Travlendar+ project by Kaifei Xu, Shinuo Yan, Yanglin Hu

POLITECNICO
MILANO 1863

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Kaifei Xu, Shinuo Yan, Yanglin Hu
Version:	1.0
Date:	07-Jan-2026
Download page:	[https://github.com/YanglinHu97/HuXuYan.git]
Copyright:	Copyright © 2026, Kaifei Xu, Shinuo Yan, Yanglin Hu – All rights reserved

1. Introduction

1.1 Purpose

This Design Document (DD) describes the architectural and detailed design of **Best Bike Paths (BBP)**, a collaborative system that supports cyclists in finding safer and smoother routes by leveraging crowd-sourced road-quality information. The DD refines the requirements defined in the RASD into a concrete design, including system components, interactions, interfaces, deployment choices, and key design decisions. The document is intended for developers and reviewers to understand *how* the system will be built and how it satisfies the RASD requirements.

1.2 Scope

BBP mediates between real-world road conditions and a digital map consumed by cyclists. The system supports: (i) recording personal trips, (ii) inserting publishable information manually, (iii) automatic detection of anomalies (e.g., potholes) using smartphone sensors during cycling with a human-in-the-loop confirmation process, (iv) aggregation/merging of multi-user publishable information while considering factors such as freshness and confirmations, and (v) path computation and visualization ranked by a route score that reflects both effectiveness and road-quality status. BBP integrates with external services for map functionalities and (when available) weather enrichment.

1.3 Definitions, Acronyms, Abbreviations

Key terms are inherited from the RASD, including: Segment, Path, Detection, Confirmation, Publishable Information, and Aggregation, where automatic detections become publishable only after explicit user confirmation, and segment status is derived from aggregated publishable inputs.

1.4 Revision History

Version 1.0 — Initial DD release for BBP design, derived from the corresponding RASD and the course assignment template.

1.5 Reference Documents

Course project assignment document (A.Y. 2025–2026) describing DD structure and rules.
BBP Requirement Analysis and Specification Document (RASD).

1.6 Document Structure

This DD is organized as follows: Chapter 2 defines the architectural design (components, deployment, runtime interactions, interfaces, patterns, and design decisions). Chapter 3 outlines the user interface design. Chapter 4 provides requirements-to-design traceability. Chapter 5 defines implementation, integration, and testing plans, followed by effort reporting and references.

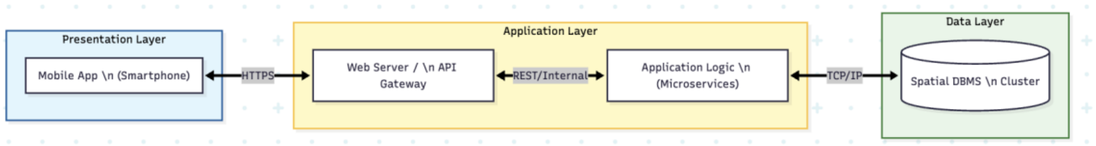
2. Architectural Design

2.1 Overview

This section provides a high-level representation of the BBP system architecture, detailing the primary components and the interactions between different architectural layers. It also outlines the design decisions made to satisfy the functional and non-functional requirements specified in the RASD.

2.1.1 System Overview

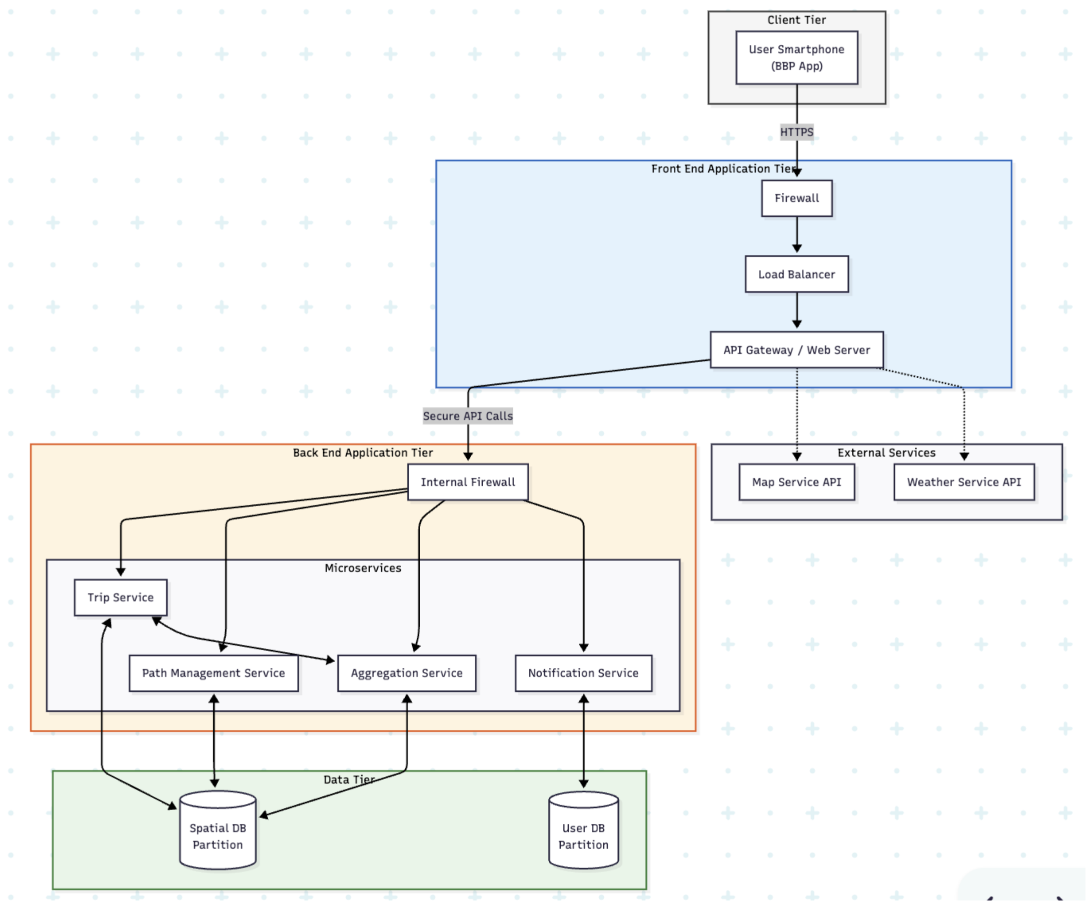
The architectural strategy selected for the BBP platform implements a robust 3-tier architecture, effectively separating the presentation, application processing, and data management concerns.
Presentation Layer: This tier resides on the client side, specifically the Mobile Application running on users' smartphones. It is responsible for the user interface, collecting sensor data (GPS and IMU), and rendering map visualizations.
Application Layer: Located on the server side, this layer is composed of a Microservices Architecture. It includes a Web Server/API Gateway that acts as the entry point, routing requests to specific backend services (e.g., Trip Service, Path Management) that execute the core business logic.
Data Layer: This tier hosts the persistent storage systems. It utilizes a spatial database management system (DBMS) to handle complex road network data, user profiles, and aggregated trip statistics, ensuring data consistency and high availability.
The information flow travels from the mobile client (Presentation) via secure HTTP requests to the Backend (Application), which processes the logic and queries or updates the Data Layer. The processed results are then formatted and returned to the client for display.



