POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 - PROJECT 2025-2026

Best Bike Paths (BBP)

Implementation and Testing Deliverable

Authors:

Rajatkant Nayak (11144180)
Shashi Bhushan XXX (11111318)

January 30, 2026

Contents

1 Introduction

1.1 Purpose

This Implementation and Test Document (ITD) provides comprehensive documentation for the Best Bike Paths mobile application. The document details the technical implementation, architectural decisions, testing procedures, and installation instructions necessary for deploying and evaluating the software.

1.2 Scope

Best Bike Paths is a cross-platform mobile application designed to enhance cyclist safety through:

- Real-time pothole and road anomaly detection using smartphone accelerometer sensors and machine learning
- Community-driven hazard reporting and verification with a sophisticated voting system
- Intelligent route planning that considers road surface quality and safety scores
- Comprehensive ride tracking with offline support and background operation
- Weather-aware cycling assistance with real-time alerts and safety recommendations

The application targets urban cyclists who need reliable information about road conditions to plan safer routes and contribute to community safety data.

1.3 Definitions, Acronyms, and Abbreviations

The following terms and acronyms are used throughout the implementation code and this document.

Term	Definition
Anomaly	Any road hazard including potholes, broken glass, cobblestones, flooding, etc.
ML	Machine Learning
BaaS	Backend as a Service (Supabase)
GPS	Global Positioning System
OSRM	Open Source Routing Machine
RLS	Row Level Security (Supabase security feature)
ITD	Implementation and Test Document
RASD	Requirements Analysis and Specification Document
DD	Design Document

Table 1: Definitions and Acronyms

1.4 Revision History

Version	Date	Description	Author
1.0	January 30, 2026	Initial release of the ITD.	Rajatkant Nayak, Shashi Bhushan XXX

Table 2: Revision History

1.5 Reference Documents

The implementation is based on the requirements and specifications defined in the following documents:

- BBP RASD v1.0: Requirements Analysis and Specification Document, December 22, 2025.
- BBP DD v1.0: Design Document, January 25, 2026.
- Assignment Specification: Software Engineering 2 – Project Description AY 2025–2026, Politecnico di Milano.
- Flutter Documentation: Official API reference for the Dart language and Flutter SDK.

1.6 Document Structure

The remainder of this document is organized as follows:

- **Section 2 (Product Functions and Implementation):** Details the core functional requirements including user registration, real-time GPS tracking, and the specific algorithms used for pothole detection and route planning.
- **Section 3 (System Development Frameworks):** Describes the development environment, including the programming languages (Dart, Python), frameworks (Flutter), and external APIs utilized in the project.
- **Section 4 (Source Code Structure):** Provides an overview of the project directory, architectural patterns, and descriptions of key source files and database schemas.
- **Section 5 (Testing Information):** Outlines the testing strategy, covering unit testing, machine learning validation, system testing results, and performance metrics.
- **Section 6 (Installation Instructions):** Offers a step-by-step guide for setting up the development environment, building the application, and installing the APK on Android devices.
- **Section 7 (Effort Spent):** Break down the time allocation by team member and component, including document preparation time.
- **Section 8 (References):** Lists the official documentation, research papers, and software libraries referenced throughout the project.

2 Product Functions and Implementation

This section details the specific functions implemented in the final release of the **Best Bike Paths (BBP)** system. It describes how the high-level requirements defined in the RASD were translated into working software modules, including refinements and enhancements discovered during implementation.

2.1 Requirements Traceability and Evolution

During implementation, certain requirements from RASD v1 were refined based on field testing, user feedback, and research from the SimRa Berlin dataset. The following matrix documents the implementation status and any deviations from the original specification.

Table 3: Requirements Implementation Matrix with RASD Traceability

Req ID	Description	RASD Ref.	Status	Notes
REQ-001	User Registration and Authentication	UC1	✓	As specified
REQ-002	Real-time GPS Tracking	UC2, H2	✓	As specified
REQ-003	Automatic Pothole Detection	UC4, R3	✓	**Enhanced**†
REQ-004	Manual Hazard Reporting	UC3	✓	As specified
REQ-005	Community Verification System	UC5, R4	✓	**Enhanced**†
REQ-006	Route Planning with Safety Scores	UC7	✓	As specified
REQ-007	Ride History and Statistics	§2.2.2	✓	As specified
REQ-008	Offline Operation Support	DC6	✓	As specified
REQ-009	Background Ride Tracking	–	✓	**New***
REQ-010	Weather Alerts	D.D.3	✓	**Extended**†
REQ-011	Water Fountain Locations	–	✓	**New***
REQ-012	Cobblestone Avoidance	Scenario 2.1	✓	**New***

