaths(Origin, Destination)
- post_publishPath(UserID, PathID)
- put_updateUserProfile(UserID, UserData, PreferencesList)
- delete_draftPath(UserID, PathID)

Trip Recording Service Interface

- post_startRecording(UserID) - Initiates GPS data collection
- post_addGPSPoint(TripID, Latitude, Longitude, Timestamp, Accuracy, InstantaneousSpeed) - Add single GPS reading
- post_stopRecording(UserID, TripID) - Finalize trip recording and compute statistics

- get_realtimeMetrics(TripID) - Get current trip statistics during recording
- get_tripMetrics(TripID) - Get finalized trip statistics with weather enrichment
- put_pauseTrip(TripID) - Pause GPS recording temporarily
- put_resumeTrip(TripID) - Resume GPS recording after pause

Path Management Service Interface

- post_createDraftPath(UserID, PathName, PathDescription, Status) - Create new path in Draft state
- post_addPathGeometry(UserID, PathID, Geometry) - Add GeoJSON LineString to path
- post_addObstacle(UserID, PathID, ObstacleType, Location) - Add obstacle/hazard
- post_savePath(UserID, PathID) - Transition path from Draft to Save state - not published
- post_publishPath(UserID, PathID) - Transition path from Draft to Published state
- get_draftPaths(UserID) - Retrieve all draft paths for user
- get_publishedPaths(UserID) - Retrieve all published paths for user
- get_pathDetails(PathID) - Retrieve complete path information with geometry and obstacles
- delete_draftPath(UserID, PathID) - Remove path (only in Draft state)
- post_queryPaths(Origin, Destination) - Public Origin-Destination query endpoint
- get_pathScore(PathID) - Get current PathScore calculation through Path Scoring Service
- get_cachedQueryResult(Origin, Destination) - Check if results cached

Path Scoring Service Interface

- request_scoreComputation(PathList) - Triggered by Path Management Service during path discovery
- compute_pathScore(PathStatus, ObstacleList) - Compute score for a single path
- rank_paths(PathList) - Order paths by descending PathScore

User Authentication Service Interface

- post_register(Email, Password, UserType) - Create new user account (RegisteredUser or UnregisteredUser)
- post_login(Email, Password)
- post_oauthLogin(Provider, AuthCode) - OAuth login (Google, Apple)
- post_logout(UserID, AccessToken) - Invalidate user session
- get_userProfile(UserID) - Retrieve user profile and preferences
- put_updateProfile(UserID, UserData, PreferencesList, ProfilePicture) - Update user information
- put_changePassword(UserID, OldPassword, NewPassword) - Change account password

Geocoding Integration Service Interface

- request_routeQuery(Origin, Destination) - Triggered by Path Management Service during path discovery

- request_reverseGeocoding(Coordinates) - Triggered by Trip Recording Service for location resolution
- publish_routesResolved(QueryID, RouteList) - Emitted when candidate routes are successfully retrieved
- publish_geocodingFailed(QueryID, ErrorMessage) - Emitted when geocoding or routing requests fail or time out (5s timeout)
- get_coordinates(Address) - Resolve address to geographic coordinates via OpenStreetMap API
- get_routes(Origin, Destination) - Retrieve candidate routes between two locations
- get_cachedRoutes(Origin, Destination) - Check cache for previously resolved routes

Weather Integration Service Interface

- subscribe_tripCompleted(TripID, Location, Timestamp) - Triggered by Trip Recording Service
- publish_weatherEnriched(TripID, WeatherData) - Emitted when weather data successfully retrieved
- publish_weatherFailed(TripID, ErrorMessage) - Emitted when weather enrichment times out (5s timeout)
- get_currentWeather(Latitude, Longitude) - Query OpenWeather API
- get_cachedWeather(Latitude, Longitude) - Check cache
- post_enrichTrip(TripID, WeatherData) - Return weather data with trip

Notification Service Interface

- subscribe_pathPublished() - From Path Management Service
- subscribe_tripCompleted() - From Trip Recording Service
- subscribe_weatherEnriched() - From Weather Integration Service
- subscribe_weatherFailed() - From Weather Integration Service

Database Server (DBMS) Interface

Each of the following methods represent database operations performed on a relational database system. The prefix indicate the type of operation: ”query” corresponds to data retrieval or read, ”create” corresponds to insertion of new data, ”store” to update or add information to existing objects, ”delete” to remove entries.

- query_trips(UserID) - Retrieve all trips for user
- query_trip(TripID) - Retrieve specific trip with all metrics
- query_paths(UserID) - Retrieve all paths created by user
- query_path(PathID) - Retrieve path with geometry, obstacles, ratings
- query_userProfile(UserID) - Get user data and preferences
- query_search(Origin, Destination) - Execute Origin-Destination path search
- query_userTrips(UserID, DateRange) - Retrieve trips in date range
- query_pathRatings(PathID) - Get all ratings and reviews for path
- create_trip(UserID, TripData, GPSPoints) - Create trip record with GPS coordinates
- create_path(UserID, PathData, Geometry, LocationList) - Create path with geometry
- create_user(Email, Password, UserType) - Create user account

- store_tripMetrics(TripID, DistanceKm, DurationSeconds, Statistics) - Store computed statistics
- store_tripWeather(TripID, WeatherData) - Attach weather to trip
- store_pathPublished(PathID, PublishedState, Timestamp) - Mark path published
- store_userProfile(UserID, UserData, PreferencesList) - Update user profile
- delete_draftPath(PathID) - Remove path in Draft state
- delete_trip(TripID) - Remove trip from personal history (not once published)