2.1.2 Detailed View

Presentation Tier The entry point for the end-user is the BBP Mobile Application. Unlike a web-only interface, this rich client performs significant local processing, including buffering sensor data during offline usage and managing local map rendering. It communicates with the backend via RESTful APIs over HTTPS.
Application Tier - Front End This sub-layer serves as the interface between the external network and the internal system logic. Key components include the Load Balancer and API Gateway. When the mobile app sends a request (e.g., to upload a trip), the Load Balancer

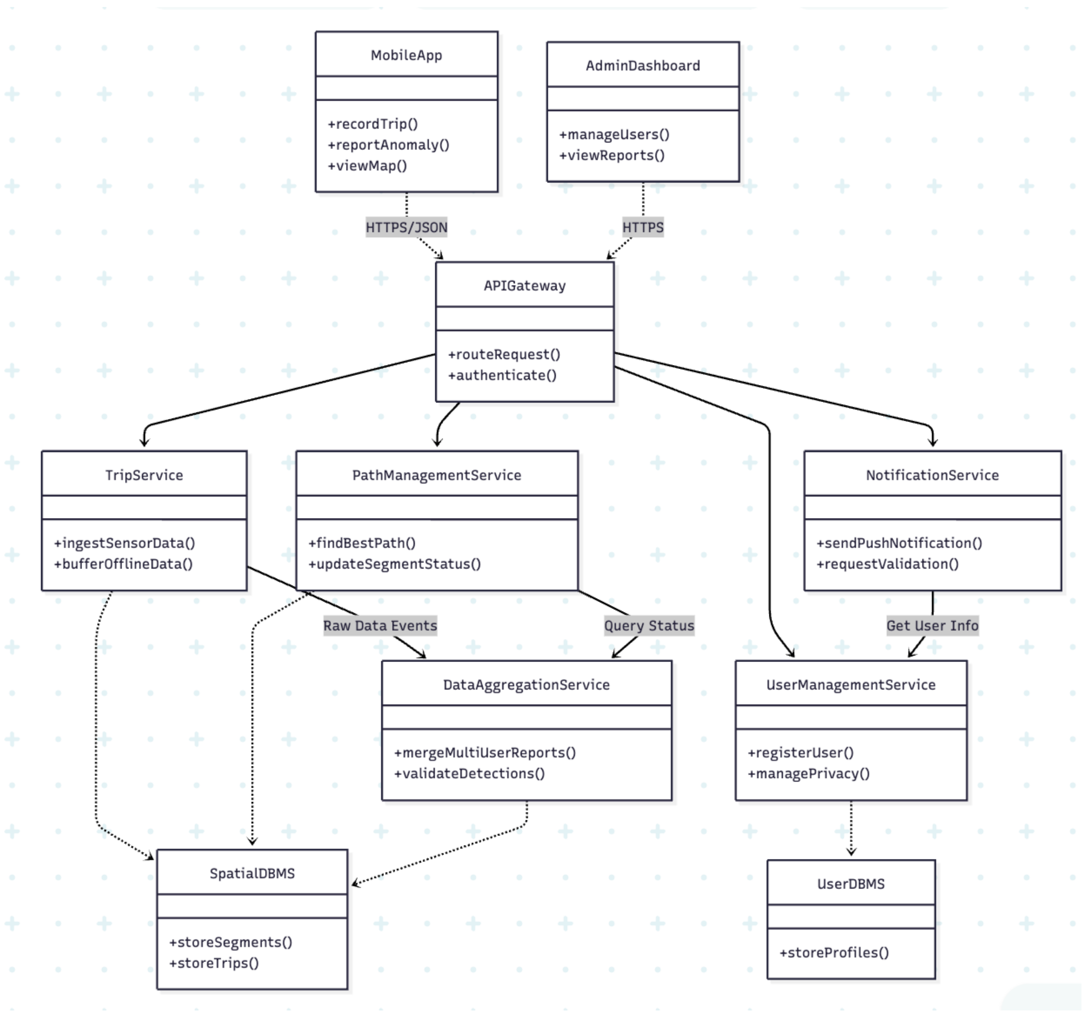
distributes the traffic to an available server instance. The API Gateway then authenticates the request, handles routing, and provides security mechanisms such as rate limiting and SSL termination, shielding the internal microservices from direct external access.
Application Tier - Back End The core logic is distributed across several independent Microservices. Each service focuses on a specific business domain (e.g., aggregating pothole data vs. calculating routes). These services communicate synchronously via REST APIs or asynchronously via message queues, allowing the system to scale specific components (like the Trip Service) independently based on demand.
Data Tier The data persistence layer is managed by a clustered DBMS optimized for geospatial queries (e.g., PostgreSQL with PostGIS). To ensure modularity, each microservice logically owns its specific data schema (e.g., the User Service manages user tables, while the Path Service manages the road graph), preventing direct cross-service database access and maintaining loose coupling.