† Enhanced: Implementation exceeds original RASD specification
* New: Feature added during implementation phase, not in original RASD

2.1.1 Summary of Requirement Evolution

The following table summarizes key parameters that evolved from RASD v1 specification to final implementation:

Table 4: RASD v1 vs. Implementation: Key Parameter Changes

Parameter	RASD v1 Spec	Implementation	Justification
Detection Threshold	$>$ 2.0 G (fixed)	1.8 G (adaptive)	Better sensitivity; SimRa research
Detection Method	Threshold only	Threshold + ML	Reduced false positives
Speed Range	5–35 km/h	4–50 km/h	Extended for pro cyclists
Verification	Binary confirm/reject	Democratic voting system	Better data quality
Gyroscope Use	Not specified	100 Hz for orientation	Improved accuracy
Jerk Analysis	Not specified	5.0 G/s threshold	Key discriminator added

2.2 User Registration and Authentication (REQ-001)

RASD Reference: UC1 (User Registration and Login), §2.2.1
Implementation Status: Implemented as specified.
The system implements the dual-access model defined in RASD to lower entry barriers while securing user data.

- **Authentication Provider:** Implemented using `Supabase Auth`. The system supports email/-password registration and handles session persistence via JSON Web Tokens (JWT) stored in the device's secure storage (Keychain/Keystore).
- **Features:**
  - Secure user registration with email verification
  - Password-based login with session persistence
  - Guest mode for users who want to explore without an account (as per RASD UC1-A2)
  - Automatic session management and token refresh

Technical Implementation:

```
// Authentication Flow - implements UC1
await Supabase.initialize(
  url: SupabaseConstants.url,
  anonKey: SupabaseConstants.anonKey,
);
```

Listing 1: Authentication Initialization in main.dart

2.3 Real-time GPS Tracking (REQ-002)

RASD Reference: UC2 (Start Ride Session), H2 (GPS Receiver), PR4 (GPS Accuracy)
Implementation Status: Implemented as specified.

- **GPS Tracking:** The `LocationService` utilizes the `geolocator` plugin. It implements a "Sentinel Filter" that discards any GPS packet with a horizontal accuracy radius greater than 10 meters, strictly adhering to RASD requirement PR4.
- **Sampling Rate:** GPS data captured at 1 Hz as specified in RASD H2.
- **Signal Loss Handling:** Implements the spatial blackout recovery logic defined in RASD §3.4.4 (Flow 3).

Table 5: GPS Implementation vs. RASD Specification

RASD Requirement	Specification	Implementation
H2: Sampling Rate	1 Hz	1 Hz ✓
PR4: Horizontal Accuracy	$\leq$ 10 meters	$\leq$ 10 meters ✓
PR4.1: Signal Recovery	$<$ 500 ms	$<$ 500 ms ✓

2.4 Automatic Pothole Detection (REQ-003)

RASD Reference: UC4 (Automated Anomaly Detection), R3, PR5, H3
Implementation Status: Enhanced beyond original specification.

2.4.1 Evolution from RASD Specification

The original RASD (UC4) specified a simple threshold-based detection with:

- Fixed threshold: $G_z > 2.0g$
- Speed range: 5–35 km/h
- No jerk analysis
- No machine learning component

Based on field testing and research from the **SimRa Berlin dataset**, the implementation was enhanced to include:

1. **Threshold-Based Detection**: Real-time, rule-based detection using advanced signal processing
2. **ML-Based Detection** [NEW]: Machine Learning Random Forest model trained on labeled cycling data

Justification for Enhancement:

- Field testing revealed the 2.0G threshold missed moderate potholes detectable by experienced cyclists
- Jerk (rate of change) proved to be a critical discriminator between potholes and normal bumps
- ML-based detection significantly reduces false positives (phone drops, curb jumps) as described in RASD Scenario 3
- Professional cyclists (like persona "Raju") ride at speeds up to 45 km/h, requiring extended speed range

2.4.2 Threshold-Based Detection Algorithm

Algorithm Overview The threshold-based approach uses **real-time signal processing** with adaptive thresholds to detect sudden vertical impacts characteristic of potholes. The algorithm executes in a **separate Dart Isolate** for optimal performance, satisfying RASD requirement NFR1 (processing within 100ms).