Cache Server Interface

- cache_geocodingResult(Address, Coordinates, TTL=24h) - Cache address to coordinates mappings
- cache_weatherData(Location, WeatherData, TTL=24h) - Cache weather results
- get_cachedTrip(TripID) - Retrieve cached Trip data
- get_cachedWeather(Location) - Retrieve cached weather
- get_cachedQueryResult(Query) - Retrieve cached search results
- get_cachedSession(UserID) - Validate session token
- invalidate_tripCache(TripID) - Clear cache for a trip
- invalidate_queryCache(Region) - Clear cache for geographic region
- invalidate_geocodingCache(Address) - Clear address cache
- invalidate_weatherCache(Location) - Clear weather cache
- invalidate_sessionCache(UserID) - Clear session cache

2.5.1 External Service Interfaces

OpenWeather API

- GET /data/3.0/onecall - Get current and forecast weather

  Parameters: latitude, longitude, units (metric), appid

  Response: current: temp, humidity, wind_speed, description , alerts: [AlertData]
- GET /data/3.0/onecall/timemachine - Get historical weather data

  Parameters: latitude, longitude, dt (unix timestamp), appid

  Response: Historical weather data for past timestamps

  Timeout: 5 seconds (hard timeout with graceful degradation)

OpenStreetMap API

- GET /search?q=address&format=json - Geocode address to coordinates

  Response: latitude, longitude, display_name, osm_type
- GET /reverse?lat=latitude&lon=longitude&format=json - Reverse geocode coordinates to address

  Response: Address and location details from coordinates

Cache Server - Redis Interface

In-memory caching for performance optimization with TTL-based expiration.

Cache Operations:

- cache_tripData(TripID, TripData, TTL=48h) – Cache trip data during offline recording
- cache_weatherData(Location, WeatherData, TTL=24h) – Cache weather results
- cache_pathQueryResult(Query, Results, TTL=2h) – Cache Origin-Destination path search results

Retrieval Operations:

- get_cachedGeocoding(Address) – Retrieve cached coordinates
- get_cachedWeather(Location) – Retrieve cached weather
- get_cachedQueryResult(Query) – Retrieve cached search results

Invalidation Operations:

- invalidate_geocodingCache(Address) – Clear address cache
- invalidate_weatherCache(Location) – Clear weather cache
- invalidate_sessionCache(UserID) – Clear session cache

2.6 Selected Architectural Style

4 Tier Architecture

With a 4-Tire Architecture, the BBP system has a clear seperation between the user and the data by having many intermediate layers. This approach allows for a decoupled system where each layer has some specific responsibilities providing a flexible, scalable and secure system. The presentation layer is responsible for user interaction, the API Gateway and network components serve the function of handling certificates, enforcing security and also in caching. The backend application layer handles the business logic and processing with the help of microservices, and the data layer manages persistent storage. This separation of concerns facilitates easier maintenance, when it comes to scaling the Frontend horizontaly to accomodate more cyclists, and flexibility in development.

Microservice Architecture

The BestBikePaths system employs a microservice architecture where the core functionalities are divided into smaller, independently deployable services. Each microservice is responsible for a specific business capability such as trip recording, path management, user authentication, weather integration, and notifications. This approach allows for rapid development, deployment, and scaling of these services. This also prevents overloading of a single application further enhancing fault isolation, where a failure in one microservice does not necessarily impact the entire system. For example, the PathQueryService and TripRecordingService can be scaled or replicated independently during peak usage times without over provisioning the entire system.

API Gateway Pattern

To facilitate the communication between the mobile application and the backend microservices, the BBP system employs an API Gateway pattern. The API Gateway serves as the main entry point for all client requests, handling authentication, routing, rate limiting, and load balancing. This provides a unified interface for interaction. It also enhances security by enforcing consistent policies and protecting the backend services.

Circuit Breaker Pattern

To guarantee system resilience and availability, especially with external services such as OpenStreetMap API and OpenWeather API, which are prone to latency and failures, the system makes use of the circuit breakers. They act by preventing cascading failures by preventing calls to an external service that is currently failing or experiencing high latency.

This allows the system to degrade gracefully, providing fallback responses or alternative flows when external dependencies are unavailable. This is experienced first hand in the Weather Integration Service where a hard timeout of 5 seconds is enforced when querying the OpenWeather API. In case of a timeout, the system gracefully degrades by saving the trip recording without the weather data.

State Pattern for Trip and Path Lifecycle

The system includes entities such as Trips and user-created Paths that have distinct lifecycle states. For instance, a Trip usually transits through different state like "Recording", "Paused", and "Completed", while a Path can be in states like "Draft", "Saved", and "Published". The State Pattern is employed to manage these lifecycle transitions explicitly. Each state encapsulates the behavior associated with that particular state, allowing for clear enforcement on what operations are permissible in each stage.