2.2 Component View

The system is composed of several functional units divided between the Client (Front End) and the Server (Back End), along with necessary external integrations.
Back-end Components
Trip Service This component is responsible for the ingestion of raw trip data. It receives streams of GPS coordinates and IMU sensor readings from the mobile app. It handles the

logic for "Trip Recording" (UC1), calculating metrics such as average speed, distance, and duration. It also manages the temporary buffering of data uploaded after offline sessions.
Path Management Service This service manages the digital representation of the road network. It handles the retrieval of "Best Bike Paths" (UC5) by querying the road graph. It is responsible for calculating the "Path Score" by weighting the distance against the surface quality status. It also processes "Manual Reports" (UC4) submitted by users regarding specific road segments.
Data Aggregation / Merge Service This is the core intelligence of the BBP system. It implements the logic for "Merging Multi-user Path Information" (UC7). It periodically analyzes "Publishable Information"—both manual reports and confirmed automatic detections. Using a voting algorithm that considers data freshness and the number of confirmations, it determines the consolidated status (e.g., Optimal vs. Requires Maintenance) for every segment in the network.
Notification & Validation Service This component manages the "Human-in-the-loop" verification process. When the Trip Service flags a potential anomaly based on sensor data, this service generates the "Detection" event and manages the lifecycle of User Confirmations (UC3). It triggers push notifications to prompt users to validate obstacles and updates the status of detections from "Unverified" to "Publishable."
User Management Service Manages the lifecycle of user accounts (Guest and Registered). It handles profile data, privacy settings (GDPR compliance), and tracks the "reliability score" of users based on their contribution history.
DBMS The persistence layer responsible for storing persistent data, including the Road Graph (Segments/Nodes), User Tables, Trip History, and the Raw/Aggregated Sensor Data.
Front-end Components
Mobile App The interface installed on the cyclist's smartphone. It includes modules for:
Sensor Handler: Background collection of Accelerometer/Gyroscope and GPS data.
Map Renderer: Visualizing the road network and navigation instructions.
UI Controller: Handling user inputs for reporting and confirmation.
Admin Dashboard A web-based interface allowing BBP system administrators to view system health, manage master data, and oversee the aggregation process.
External Services
Map Service API (e.g., Mapbox/Google Maps) Used for geocoding (converting addresses to coordinates), reverse geocoding, and providing base map tiles for visualization.
Weather Service API Provides meteorological data (temperature, wind, precipitation) based on timestamp and location to enrich trip records.



2.3 Deployment View

This section outlines the physical deployment environment of the BBP system.

Mobile Device: The cyclist's smartphone acts as the execution environment for the Client. It requires access to hardware sensors (GPS, IMU) and internet connectivity (4G/5G/Wi-Fi).

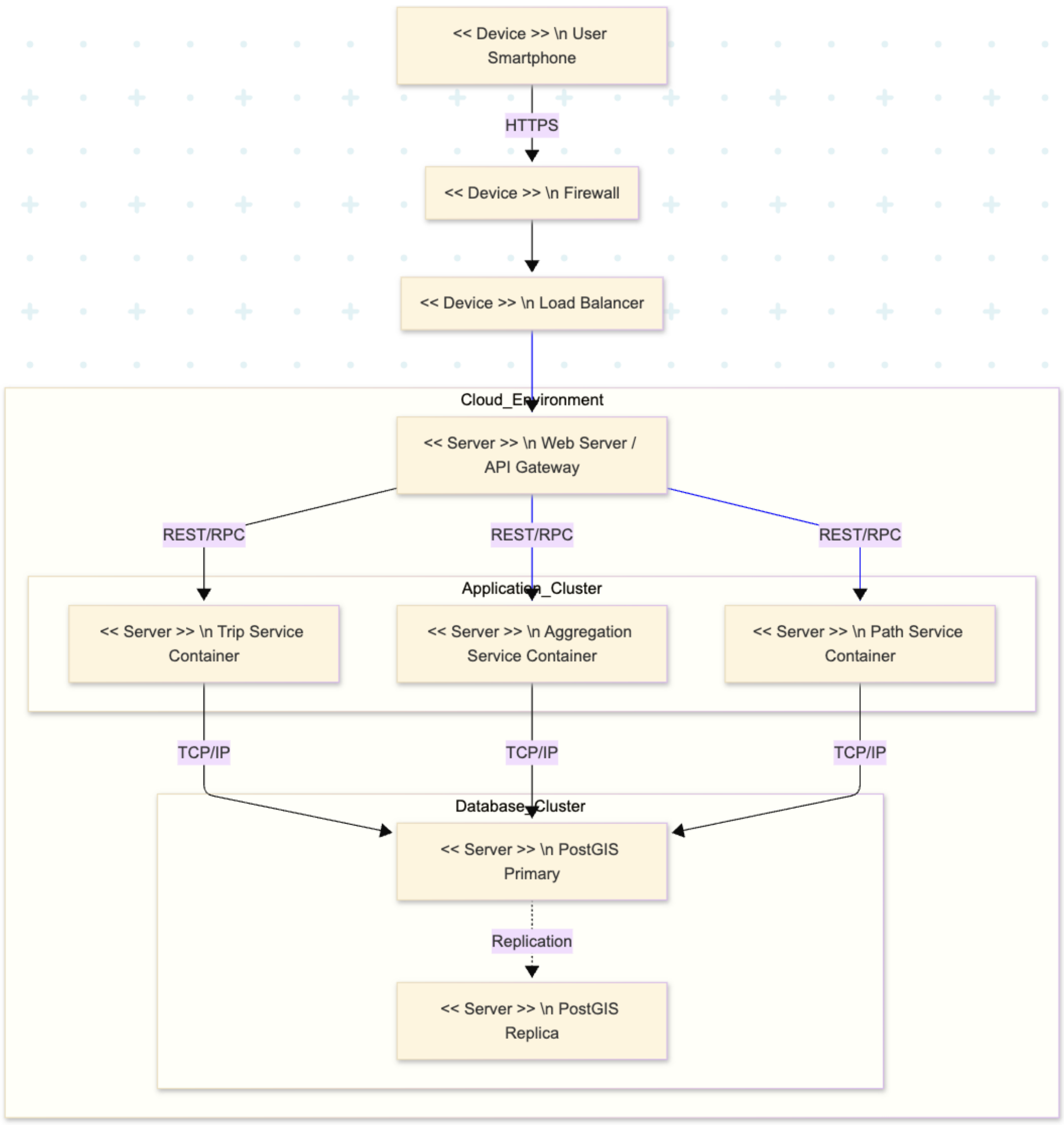
Cloud Infrastructure: The backend is deployed on a scalable cloud provider (e.g., AWS/Azure).