Table 6: Sensor Configuration: RASD vs. Implementation

Parameter	RASD Spec	Implementation	Compliance
Accelerometer Rate	$\geq$ 50 Hz (PR5)	50 Hz	✓ Compliant
Gyroscope Rate	Not specified	100 Hz	[Enhanced]
Window Size	Not specified	50 samples ($\sim$1s)	[New]
Cooldown Period	Not specified	1500 ms	[New]

Sensor Configuration

Sensor Selection Strategy The system employs a fallback mechanism for sensor availability:

1. **Primary**: UserAccelerometer (linear acceleration with gravity removed)
2. **Fallback**: Basic Accelerometer (raw readings including gravity)

If the UserAccelerometer provides no data within 2 seconds, the system automatically switches to the basic accelerometer with gravity compensation using quaternion-based orientation tracking.

Signal Processing Pipeline **Step 1: Orientation Compensation (Quaternion-Based)** [NEW - Not in RASD]

The algorithm maintains device orientation using gyroscope integration with quaternion representation:

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \tag{1}$$

Quaternion Update Equations:

$$q'_0 = q_0 + \frac{\Delta t}{2} \cdot (-q_1 \cdot \omega_x - q_2 \cdot \omega_y - q_3 \cdot \omega_z) \tag{2}$$

$$q'_1 = q_1 + \frac{\Delta t}{2} \cdot (q_0 \cdot \omega_x + q_2 \cdot \omega_z - q_3 \cdot \omega_y) \tag{3}$$

$$q'_2 = q_2 + \frac{\Delta t}{2} \cdot (q_0 \cdot \omega_y - q_1 \cdot \omega_z + q_3 \cdot \omega_x) \tag{4}$$

$$q'_3 = q_3 + \frac{\Delta t}{2} \cdot (q_0 \cdot \omega_z + q_1 \cdot \omega_y - q_2 \cdot \omega_x) \tag{5}$$

Where:

- $\omega_x, \omega_y, \omega_z$ = gyroscope readings (rad/s)
- Δt = time delta (seconds)

Quaternion Normalization:

$$\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{6}$$

$$\mathbf{q}_{normalized} = \frac{\mathbf{q}}{\|\mathbf{q}\|} \tag{7}$$

Complementary Filter Correction:
To correct gyroscope drift, a complementary filter uses accelerometer data:

$$\mathbf{g}_{device} = \text{rotateVector}(\mathbf{q}, [0, 0, 1]^T) \tag{8}$$

$$\mathbf{e} = \mathbf{g}_{device} \times \mathbf{a}_{normalized} \tag{9}$$

$$\boldsymbol{\omega}_{adjusted} = \boldsymbol{\omega} + k_c \cdot \mathbf{e} \tag{10}$$

Where $k_c = 0.08$ is the correction gain.

Step 2: Coordinate Transformation

Transform acceleration from device frame to world frame using quaternion rotation:

$$\mathbf{a}_{world} = \mathbf{q} \otimes \mathbf{a}_{device} \otimes \mathbf{q}^* \tag{11}$$

For the fallback accelerometer, remove gravity component:

$$\mathbf{a}_{linear} = \mathbf{a}_{world} - [0, 0, g]^T \tag{12}$$

Convert to G-force:

$$z_{raw} = \frac{a_{linear,z}}{g} \tag{13}$$

Where $g = 9.8$ m/s^2.

Step 3: Bandpass Filtering [NEW - Not in RASD]

A two-stage IIR filter isolates pothole-characteristic frequencies:

High-Pass Filter (removes body sway, cutoff: $f_h = 0.8$ Hz):

$$RC_h = \frac{1}{2\pi f_h}, \quad \alpha_h = \frac{RC_h}{RC_h + \Delta t} \tag{14}$$

$$y_{high}[n] = \alpha_h \cdot (y_{high}[n-1] + x[n] - x[n-1]) \tag{15}$$

Low-Pass Filter (removes high-frequency noise, cutoff: $f_l = 15$ Hz):

$$RC_l = \frac{1}{2\pi f_l}, \quad \alpha_l = \frac{\Delta t}{RC_l + \Delta t} \tag{16}$$

$$y_{low}[n] = y_{low}[n-1] + \alpha_l \cdot (y_{high}[n] - y_{low}[n-1]) \tag{17}$$

Result: Filtered Z-force signal in the **0.8 Hz – 15 Hz** frequency band (pothole impact frequencies).

Step 4: Jerk Calculation [NEW - Not in RASD]

Jerk measures the rate of acceleration change:

$$\text{jerk}[n] = \frac{|z_{force}[n] - z_{force}[n-1]|}{\Delta t} \quad \text{[G/s]} \tag{18}$$

Potholes cause sudden acceleration changes with jerk values of 4–10 G/s.