2.7 Other Design Decisions

PathScore Computation Logic

The path score is an integral and fundamental part of the BBP system and is encapsulated within the Path Scoring service. This metric is predominantly used to rank paths during the manual path query functionality. The path score computation logic in the Path Scoring Service is responsible for evaluating the quality and safety of the bike path based on the path condition and obstacles present on the path. It takes into account the path condition and the severity of obstacles such as potholes, debris, construction zones and infrastructure issues to come up with a weighted score between 0 (Poor Quality) to 10 (Excellent Quality). This modular design allows for fine-tuning of the scoring weights as new data or new feedback is obtained

The weights for the path condition and obstacles are determined below:

Table 1: Path Score Weights

Path Status	Weighted Score	Impact
Optimal	10	Properly maintained
Medium	7	Moderate condition
Sufficient	5	Usable but need work
Maintenance required	3	Need major maintenance

Table 2: Obstacle Weights

Obstacle Type	Weighted Score	Impact
Pothole	1	Critical dangerous hazard
Construction	0.6	Active work area, lane closure
Debris	0.4	Trash, fallen leaves and broken branches
Infrastructure	0.2	Signs, benches and other infrastructure

The path score is computed as follows: Step 1: Calculating obstacle score:
$weighted\_obstacle\_score = \sum(obstacle\_count\_i \times obstacle\_weight)$
For example, if a path has a path score of 10 and for obstacles has 3 potholes, 2 construction zones and 5 debris obstacles and 4 infrastructure obstacles, the weighted obstacle score would be calculated as:

$weighted\_obstacle\_score = (3 \times 1) + (2 \times 0.6) + (5 \times 0.4) + (4 \times 0.2) = 3 + 1.2 + 2 + 0.8 = 7$
Step 2:Calculating Obstacle Weight::

$$obstacle\_density = \frac{Weighted\_obstacle\_score}{Path\_distance}$$

For the above example, if the total path distance in km is 2.5. Therefore, the obstacle density would be:

$$obstacle\_density = \frac{7}{2.5} = 2.8\ weighted\_obstacle\_per\_km$$

Step 3: Final Safety Score Calculation:

$$\begin{aligned} Safety\_Score &= 10 - (obstacle\_density \times 1.25) \\ &= max(0, Safety\_Score) \end{aligned}$$

For the above example, the final safety score would be:

$$Safety\_Score = 10 - (2.8 \times 1.25) = 10 - 3.5 = 6.5$$

Step 4: Normalizing the Path Score:

$$Path\_Score = (Status\_Score \times 0.7) + (Safety\_Score \times 0.3)$$

For the above example, assuming the path condition is "Optimal" with a status score of 10, the final path score would be:

$$Path\_Score = (10 \times 0.7) + (6.5 \times 0.3) = 7 + 1.95 = 8.95$$

Automated Anomaly Detection

The automated nature of BBP system for reporting of potholes or speed bumps is a crucial feature that greatly enhances the user experience. To achieve this, the Trip Recording Service incoperates an obstacle detection algorith which relies on the sensons readings of both accelerometer and gyroscope data from the mobile device. The system accquires raw signals from Device Sensor Adapter which houses the two sensors, the system then monitors the data to identify pattern which reflect physical entities like riding over potholes or surface irregularities

To aid the user in accurate reporting, the system uses live-notifications to alert the user. If he/she wishes to record the detected obstacle in the form of a notification which stays on the screen for 5 seconds. A desgin decision critical to the BBP system is confirming the obstacle is true. A cyclist can simply ignore the live notification to confirm recording of the obstacle. This has been done incase of false positives where the system detects an obstacle when there is none as they can trigger while riding over a cobblestone path or a speed bump even though high-pass filter for frequencies of (2Hz to 10Hz) for these usecases is already implemented. If the user accepts the notification, the obstacle is recorded and added to the path data. This approach minimizes false positives by involving user confirmation while still leveraging automated detection to streamline the reporting process. The energy from the vibration can be used to detect obstacles using the accelerometer data. The energy can be computed using the following formula:

$$\text{energy} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Where a_x, a_y, and a_z are the accelerometer readings along the x, y, and z axes respectively. When the computed energy exceeds a predefined threshold (e.g., 1.5g), an obstacle event is triggered.

Similarly, gyroscope data can be used to detect sudden changes in orientation. The angular velocity can be computed using the following formula:

$$\omega_{\text{total}} = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$$

Where ω_x, ω_y, and ω_z are the gyroscope readings along the x, y, and z axes respectively. When a sudden change in angular velocity is detected, an obstacle event is triggered. A combination of both accelerometer and gyroscope data can be used to improve the accuracy of obstacle detection. For example, an obstacle event is only triggered when both the energy from the accelerometer and the angular velocity from the gyroscope exceed their respective thresholds. This is used to negate false positives from sudden movements such as breaking hard, other road undulations that are not related to obstacles. Further the can account for speed of the cyclist by using the GPS data to adjust sensitivity of the obstacle detection algorithm. At higher speeds, the thresholds for energy and angular velocity can be increased to reduce false positives, while at lower speeds, the thresholds can be decreased to improve sensitivity.

Proxy Manager

The presence of a proxy manager along with the API Gateway adds an additional layer of security and abstraction between the client and backend services. The Proxy manager usually handles the decryption of incoming HTTPS requests and SSL/TTL certificates. It also helps in managing outbound requests to third-party services like OpenWeather API, OpenStreetMap API and OAuth providers in the BBP system. This separation of concerns allows the API Gateway to focus on routing, authentication, and rate limiting, while the Proxy Manager can deal with other network tasks like caching where the manager stores copies of frequently requested data, conducting security checks before sending requests to the main backend server.

Rate limiting at API Gateway

To protect the main backend microservices from abuse and also from denial-of-service attacks, rate limiting implementation at the API gateway ensure that each user can only make a certain number of requests within a stipulated timeframe. This facilitates fairness between users usage by maintaining an experience which is consistent.