Load Balancer: The entry point for all HTTP traffic, distributing load across available worker nodes.

Application Servers: Containerized environments (e.g., Docker/Kubernetes) hosting the microservices (Trip Service, Aggregation Service, etc.).

Database Cluster: A replicated set of database instances (Primary/Replica) to ensure high availability and data redundancy.

Firewalls: Placed before the Load Balancer and between the Application and Data layers to filter traffic and enforce security rules.


<< Device >> \n User Smartphone
HTTPS
<< Device >> \n Firewall
<< Device >> \n Load Balancer
Cloud_Environment
<< Server >> \n Web Server / API Gateway
REST/RPC
REST/RPC
REST/RPC
Application_Cluster
<< Server >> \n Trip Service Container
<< Server >> \n Aggregation Service Container
<< Server >> \n Path Service Container
TCP/IP
TCP/IP
TCP/IP
Database_Cluster
<< Server >> \n PostGIS Primary
Replication
<< Server >> \n PostGIS Replica

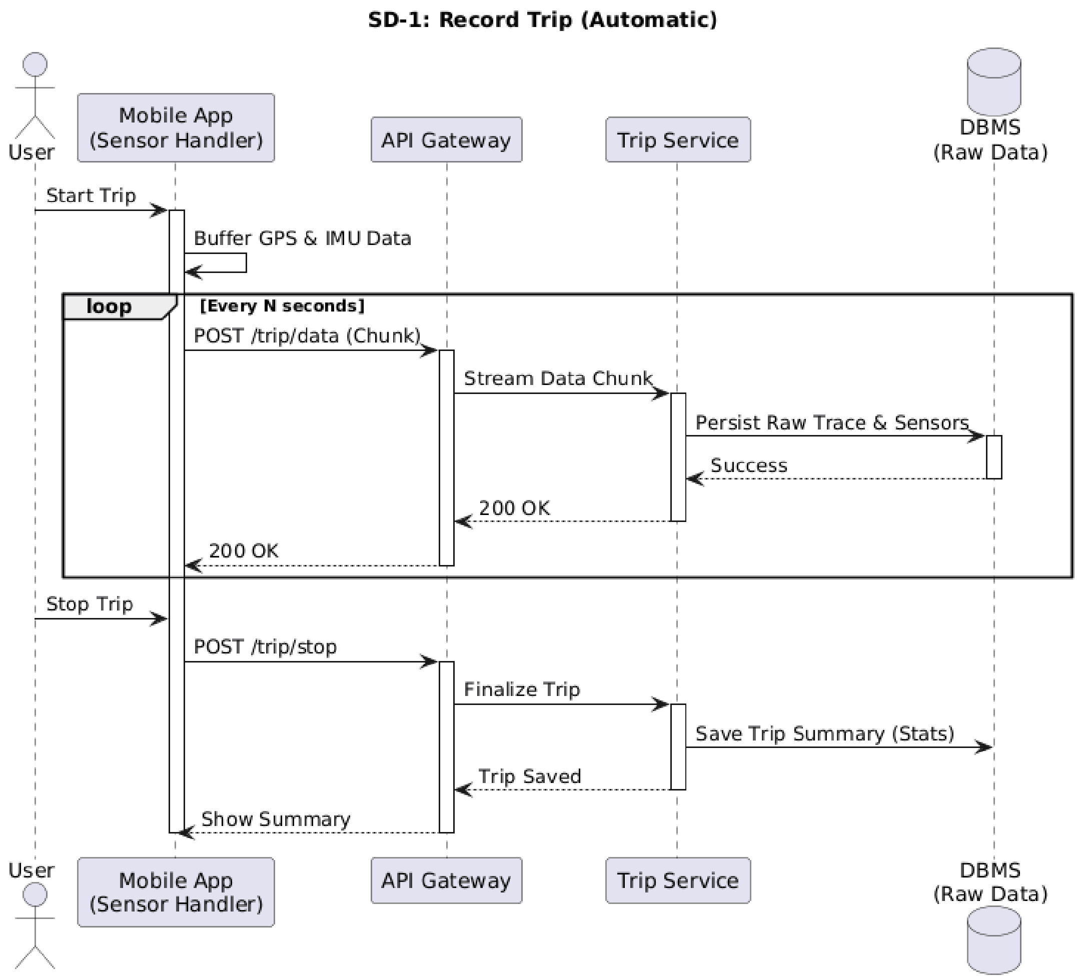
2.4 Runtime View

This section describes the dynamic behavior of the system by illustrating the message flow between various components using UML sequence diagrams. These scenarios cover critical business processes such as data collection, manual verification, backend aggregation, and path computation, demonstrating how the microservices defined in Section 2.2 (e.g., Trip Service, Path Service, Aggregation Service) collaborate to fulfill the requirements.

2.4.1 Scenario 1: Record Trip (Automatic Recording)

This scenario illustrates how the Trip Service handles high-frequency sensor data streams generated during a user's ride. To ensure application performance and responsiveness, the system adopts an asynchronous write strategy: raw data is quickly persisted to the database first, while complex processing is deferred.