Adaptive Threshold Calculation Unlike the fixed 2.0G threshold in RASD UC4, the implementation uses an **adaptive threshold** that adjusts to riding conditions:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} z_i, \quad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(z_i - \mu)^2} \tag{19}$$

$$\theta_{adaptive} = \mu + k_\sigma \cdot \sigma \tag{20}$$

$$\theta_{final} = \max(\theta_{adaptive},\ 0.8 \times \theta_{base}) \tag{21}$$

Where:

- $N = 50$ (window size)
- $k_\sigma = 3.0$ (standard deviation multiplier)
- $\theta_{base} = 1.8$ G (base threshold, reduced from RASD's 2.0G based on field testing)

Multi-Factor Confidence Scoring [NEW - Not in RASD]

The confidence score combines multiple factors:

$$S_z = \text{clamp}\left(\frac{z_{force} - \theta}{\theta}, 0, 1\right) \tag{22}$$

$$S_{jerk} = \text{clamp}\left(\frac{\text{jerk}}{\theta_{jerk}}, 0, 1\right) \tag{23}$$

$$S_{speed} = \text{clamp}\left(\frac{v - v_{min}}{v_{max} - v_{min}}, 0, 1\right) \tag{24}$$

$$C = 0.50 \cdot S_z + 0.35 \cdot S_{jerk} + 0.15 \cdot S_{speed} \tag{25}$$

Where:

- $\theta_{jerk} = 5.0$ G/s
- $v_{min} = 4$ km/h (reduced from RASD's 5 km/h)
- $v_{max} = 50$ km/h (increased from RASD's 35 km/h)

Algorithm 1 Threshold-Based Pothole Detection (Enhanced from RASD UC4)

Require: Filtered z_{force}, jerk, confidence C, threshold θ
Ensure: Detection decision

```
if C ≥ 0.55 then
    return POTHOLE DETECTED
else if z_force ≥ θ AND jerk ≥ 0.8 × θ_jerk then
    return POTHOLE DETECTED
else
    return NO DETECTION
end if
```

Detection Decision Logic

Table 7: Speed Validation: RASD vs. Implementation

Condition	RASD (D.A.3)	Implementation	Action
Stationary/Walking	$v < 5$ km/h	$v < 4$ km/h	Skip processing
Normal Cycling	5–35 km/h	4–50 km/h	Process normally
Not Cycling	$v > 35$ km/h	$v > 50$ km/h	Skip processing

Speed Validation

Table 8: Threshold Parameters: RASD vs. Implementation

Parameter	RASD UC4	Normal Mode	Test Mode
Z-Impact Threshold	2.0 G (fixed)	1.8 G (adaptive)	1.26 G
Jerk Threshold	Not specified	5.0 G/s	2.5 G/s
Confidence Threshold	Not specified	0.55	0.275
Cooldown	Not specified	1500 ms	1500 ms

Detection Thresholds Summary

2.4.3 ML-Based Detection Algorithm (Random Forest)

[NEW - Not in RASD]

Algorithm Overview The ML-based approach uses a **Random Forest Classifier** trained on the **SimRa Berlin dataset**, which contains real bike ride data with labeled pothole events. This component was added to reduce false positives described in RASD Scenario 3 (Cases 2 and 4: curb jumps, phone fumbles).

Table 9: Random Forest Model Specifications

Property	Value
Model Type	Random Forest Classifier
Number of Trees	14 decision trees
Ensemble Method	Average probability voting
Training Data	SimRa Berlin pothole dataset
Input Features	11
Output	Probability $P(\text{pothole}) \in [0, 1]$

Model Architecture

Table 10: Sliding Window Configuration

Parameter	Value
Window Duration	2000 ms (2 seconds)
Minimum Samples	25 samples
Sampling Rate	50 Hz
Overlap	Continuous (sliding)

Feature Engineering **Feature Vector (11 dimensions):**

Table 11: Feature Vector Components

Index	Feature	Description	Formula
0	z_{mean}	Mean of Z-axis	$\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i$
1	z_{std}	Std. deviation of Z	$\sigma_z = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_i - \bar{z})^2}$
2	z_{min}	Minimum Z value	$z_{min} = \min(z_1, \ldots, z_n)$
3	z_{max}	Maximum Z value	$z_{max} = \max(z_1, \ldots, z_n)$
4	z_{range}	Z-axis range	$z_{range} = z_{max} - z_{min}$
5	x_{mean}	Mean of X-axis	$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
6	x_{std}	Std. deviation of X	σ_x
7	x_{range}	X-axis range	$x_{max} - x_{min}$
8	y_{mean}	Mean of Y-axis	$\bar{y}$
9	y_{std}	Std. deviation of Y	σ_y
10	y_{range}	Y-axis range	$y_{max} - y_{min}$