Timeouts and Graceful Degradation

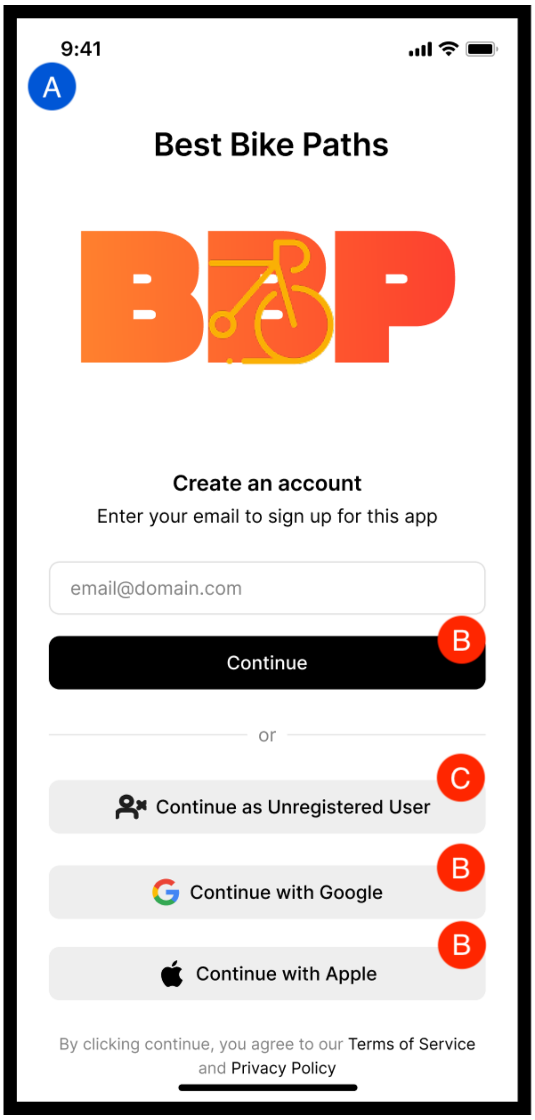
with integration of external services like OpenWeather API for weathersnashots, OpenStreetMap API for geocoding it is crucial to implement timeouts to prevent the system from hanging indefinitely while waiting for a response. Hence,the BBP system enforces a hard timeout for these external services like a timeout of 5 seconds for weather data retrieval. Incase the external service does not respond within the stipulated time, the system gracefully degrades by saving the trip recording without the weather data. This ensures that users can still complete their trips and have their trip data recorded even if the weather service is temporarily unavailable. This can be retroactively added to the trip data when the weather service becomes available again.

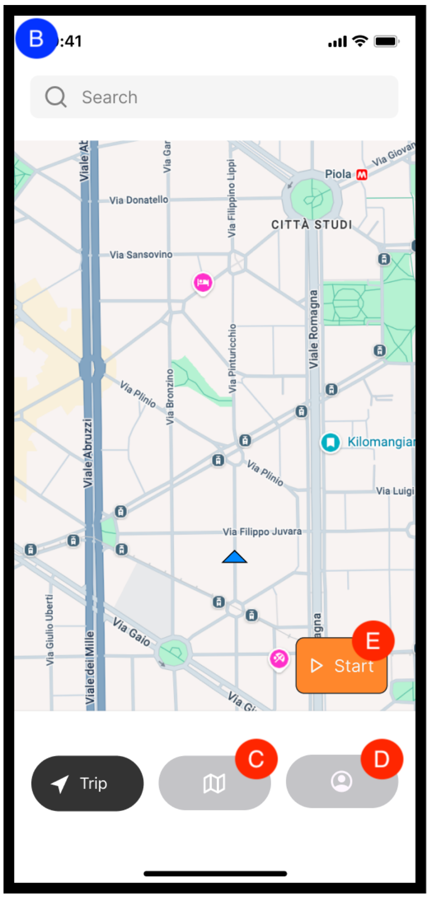
3 User Interface Design

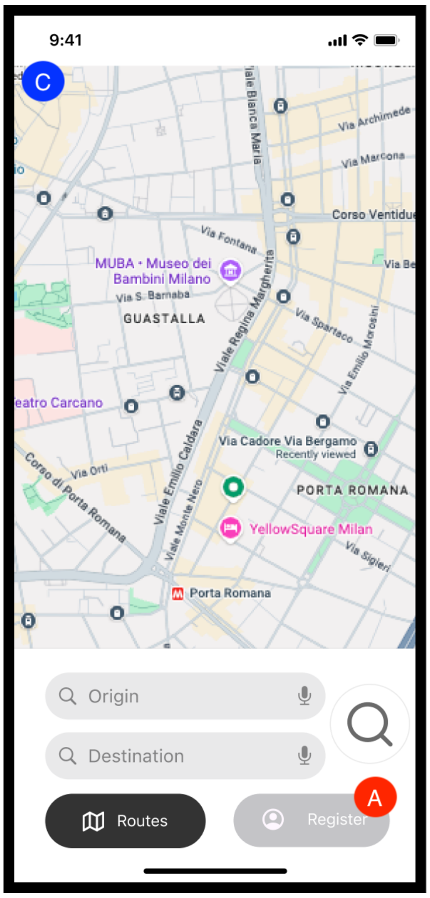
In this section, we outline the design principles and components of the user interface for our application. The UI is designed to be intuitive, responsive, and accessible across various touch enabled devices. The user interaction aspects have been highlighted.