Reference: [RASD: UC1 Record Trip]

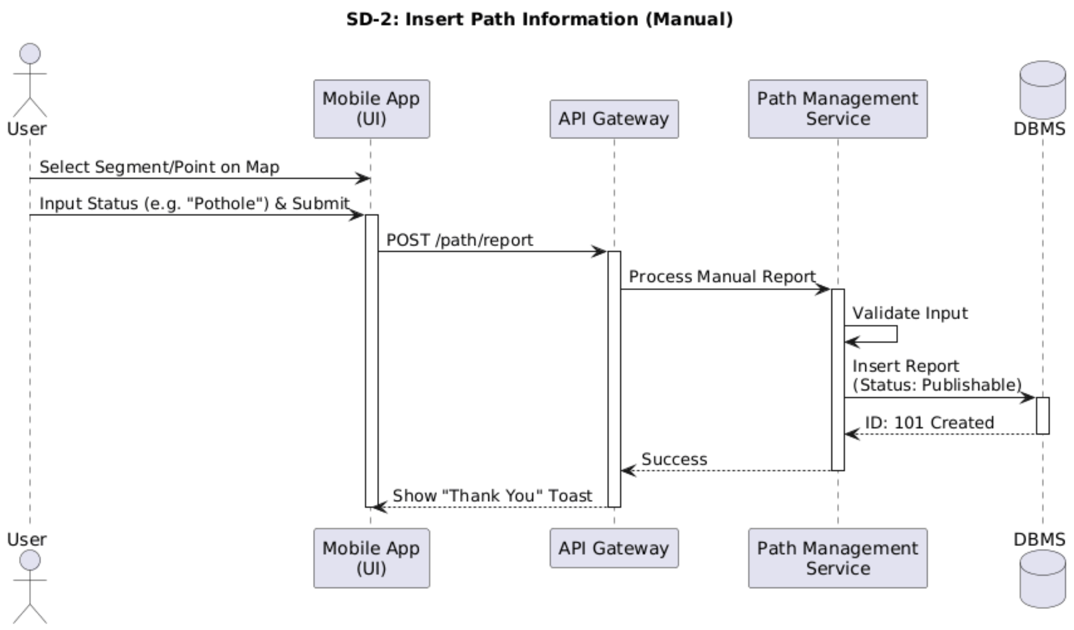


[Figure 2.1 Sequence Diagram for Record Trip (Automatic Recording)]

2.4.2 Scenario 2: Insert Path Information (Manual Reporting)

This scenario describes the workflow for users manually reporting road conditions (e.g., potholes). Unlike automatically detected anomalies, manually submitted reports are considered "Publishable" by default and are immediately eligible for the aggregation process.

Reference: [RASD: UC4 Insert Manual Path Information]

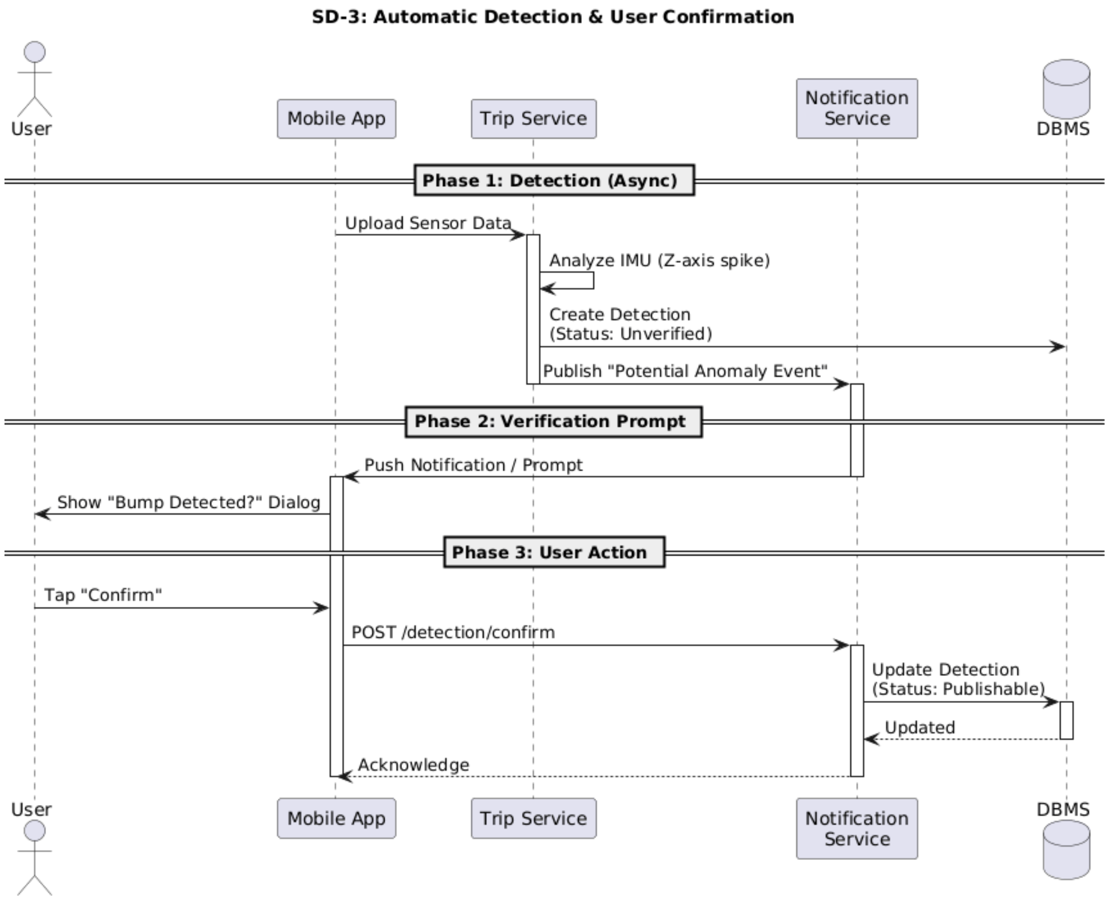


[Figure 2.2 Sequence Diagram for Insert Path Information]

2.4.3 Scenario 3: Automatic Detection & User Confirmation

This interaction represents a complex "Human-in-the-loop" process. To avoid blocking the critical trip recording thread, anomaly detection and user notification occur asynchronously. The Notification Service is responsible for engaging the user to verify the detected obstacle, ensuring that only confirmed data enters the system as publishable information.