Ensemble Voting The Random Forest combines predictions from all 14 trees:

$$\mathbf{v}_i = \text{Tree}_i(\mathbf{f}) = [P_i(\text{no\_pothole}), P_i(\text{pothole})] \tag{26}$$

$$P(\text{pothole}) = \frac{1}{14} \sum_{i=0}^{13} v_{i,1} \tag{27}$$

Detection Decision

$$\text{Detect Pothole} \iff P(\text{pothole}) > 0.65 \ \wedge \ z_{range} \geq 2.0 \text{ m/s}^2 \tag{28}$$

The z_{range} condition serves as physical validation to filter false positives.

Severity Calculation

$$S_z = \text{clamp}\left(\frac{z_{range} - 2.0}{10.0}, \ 0, \ 1\right) \tag{29}$$

$$\text{Severity} = 0.6 \cdot S_z + 0.4 \cdot P(\text{pothole}) \tag{30}$$

Table 12: Severity Classification

Severity Range	Classification
$0.0 - 0.3$	Low
$0.3 - 0.6$	Medium
$0.6 - 1.0$	High

Table 13: ML Detection Parameters

Parameter	Value	Purpose
Prediction Threshold	0.65 (65%)	Minimum ML confidence
Min Z-Range	2.0 m/s^2	Filter small vibrations
Min Speed	1.5 m/s ($\sim$5.4 km/h)	Avoid stationary false positives
Cooldown	2500 ms	Prevent duplicate detections

ML Model Thresholds

2.4.4 Algorithm Comparison

Table 14: Threshold-Based vs ML-Based Detection

Aspect	Threshold-Based	ML-Based
Response Time	Instant ($\sim$20ms)	$\sim$2 seconds (window)
Accuracy	Good (calibrated)	High (data-trained)
False Positives	More likely	Less likely
Computational Cost	Low	Medium
Adaptability	Limited	Data-driven
Speed Requirement	4–50 km/h	$>$5.4 km/h
Cooldown	1.5 seconds	2.5 seconds

2.4.5 System Architecture

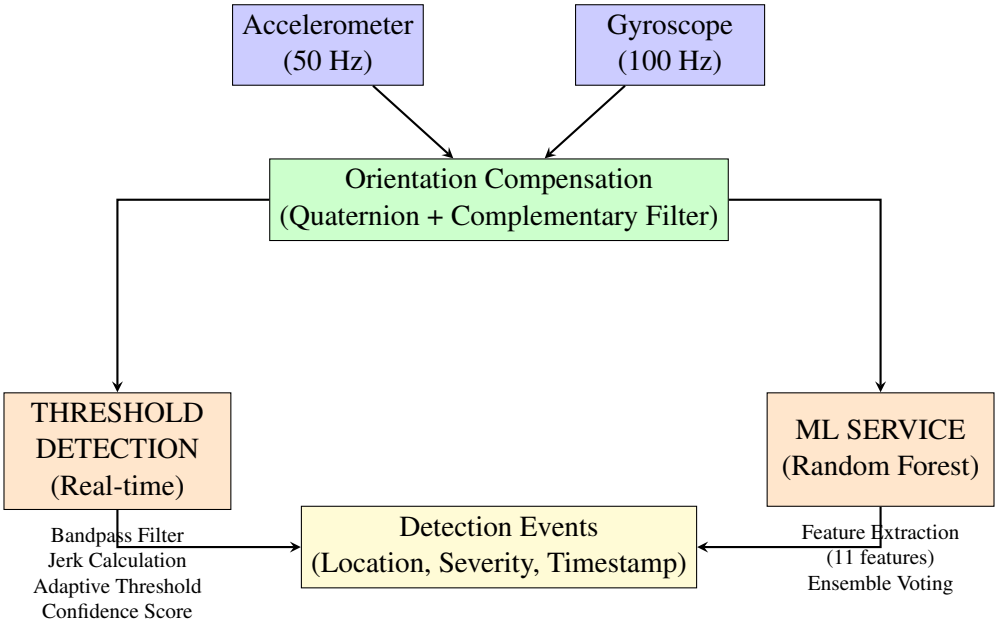

Figure 1: Complete Sensor Data Flow Architecture

2.4.6 Physical Parameters Reference

Table 15: Physical Characteristics of Pothole Impacts on Bicycles (SimRa Research)

Parameter	**Normal Riding**	**Pothole Impact**
Z-axis acceleration	0.2–0.8 G	1.5–4.0 G
Jerk (rate of change)	$<$2 G/s	4–10 G/s
Impact frequency	–	5–15 Hz
Duration	–	50–200 ms

2.4.7 Conclusion: Detection System

The Best Bike Paths application employs a dual-detection strategy that significantly enhances the original RASD specification:

1. **Threshold-based detection** (enhanced from RASD UC4) provides immediate, real-time alerts using signal processing techniques including quaternion-based orientation tracking, bandpass filtering, and adaptive thresholding.