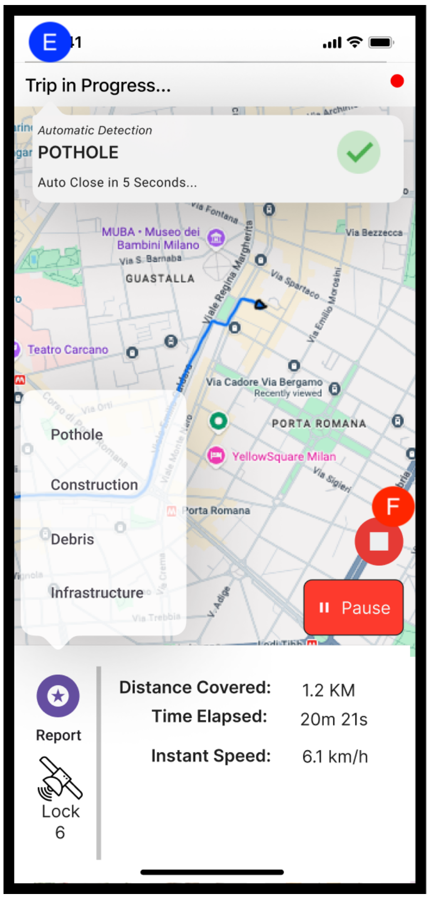
Legend:

- Red circles with letters indicate the main user interface components.
- Blue circles with letter indicate corresponding landing page once an UI element is pressed.


9:41
A
Best Bike Paths
BBP
Create an account
Enter your email to sign up for this app
email@domain.com
B
Continue
or
C
Continue as Unregistered User
B
Continue with Google
B
Continue with Apple
By clicking continue, you agree to our Terms of Service and Privacy Policy


B
:41
Search
E
Start
Trip
C
D


9:41
C
Origin
Destination
Routes
Register
A



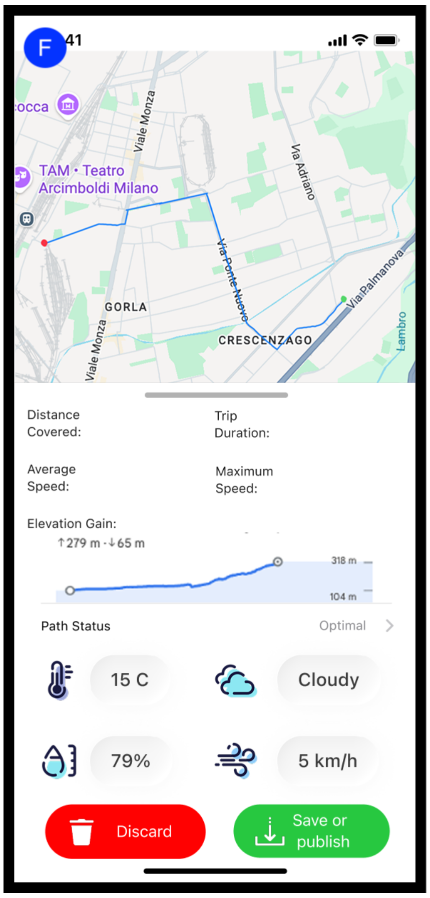
Trip in Progress...

Automatic Detection
POTHOLE
Auto Close in 5 Seconds...

Pothole
Construction
Debris
Infrastructure

Pause

Distance Covered:	1.2 KM
Time Elapsed:	20m 21s
Instant Speed:	6.1 km/h

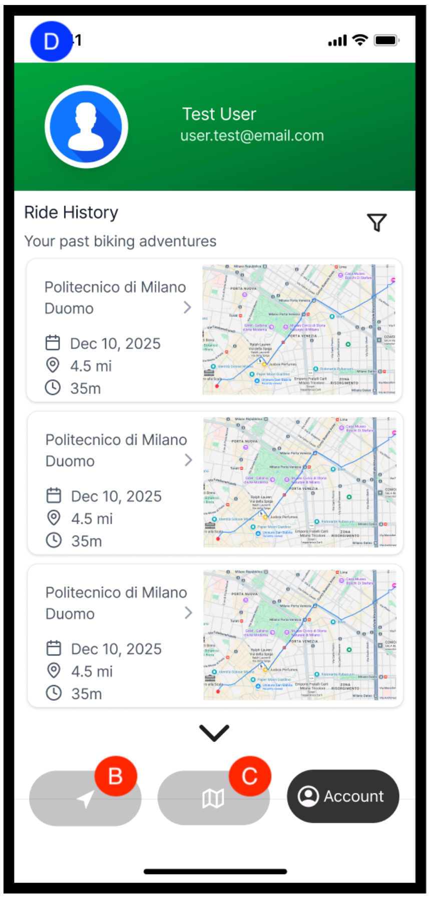


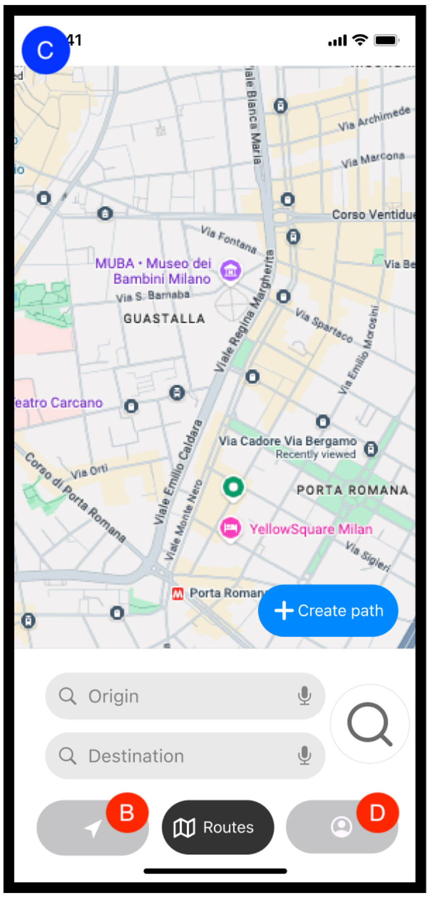
Distance Covered:
Trip Duration:
Average Speed:
Maximum Speed:
Elevation Gain:

Path Status Optimal

15 C
Cloudy
79%
5 km/h

Discard
Save or publish


D
Test User
user.test@email.com
Ride History
Your past biking adventures
Politecnico di Milano
Duomo
Dec 10, 2025
4.5 mi
35m
Politecnico di Milano
Duomo
Dec 10, 2025
4.5 mi
35m
Politecnico di Milano
Duomo
Dec 10, 2025
4.5 mi
35m
B
C
Account


C
Create path
Origin
Destination
B
Routes
D

4 Requirements Traceability

This section describes the relationship between the functional requirements of the Best Bike Paths (BBP) system specified in the RASD and the architectural components defined in the Architecture Design section of this document. The traceability mapping identifies the components responsible for realizing each requirement, ensuring that all required functionalities are covered by the proposed design and that responsibilities are clearly distributed among system components.

User Access and Profile Management

Requirements	**Components**
R1: The system shall allow a `User` to sign up and create a new account.	Mobile User Interface API Gateway User Authentication Service DBMS
R2: The system shall allow a User to continue as an `UnregisteredUser`.	Mobile User Interface API Gateway User Authentication Service
R3: The system shall allow a `RegisteredUser` to log in.	Mobile User Interface API Gateway User Authentication Service
R4: The system shall allow a `RegisteredUser` to update and modify profile information.	Mobile User Interface API Gateway User Authentication Service DBMS
R5: The system shall restrict an `UnregisteredUser` to path querying and visualization functionalities only.	API Gateway User Authentication Service Path Management Service

Trip Recording and Sensing

Requirements	**Components**
R6: The system shall request User permission to access GPS and motion sensors on the device.	Mobile User Interface Device Sensors Adapter
R7: The system shall detect biking activity based on an established minimum sustained velocity.	Mobile User Interface Device Sensors Adapter
R8: The system shall allow a `RegisteredUser` to initiate trip recording.	Mobile User Interface API Gateway Path Management Service
R9: The system shall display real-time trip data during recording.	Mobile User Interface

R10: The system shall allow a `RegisteredUser` to pause and resume trip recording.	Mobile User Interface API Gateway Path Management Service
R11: The system shall allow a `RegisteredUser` to discard a trip before final upload.	Mobile User Interface API Gateway Path Management Service
R12: The system shall compute trip statistics.	Trip Recording Service
R13: The system shall store a completed trip and its data if the `RegisteredUser` confirms saving it.	Mobile User Interface API Gateway Trip Recording Service DBMS
R14: The system shall allow a `RegisteredUser` to view past trips and access trip history.	Mobile User Interface API Gateway Trip Recording Service DBMS
R15: The system shall prevent GPS data loss during network unavailability.	Mobile User Interface Cache Server Interface

Weather Enrichment

Requirements	**Components**
R16: The system shall retrieve meteorological data for a completed trip from an external service.	Weather Integration Service
R17: The system shall attach weather information with the trip if available.	Trip Recording Service Weather Integration Service
R18: The system shall allow the User to manually refresh weather enrichment.	Mobile User Interface API Gateway Weather Integration Service
R19: The system shall gracefully degrade if the weather service is unavailable.	Trip Recording Service Weather Integration Service

Manual and Automated Path Contribution

Requirements	Components
R20: The system shall allow a `RegisteredUser` to manually insert bike path information.	Mobile User Interface API Gateway Path Management Service DBMS
R21: The system shall allow a `RegisteredUser` to review and confirm automatically acquired path information.	Mobile User Interface API Gateway Path Management Service
R22: The system shall publish path information when the `RegisteredUser` makes it publishable.	Path Management Service DBMS
R23: The system shall allow a `RegisteredUser` to control the publication of contributed path information.	Mobile User Interface API Gateway Path Management Service
R24: The system shall gracefully degrade if the geocoding service is unavailable.	Path Management Service Geocoding Integration Service

Path Discovery and Visualization

Requirements	Components
R25: The system shall compute a `PathScore` for each path based on `PathStatus` and obstacles present on the path.	Path Scoring Service
R26: The system shall list routes ordered by descending `PathScore`.	Path Management Service Path Scoring Service
R27: The system shall allow any User to visualize routes on an interactive map.	Mobile User Interface
R28: The system shall allow any User to query paths between an origin and a destination.	Mobile User Interface API Gateway Path Management Service Geocoding Integration Service

5 Implementation, Integration and Test Plan

5.1 Overview

This chapter outlines the implementation, integration, and testing strategies for the BBP system with the primary goal being the development of mobile centered architecture with microservices across 4 tier architecture. It details the development environment, tools, and methodologies that will be employed to ensure a robust and reliable application. The chapter also describes the testing plan, including unit tests, integration tests, system tests, and user acceptance tests to validate the functionality and performance of the system.

The system can be broken down into main components like the mobile application, backend microservices, database, and external service integrations. For the process of implementing, integrating, and testing these components, Bottom-up approach with Thread strategy will be used.

5.2 Implementation Plan

The implementation of the BBP system will be carried out in iterative phases, following combined bottom-up and thread strategy to ensure that each component is developed, integrated, and tested thoroughly before moving on to the next one. This implementation further facilitates bug tracking by allowing incremental development and testing of each component.