Reference: [RASD: UC2 Automatic Obstacle Detection, UC3 Confirm/Correct]

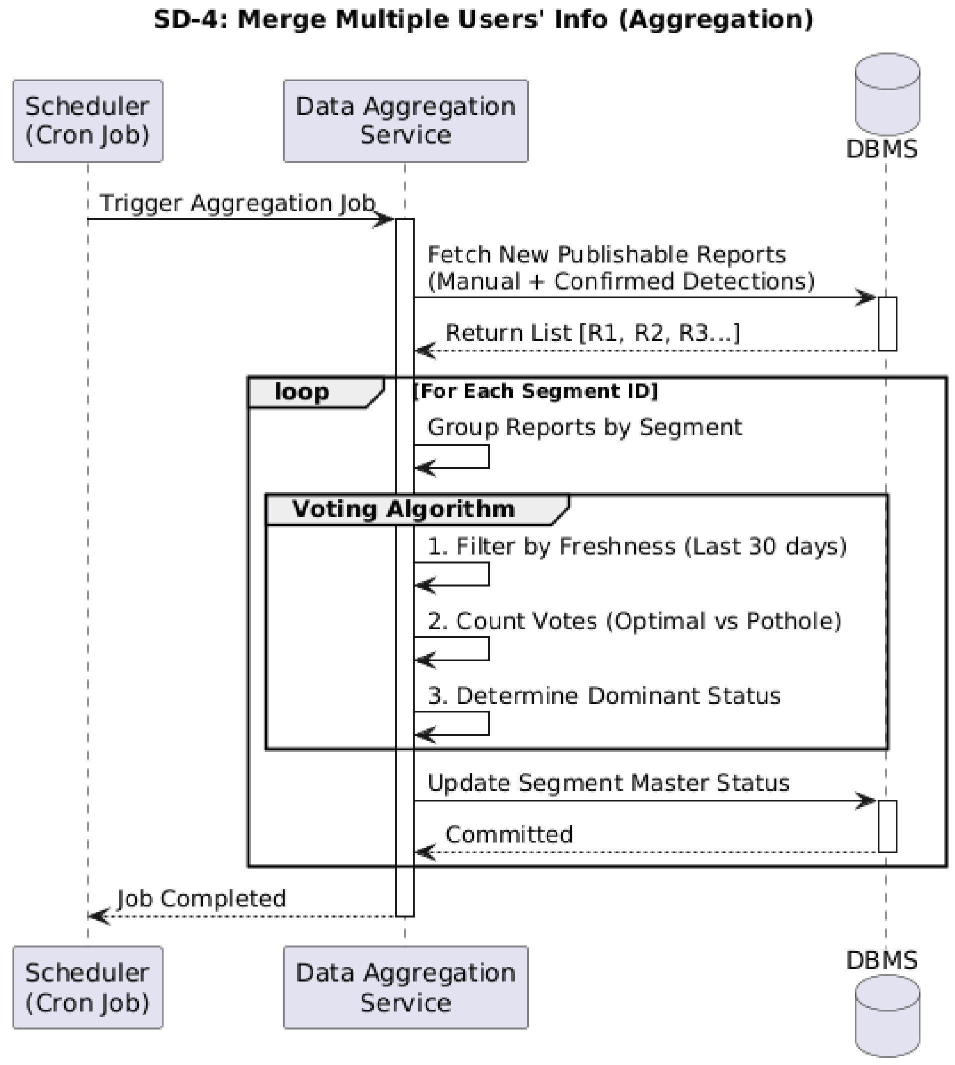

SD-3: Automatic Detection & User Confirmation
User
Mobile App
Trip Service
Notification Service
DBMS
Phase 1: Detection (Async)
Upload Sensor Data
Analyze IMU (Z-axis spike)
Create Detection (Status: Unverified)
Publish "Potential Anomaly Event"
Phase 2: Verification Prompt
Push Notification / Prompt
Show "Bump Detected?" Dialog
Phase 3: User Action
Tap "Confirm"
POST /detection/confirm
Update Detection (Status: Publishable)
Updated
Acknowledge

[Figure 2.3 Sequence Diagram for Automatic Detection & User Confirmation]

2.4.4 Scenario 4: Merge Multiple Users' Information (Data Aggregation)

This diagram demonstrates the core logic of the BBP system: how the backend periodically processes "Publishable Data" collected from multiple users. The **Data Aggregation Service** executes a voting algorithm weighted by data freshness (as defined in the RASD) to determine the unified status of a road segment.

Reference: [RASD: UC7 Merge Multi-user Path Information]