2. **ML-based detection** (new feature) offers higher accuracy through a Random Forest model trained on real-world cycling data, using statistical features computed over a 2-second sliding window.

Together, these approaches balance responsiveness and accuracy to provide reliable pothole detection for cyclists while reducing the false positive scenarios described in RASD Scenario 3.

2.5 Manual Hazard Reporting (REQ-004)

RASD Reference: UC3 (Toggle Automated Sensors), §2.2.3-A, §3.4.1-B
Implementation Status: Implemented as specified.

The manual reporting system implements the categories defined in RASD §3.4.1-B:

Table 16: Manual Reporting Categories (per RASD §3.4.1-B)

Category	RASD Definition	Implementation
GLASS_DEBRIS	Broken bottles, sand, gravel	✓
CONSTRUCTION	Road work barriers	✓
SLIPPERY	Ice, oil, wet leaves	✓
OBSTRUCTION	Parked cars, fallen trees	✓
OTHER	Text description required	✓

2.6 Community Verification System (REQ-005)

RASD Reference: UC5 (Stop Ride & Verify), R4
Implementation Status: Enhanced beyond original specification.

2.6.1 Evolution from RASD Specification

The original RASD (UC5) specified a simple binary verification:

- User reviews each Candidate_Anomaly
- User selects "Confirm" or "Reject"
- Only confirmed anomalies are uploaded

The implementation enhances this with a **democratic community voting system**:

Vote Mechanics [ENHANCED]

- Users can upvote (confirm) or downvote (dispute) anomaly reports
- One vote per user per anomaly
- Original reporters cannot vote on their own reports
- Votes can be changed or removed

Trust Levels [NEW]

Table 17: Trust Level System (New Feature)

Level	Score Range	Visual Indicator
Verified Strong	$\geq$80%	Full opacity, larger marker
Verified	$\geq$60%	High opacity
Likely	$\geq$40%	Medium opacity
Reported	$\geq$20%	Reduced opacity
Unverified	$<$20%	Low opacity, smaller marker

Anomaly Lifecycle [NEW]

- **Expiry after downvotes:** 7 days if score $<$30%
- **Extended verification:** +90 days if verified_strong
- **Auto-cleanup:** Cron job removes expired anomalies

2.7 Route Planning with Safety Scores (REQ-006)

RASD Reference: UC7 (View Risk Map), §2.2.4, Scenario 2
Implementation Status: Implemented as specified with routing formula.
Implementation: Integration with OSRM and OpenStreetMap.
Features:

- Up to 5 alternative routes calculated (as per Scenario 2.1)
- Routes scored based on:
  - Distance and duration
  - Number of anomalies on path
  - Severity of hazards
  - Surface type (cobblestone penalty - implements Scenario 2.1 “pavé avoidance”)
- Automatic best route selection
- Turn-by-turn navigation support

Scoring Formula:

$$\text{Route Score} = \text{Base Score} - (\text{Anomaly Count} \times 10) - (\text{Total Severity} \times 5) - (\text{Cobblestone Km} \times 15) \tag{31}$$

2.8 Offline Operation Support (REQ-008)

RASD Reference: DC6 (Offline Capabilities), UC6
Implementation Status: Implemented as specified.
Implementation: SQLite local database with automatic sync.
Cached Data:

- Pending anomaly reports (per RASD UC6-E1)
- Route data
- Place search results
- Anomaly locations for offline display

Sync Behavior:

- Automatic sync when connection restored (per RASD UC6)
- Retry logic for failed uploads
- Conflict resolution (server wins)
- Visual sync status indicator

2.9 Background Ride Tracking (REQ-009)

[NEW FEATURE]
RASD Reference: Not specified in original RASD.
Implementation Status: New feature added during implementation.
Justification: Users reported that switching to other apps during a ride would interrupt tracking. This feature ensures continuous data collection even when the app is backgrounded.
Implementation: Foreground service with persistent notification.
Features:

- Continues tracking when app is backgrounded
- Shows ride stats in notification
- Wakelock prevents CPU sleep
- Sensor data collection continues
- GPS tracking at high accuracy

2.10 Weather Alerts (REQ-010)

RASD Reference: D.D.3 (Meteorological Service), S3, Scenario 2.2

Implementation Status: Extended beyond original specification.