- Thread Strategy focuses on implementing and testing end to end functionality of vital features that spans across multiple components. One such example is the trip recording and weather enrichment feature that involves implementation and testing of mobile application, backend microservices, database interactions, and external weather API integration.
- Bottom-up approach will be used to implement and test individual components starting from the lowest level such as databases, Device sensor adapters before being integrated into higher-level services such as Trip recording services. This approach ensures that each low level dependency if fully functional before being integrated into more complex higher level services.

The implementation will be carried out in the following phases:

- **Phase 1: Foundation** This phase focuses on setting up the development environment, version control systems, and continuous integration pipelines. beginning with design and implementation of data schemas, and core microservices such as user authentication services, Trip recording services- basic CRUD and API Gateway will be developed and tested. Along with these a basic UI for user registration and login with the ability to view empty trip list will be implemented
- **Phase 2: Core Functionality** In this phase, the primary functionality of the mobile application will be implemented such as device sensory adapters in client side for GPS and motion sensor data. In addition to this trip recording will get enhanced functionality with operations such as starting, pausing, resuming, completing a trip for computing trip statistics. Persist trip and statistics in the database and expose them to the corresponding API and add basic user interface components. Each feature will be developed using the bottom-up approach, ensuring that all underlying services are functional before integrating them into the mobile application.
- **Phase 3: Advanced Features** This phase will focus on implementing advanced features such as data visualization, Path Management Service, Path Scoring Service and the interaction between them. Integration of Geocoding services to aid path management for Origin-Destination queries and integration tests will be conducted to ensure that all components work seamlessly together.

- **Phase 4: Final Integration and Testing** The final phase involves implementation of weather integration services and its interaction with external agents, along with implementation of notification services to aid in delivering push notifications. Subsequently, event bus is implemented for asynchronous communication between services. Comprehensive system testing, including performance testing, security testing, and user acceptance testing is to be carried out. Any identified issues will be addressed before the final deployment of the application.

5.3 Integration Plan

The integration of the BBP system components will be carried out in a systematic manner to ensure that it happens at service boundaries rather than database level. This approach minimizes the risk of integration issues and ensures that each component can communicate effectively with others. The integration plan follows the bottom-up approach with thread based strategy, starting with the integration of low-level components and gradually moving up to higher-level services. Testing in mobile environments presents unique challenges due to the diversity of devices, operating systems, and network conditions. Therefore, the testing plan includes strategies to address these challenges, such as testing on a range of devices and emulators, as well as simulating different network conditions to ensure robust performance across various scenarios. Integration will occur across the three primary interfaces:

- **Synchronous Communication**: Communication between the mobile layer and the backend are fulfilled via API Gateway with the use of RESTful APIs over HTTPS. Each microservice exposes its functionality through well-defined endpoints, allowing the mobile application to interact with them seamlessly. API Gateway handles request routing, authentication, and rate limiting to ensure secure, efficient communication and decoupling from the microservices.
- **Asynchronous Communication**: For inter-service communication within the backend, an event-driven architecture is employed using a message broker, in the case of BBP system, through RabbitMQ. This allows services to publish and subscribe to events, enabling loose coupling and scalability. For example, when a trip is completed, the Trip Recording Service can publish an event called `Trip.Completed` that the Weather Integration Service subscribes to for weather enrichment. Integration testing at this phase can look into apt event publishing and consumption across services.
- **Data Persistence**: The BBP system uses a database per service for data persistence, with each microservice managing its own schema enriched database. Integration at this level involves ensuring data consistency before being exposed to services APIs. For example, the Trip Recording Service must ensure that trip data is correctly stored and retrieved from its database before being made available through its API endpoints.

Services are to be integrated in an incremental fashion, starting with the core services such as User Authentication Service and Trip Recording Service, followed by Path Management Service and its peer Path Scoring Service, Weather Integration Service, and Notification Service. Each integration step is followed by rigorous testing to validate the interactions between services and ensure that data flows correctly across the system. Once the microsrvices are integrated, API Gateway is to be configured with routing rules for each microservice, along with security policies such as authentication and rate limiting to ensure secure and efficient communication between the mobile application and backend services. Caching strategies are also implemented at the API Gateway level to enhance performance and reduce latency for frequently accessed data. As far as integration of client side mobile application is concerned, initially mock servers are used to simulate the services which can gradually be replaced with real API endpoints as the backend services get integrated and stabilized. This approach allows for parallel development of the mobile application and backend services, reducing overall development time. Testing can be carried out on iOS emulators and Android ones, along with testing for offline caching. In addition to these and similarly integration of graceful degradation of BBP system with Redis for caching needs to be Implemented.

Further to main high integration stabillity, A continuous integration (CI) pipeline is established to automate the build with static code analysis checking for syntax errors and standard code violations including readability and naming convention to name a few , unit testing to be carried out to test logic within services and deployment of these successful builds with the help of containers. In addition to these, since the devices that run the BBP is diverse in terms of hardware capabilities, operating systems and also with respect to network conditions, Integration planning must include strategies to address these pitfalls.

5.4 System Testing

The system testing plan aims to validate the behavior of the fully integrated BBP system under realistic usage conditions, once all the components have been integrated. We propose different strategies to guarantee that the system behaves as expected under different circumstances and case scenarios, which are derived from system functional requirements and use cases and focus on verifying complete realistic workflows along all the layers in the 4 tier proposed architecture, including mobile interaction patterns, stateful user actions, long-running activities, and dependencies on external services.

The functional testing strategy verifies that end-to-end user workflows behave according to requirements, validating core functionality such as trip recording, path discovery, manual and automatic path contribution, and data visualization. We give special attention to the case of stateful conditions such as trip lifecycle management and path publication and visibility rules, for which we can follow defined constraints that allow us to assess a correct management of data. For example, automatically sensed path data remains private by default until a user explicitly confirms its publication. We also relate functional testing to the differentiation among registered and unregistered users and the proper enforcement of access restrictions.