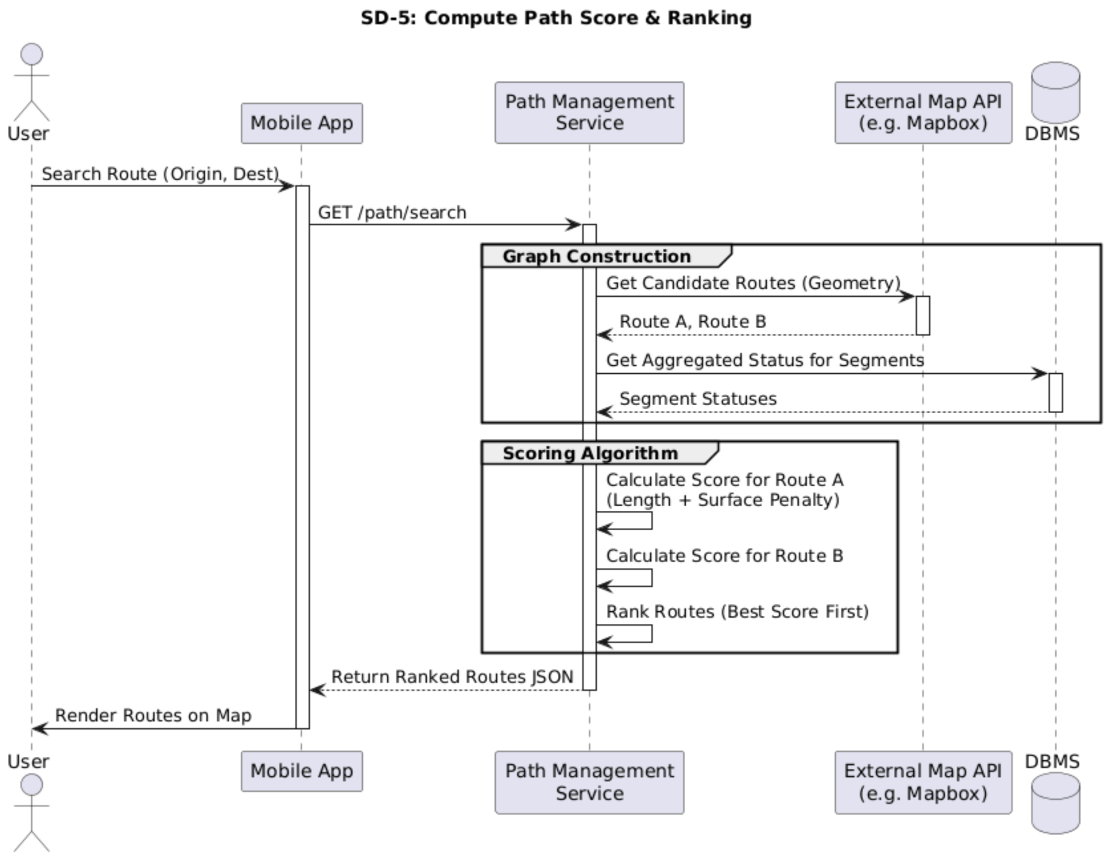

SD-4: Merge Multiple Users' Info (Aggregation)
Scheduler (Cron Job)
Data Aggregation Service
DBMS
Trigger Aggregation Job
Fetch New Publishable Reports (Manual + Confirmed Detections)
Return List [R1, R2, R3...]
loop [For Each Segment ID]
Group Reports by Segment
Voting Algorithm
1. Filter by Freshness (Last 30 days)
2. Count Votes (Optimal vs Pothole)
3. Determine Dominant Status
Update Segment Master Status
Committed
Job Completed

[Figure 2.4 Sequence Diagram for Data Aggregation]

2.4.5 Scenario 5: Compute Path Score & Ranking

This scenario illustrates the "Read" path for route planning. The Path Management Service combines static geographical data from the External Map API with the dynamic "Aggregated Segment Status" stored in the BBP database. This allows the system to calculate a "Path Score" that balances distance with road surface quality, providing the user with the safest and smoothest route options.

Reference: [RASD: UC5 View Best Bike Path]


SD-5: Compute Path Score & Ranking
User
Mobile App
Path Management Service
External Map API (e.g. Mapbox)
DBMS
Search Route (Origin, Dest)
GET /path/search
Graph Construction
Get Candidate Routes (Geometry)
Route A, Route B
Get Aggregated Status for Segments
Segment Statuses
Scoring Algorithm
Calculate Score for Route A (Length + Surface Penalty)
Calculate Score for Route B
Rank Routes (Best Score First)
Return Ranked Routes JSON
Render Routes on Map

[Figure 2.5 Sequence Diagram for Path Computation]

2.5 Component Interfaces

This section summarizes the primary APIs exposed by the BBP system components. The system utilizes RESTful APIs to communicate via JSON over HTTPS. Below is a definition of the key endpoints provided by each microservice.

2.5.1 Trip Service API (Manages trip recording and raw data ingestion)

1.POST /trip/start(userId, timestamp, startLocation): Initiates a new trip session.
2.POST /trip/data(tripId, sensorDataChunk): Uploads buffered GPS and IMU sensor data streams.
3.POST /trip/stop(tripId, endLocation, totalMetrics): Finalizes the trip and saves summary statistics.
4.GET /trip/history(userId): Retrieves the list of past trips for a user.

2.5.2Path Management Service API (Manages routing and manual reporting)

1.GET /path/search(origin, destination, preferences): Computes and returns a ranked list of "Best Bike Paths" based on length and surface quality.
2.POST /path/report(userId, segmentId, status, description): Submits a manual report for a specific road segment.

3.GET /path/segment_status(segmentId): Retrieves the current aggregated quality status of a segment.

2.5.3 Notification & Validation Service API （Manages user interactions and confirmations）

GET /detection/pending(userId): Retrieves a list of unverified anomalies waiting for user action.
1.1POST /detection/confirm(detectionId, userAction): Submits the user's validation (Confirm/Reject/NotSure) for a detected anomaly.

2.5.4 Data Aggregation Service API (Manages internal merging logic)

POST /aggregation/trigger(): (Internal/Admin) Manually triggers the batch aggregation process.
1.GET /aggregation/status(segmentId): Returns the detailed voting breakdown (freshness/confidence) for a segment.

2.5.5 User Ma ment Service API

1.POST /auth/login(credentials): Authenticates a user and returns a JWT token.
2.POST /auth/register(userData): Registers a new Guest or Regular user.
3.PUT /user/settings(userId, privacyConfig): Updates user preferences and GDPR privacy settings.

2.5.6 External Interfaces (Consumed)

MapService.getRoute(origin, dest): Used for geocoding and base route geometry.
1.WeatherService.getConditions(lat, lon, time): Used to enrich trip history with meteorological data.

2.6 Selected Architectural Styles and Patterns

Client-Server Architecture BBP adopts a strict Client-Server model. The Mobile App (Client) is responsible for data acquisition (sensors) and presentation, while the Backend (Server) handles heavy computational tasks like pathfinding algorithms and statistical aggregation. This separation ensures that the mobile device's battery and processing power are conserved, delegating resource-intensive operations to the cloud.
Layered & Microservices Architecture The backend is structured as a Layered Architecture (Controller, Service, Repository layers within code) implemented via Microservices.
Why Microservices? The "Trip Recording" feature requires high-throughput to write operations (streaming sensor data), while the "Data Aggregation" is a periodic, compute-heavy batch process. Decoupling these into separate services allows BBP to scale the Trip Service independently during peak commuting hours without provisioning unnecessary resources for the Aggregation Service.