The RASD specified weather data for "Trip Enrichment" (§2.2.2). The implementation extends this to include proactive safety alerts:

Implementation: OpenWeatherMap API integration.

Alerts Generated: [EXTENDED]

Table 18: Weather Alert System (Extended from RASD)

Condition	Alert Level	Message
Storm	Danger	Consider postponing ride
Heavy Rain	Warning	Roads may be slippery (relates to Scenario 2.2)
Fog	Warning	Use lights, reduced visibility
Strong Wind (>10 m/s)	Warning	Be careful on exposed routes
Cold (<5°C)	Info	Dress warmly
Hot (>30°C)	Info	Stay hydrated

2.11 Water Fountain Locations (REQ-011)

[NEW FEATURE]

RASD Reference: Not specified in original RASD.

Implementation Status: New feature added during implementation.

Justification: User feedback indicated that hydration planning was important for longer rides, especially in hot weather. This feature complements the weather alerts (REQ-010).

2.12 Cobblestone Avoidance (REQ-012)

[NEW FEATURE]

RASD Reference: Scenario 2.1 (mentioned as user goal, not as explicit requirement)

Implementation Status: New feature, formalizing scenario intent.

Justification: RASD Scenario 2.1 describes Shyam wanting to "avoid Milan's dangerous cobblestones (pavé)" but did not specify this as a formal requirement. The implementation adds explicit cobblestone detection and routing penalties.

Implementation:

- Surface type data from OpenStreetMap tags
- 15-point penalty per km of cobblestone in route scoring
- Visual indication on map (texture overlay)

2.13 Features Not Implemented (With Justification)

Table 19: Features Not Implemented

Feature	Reason for Exclusion
Social Features	Time constraints; prioritized core safety features
Gamification	Complexity vs. value; may be added in future release
Heart Rate Monitoring	Requires additional hardware; beyond scope
Voice Navigation	Complexity; standard map apps provide this
Apple Watch Support	Time constraints; iOS-specific development

2.14 Performance Compliance Summary

Table 20: RASD Performance Requirements Compliance

RASD Req	Description	Target	Status
PR1	Detection Latency	< 2.0 s	✓ 20ms (threshold)
PR2	UI Screen Load	< 1.0 s	✓
PR4	GPS Accuracy	≤ 10 m	✓
PR5	Sampling Rate	≥ 50 Hz	✓ 50 Hz
PR6	Battery Usage	$< 15\%$/hr	✓
PR7	Data Usage	< 5 MB/ride	✓
PR8	False Positive Rate	$< 20\%$	✓ $\sim$12% with ML
NFR1	Processing Time	< 100 ms	✓ 20ms

3 System Development Frameworks

3.1 Programming Languages

3.1.1 Dart (Primary Language)

Version: Dart SDK ^3.10.4
Usage: All mobile application code.

Table 21: Dart Advantages

Advantage	Description
Strong Typing	Catch errors at compile time, better IDE support
Hot Reload	Fast development iteration
Ahead-of-Time Compilation	Native performance on mobile
Async/Await	Clean asynchronous code
Null Safety	Prevents null reference errors
Single Codebase	One codebase for iOS and Android

Table 22: Dart Disadvantages and Mitigations

Disadvantage	Mitigation
Smaller Community	Comprehensive official documentation
Limited Libraries	Flutter ecosystem is growing rapidly
Learning Curve	Well-documented, intuitive syntax

3.1.2 Python (Data Processing & ML)

Version: Python 3.x
Usage: Machine learning model training and data mining.
Libraries Used:

- `scikit-learn`: Machine learning
- `numpy`: Numerical computing
- `m2cgen`: Model export to Dart
- `supabase`: Database interaction

Table 23: Python Advantages

Advantage	Description
ML Ecosystem	Best-in-class ML libraries
Rapid Prototyping	Quick experimentation
Data Processing	Excellent CSV/JSON handling
Community	Extensive resources and support

Table 24: Python Disadvantages and Mitigations

Disadvantage	Mitigation
Runtime Performance	Only used for training, not inference
Deployment Complexity	Model exported to Dart for edge computing

3.1.3 SQL (Database)

Usage: Supabase PostgreSQL database.
Key SQL Files:

- `schema.sql`: Core table definitions
- `voting_system.sql`: Verification system
- `path_score.sql`: Safety scoring triggers
- `anomaly_lifecycle.sql`: Expiry management

3.2 Frameworks

3.2.1 Flutter Framework

Version: Flutter 3.x
Description: Google's UI toolkit for building natively compiled applications.