Performance testing evaluates the responsiveness of the system under expected conditions. Since we have defined expected completion times for specific service interactions, the goal is to ensure that those metrics are being held such that system remains responsive during normal usage. Load testing extends this strategy by applying increased usage conditions like concurrent access from multiple users, so we can assess and guarantee that BBP system still can sustain a higher service demand without functional degradation. Furthermore, stress testing examines the robustness of the system under extreme conditions such as eventual spikes in requests or region unavailability, to validate gracefull degradation strategies, resilience and data recovery mechanisms.

On the other hand, usability testing focuses more on the presentation layer of the architecture, assessing parameters like intuitiveness and responsiveness of the system touch-based user interface for mobile devices. This strategy handles the verification of core interaction flows, such as starting and stopping trip recording, reviewing path publication, requesting routes between origin and destination, navigating through maps, etc. Usability testing complements the current system testing strategy by aiming to ensure a smooth user experience rather than focusing on architectural correctness.

Security and privacy testing is included in the testing plan to verify that the system adequately protects data and is resilient to common attacks. We address testing for well-known threats such as injection attacks and cross-site scripting, as well as the correct handling of user sensitive data like location. These tests ensure that GPS traces and user information is stored, transmitted and accessed complying with privacy requirements.

One important aspect of the complete system testing strategy is the validation of reliability and fault-tolerance mechanisms that happens at runtime. For example, offline data handling with the help of caching adapters should be assessed to guarantee synchronization once connectivity is restored. On the other hand, expected retry strategies for interactions with external services are also expected to be gracefully handled. These scenarios are critical to ensuring correct system behavior in mobile environments where network conditions may be unstable. External services, including geocoding and weather providers, are replaced by simulated endpoints during testing to ensure repeatability and to allow controlled evaluation of scenarios such as timeouts and service unavailability. Test data is isolated across test runs, and persistent storage is reset to a known state to avoid cross-test interference.

Finally, acceptance testing is performed to validate that the system meets expectations and regulatory requirements. This includes compliance testing against data protection regulations such as GDPR. Also, Alpha and Beta testing phases are included for cases that involve real-life sensor data to confirm that the system behaves correctly under realistic operating conditions that cannot be emulated accurately on controlled environments.

5.5 Additional Considerations for Testing

As an additional measure, automated testing is integrated into the development workflow to ensure that unit tests, integration tests, and system tests are executed regularly. This continuous testing approach helps in early detection of issues and maintains the overall quality of the system throughout the development lifecycle.

Due to the nature of mobile and third-party dependencies, certain aspects such as real-time sensor accuracy and external service reliability can only be partially assessed during testing. Where feasible, these aspects are validated through controlled experiments and user feedback during the beta testing phase.

Developers should include both alpha and beta testing stages. In the alpha stage, the development team and a small group of stakeholders test the application internally to detect and resolve major issues. After that, a beta stage takes place, allowing a broader group of users to try the application in real-world conditions. All testing activities should follow industry best practices and be fully documented to track issues and resolutions with the use of a task management software.

6 Effort spent

The following table displays the effort spent by each team member on the different sections of this document, measured in hours. The division of work is only indicative and may not reflect the actual time spent by each member as each section requires collaboration from all members.

Team Member	**Sections 1**	**Section 2**	**Section 3**	**Section 4**	**Section 5**
Cristhian Mejia	3	12	0	4	5
Sravan Yerranagu	0	15	3	0	12

7 References

References

[1] Software Engineering 2, *2025/26 Requirement Engineering and Design Project Document* The specification for the 2025/26 Requirement Engineering and Design Project for the Software Engineering II course.

[2] Software Engineering 2, *Requirements Analysis and Specification Document* The RASD document for the 2025/26 Requirement Engineering and Design Project for the Software Engineering II course.

[3] The slides on the WeBeep page of the Software Engineering II course.

[4] JGraph, *diagrams.net (draw.io)*, version 29.0.3. Available at: `https://www.diagrams.net/`.

[5] Grasset, Arnaud. *PlantUML - Open Source Unified Modeling Language (UML) Diagram Tool.* Available at: `https://plantuml.com/`

[6] OpenWeather, *OpenWeather API Documentation.* Available at: `https://openweathermap.org/api`

[7] OpenStreetMap, *OpenStreetMap API Documentation.* Available at: `https://wiki.openstreetmap.org/wiki/API`

[8] OWASP Foundation - Standrd Awareness Document, *OWASP Top 10 - 2021.* Available at: `https://owasp.org/Top10/2021/`

[9] D. Hardt, *The OAuth 2.0 Authorization Framework*, datatracker.ietf.org, Oct. 2012. Available at: `https://datatracker.ietf.org/doc/html/rfc6749`

[10] REST API Tutorial, *REST API Tutorial: Learn REST API Design*, Available at: `https://www.restapitutorial.com`.

[11] The PostgreSQL Global Development Group. *PostgreSQL 15 Documentation.* Available at: `https://www.postgresql.org/docs/15/`

[12] PostGIS Project. *PostGIS 3.3 Documentation - Spatial Database Extender for PostgreSQL.* Available at: `https://postgis.net/documentation/`

[13] Figma, Inc. *Figma - Collaborative Design Platform.* Available at: `https://www.figma.com/`

[14] European Union, *General Data Protection Regulation (GDPR)*, Official Journal of the European Union, 2016/679. Available at: `https://gdpr-info.eu/`.