To handle the asynchronous nature of anomaly detection, BBP uses a Pub/Sub pattern. When the Trip Service detects a sensor spike, it publishes an event. The Notification Service subscribes to these events to trigger user validation, ensuring that the trip recording process is not blocked by the validation logic.

2.7 Other Design Decisions

Proxy as Intermediary a Reverse Proxy / API Gateway is placed between the mobile clients and the backend services. This component centralizes cross-cutting concerns such as SSL termination, Authentication token verification, and Request Rate Limiting. It simplifies the client's logic, as the app only needs to communicate with a single endpoint URL regardless of how many microservices operate behind the scenes.
Offline-First Strategy Given the urban mobility context, cyclists may pass through areas with poor network coverage. The system is designed with an "Offline-First" approach. The Mobile App creates a local buffer (SQLite/Realm) to store trip data and manual reports. A synchronization manager in the app monitors network connectivity and automatically pushes buffered data to the Trip Service once the connection is restored, ensuring no data loss.
GDPR & Privacy Handling To comply with regulatory constraints, the architecture enforces privacy by design. Location data is anonymized at the storage level. The User Management Service includes specific logic to obfuscate the precise start and end points of trips (often home or work locations) before the data is made available to the Aggregation Service, ensuring that community maps cannot be reverse engineered to track individuals.

3. User Interface Design

The system provides two main user interfaces:
Mobile Application (for cyclists)
Admin Web Dashboard (for administrators/operators)

3.1 Overall UI Principles (Realistic to Implement)

Map-first UI: Most features are accessed and visualized on a map (road quality, reporting, route viewing).
Minimal interaction while riding: During cycling, the UI shows only essential information and avoids complex user input.
Offline-first behavior: If network is unavailable, the app stores data locally and synchronizes it when connectivity is restored.
Basic privacy-by-design: The system avoids exposing highly precise start/end points in public views.

3.2 Mobile App – Main Screens

1. Authentication / Guest Access

Welcome screen: Login / Register / Continue as Guest
After login: store user session info (used to upload trips and reports)

2.Home Map (Core Screen)
Map view showing current location
Road segments are color-coded by status (e.g., Good / Medium / Bad)
Three primary actions:
Start Trip
Report Road Condition / Obstacle
Trip History

3.Trip Recording (Basic Tracking)
Tap "Start Trip"
During ride: show timer, distance, speed
On "Stop": show trip summary (total distance, total duration)

4. Manual Reporting (Simple Input)
User taps a location/segment on the map → selects a category → submits.
A minimal, feasible set of categories:
Road condition: Good / Medium / Bad
Obstacle: Pothole / Construction / Other
After submission, show a confirmation message (e.g., "Report submitted").

5.Automatic Detection + User Confirmation (Minimal Feasible Version)
If the app detects a strong bump event during a ride (optional feature), it shows a short prompt:
"Did you encounter a pothole/bump just now?"
Confirm / Reject / Not sure
If automatic detection is not feasible within time constraints, the first release can support manual reporting only, and detection can be listed under "Future Improvements".

6.Route Planning (Simple and Achievable)
User enters origin and destination (or uses current location as origin).
The app shows 1–3 candidate routes.
Each route displays: distance + a coarse "quality summary" (based on segment colors/scores).
Implementation note: the system can initially rely on an external map routing API to get candidate routes, then apply a simple ranking using the aggregated road quality score. If needed, ranking can be added later.

7. Trip History
List view: date, distance, duration for each trip

Detail view: map polyline (optional) + segment status overlay (optional)

8. Settings
Data sharing consent (toggle on/off)
Privacy settings (simple statement: precise start/end points are not publicly displayed)

3.3 Admin Web Dashboard

The Admin Dashboard focuses on three practical modules:
View Segment Status
Map view with segment colors (Good / Medium / Bad)
View Reports
List of submitted reports (time, location, type)
Basic filters (e.g., by time)
Trigger Status Update (Aggregation)
A button to run a recalculation that updates segment status based on recent reports.

4. Requirements Traceability

This section explains how key requirements/use cases are mapped to concrete modules/screens.

Requirement / Use Case	Implementation Mapping (Modules / Screens)
Trip recording	Mobile App: Start Trip / Stop Trip / Trip History
Manual reporting	Mobile App: map selection → category selection → submit
Best route viewing	Mobile App: Route Planning screen (origin/destination → route results)
Multi-user data merge	Backend aggregation job computes segment status from multiple users’ reports
Automatic detection & confirmation (optional)	In-app confirmation prompt; can be deferred if not feasible

5. Implementation, Integration and Test Plan

5.1 Implementation Order

Implement core Mobile App UI first
Home Map, Trip Recording, Trip Summary, Trip History (can be stubbed without backend first)
Implement minimal backend APIs

Trip upload (start/stop)
Report submission (location/segment + category)
Integrate Mobile App with backend
Start/stop trip → backend persists trip
Report submission → backend persists report
Trip history → backend returns stored trips
Implement Aggregation (Segment Status Update)
A simple rule-based aggregation:
If most reports indicate “Bad” → segment becomes “Bad”
If most reports indicate “Medium” → segment becomes “Medium”
Otherwise → “Good”
Admin dashboard provides a manual “Run aggregation” button (simplest operational approach)

5.2 Integration Plan

Run backend services and database locally (or in a dev environment)
Mobile app calls backend REST APIs
Admin dashboard calls backend REST APIs

5.3 Test Plan (Achievable Scope)

Unit tests (basic)
Verify report persistence
Verify aggregation logic outputs correct segment status
End-to-End tests (main focus)
Start trip → stop trip → trip appears in history
Submit a pothole report → report appears in admin dashboard
Submit multiple “Bad” reports for same segment → segment status becomes “Bad” on the map
Route planning returns a route (ranking optional for first version)

6. Effort Spent

The following table shows the time allocation of each group member across different stages of the project:

Group Member	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5	Total Hours
Kaifei Xu	3	14	9	3	6	35
Shinuo Yan	1	3	12	9	10	35
Yanglin Hu	1	6	13	6	10	36