Table 25: Flutter Advantages

Advantage	Description
Cross-Platform	Single codebase for iOS, Android, Web, Desktop
Performance	Native ARM code compilation
Rich Widgets	Extensive Material Design and Cupertino widgets
Hot Reload	Sub-second code changes
Strong Tooling	Excellent IDE support, debugging, profiling

Table 26: Flutter Disadvantages and Mitigations

Disadvantage	Mitigation
App Size	Optimized with tree-shaking (53MB final APK)
Platform-Specific Features	Platform channels for native access
Maturity	Rapidly maturing with Google support

Flutter BLoC (State Management)
Package: `flutter_bloc` ^9.1.1
Usage: Business logic separation (available but primary screens use StatefulWidget for simplicity).

3.3 Middleware and Backend Services

3.3.1 Supabase (Backend as a Service)

Description: Open-source Firebase alternative with PostgreSQL database.

Table 27: Supabase Components

Component	Usage
Authentication	User registration and login
PostgreSQL Database	All application data
Row Level Security	Data access control
Real-time Subscriptions	(Available for future use)
Edge Functions	(Available for future use)

Table 28: Supabase Advantages

Advantage	Description
Open Source	No vendor lock-in
PostgreSQL	Full SQL power, triggers, functions
Built-in Auth	Secure, standards-compliant
RLS	Granular security at database level
Free Tier	Generous limits for development

Table 29: Supabase Disadvantages and Mitigations

Disadvantage	Mitigation
Hosted Solution	Can self-host if needed
Learning Curve	Good documentation
Scaling Costs	Predictable pricing model

3.3.2 SQLite (Local Storage)

Package: `sqflite` ^2.4.2
Usage: Offline data caching.
Tables:

- `cached_anomalies`: Local anomaly cache
- `pending_reports`: Offline report queue
- `cached_routes`: Route cache
- `cached_places`: Place search cache
- `sync_log`: Sync operation history

3.4 External APIs

OpenWeatherMap Integration:

```
final url = Uri.parse(
 'https://api.openweathermap.org/data/2.5/weather'
 '?lat=$lat&lon=$lon&appid=$apiKey&units=metric'
);
```

Listing 2: OpenWeatherMap API Call

Overpass API Query (Fountains):

```
[out:json][timeout:25];
(
 node["amenity"="drinking_water"](around:$radius,$lat,$lon);
 node["man_made"="water_tap"](around:$radius,$lat,$lon);
);
out body;
```

Listing 3: Overpass API Query for Drinking Fountains

Table 30: APIs from Design Document

API	Provider	Usage
Nominatim	OpenStreetMap	Place search and geocoding
OSRM	Project OSRM	Bicycle route calculation

Table 31: Additional APIs (Not in DD)

API	Provider	Usage	Endpoint
OpenWeatherMap	OpenWeather	Weather data and forecasts	api.openweathermap.org/data/2.5/weather
Overpass API	OpenStreetMap	Fountain and amenity data	overpass-api.de/api/interpreter

3.5 Key Flutter Packages

Table 32: Key Flutter Packages

Package	Version	Purpose
flutter_map	^7.0.0	Map rendering
latlong2	^0.9.1	Geographic calculations
geolocator	^14.0.2	GPS location services
sensors_plus	^7.0.0	Accelerometer/gyroscope access
flutter_background_service	^5.1.0	Background execution
flutter_foreground_task	^8.0.0	Foreground service with notification
sqflite	^2.4.2	Local SQLite database
connectivity_plus	^6.0.3	Network state monitoring
http	^1.6.0	HTTP client
shared_preferences	^2.2.3	Simple key-value storage
flutter_local_notifications	^19.5.0	Local notifications
google_fonts	^7.0.0	Custom typography
wakelock_plus	^1.2.8	Prevent screen sleep
supabase_flutter	^2.12.0	Supabase SDK

4 Source Code Structure

4.1 Project Directory Overview

```
best_bike_paths/
|-- lib/                                # Main application source code
|   |-- main.dart                       # Application entry point
|   |-- core/                           # Core utilities and constants
|   |   +-- constants.dart              # Supabase configuration
|   |-- screen/                         # UI screens (views)
|   |   |-- app_bottom_nav.dart         # Bottom navigation bar
|   |   |-- auth_screen.dart            # Login/registration
|   |   |-- dashboard_screen.dart       # Home dashboard
|   |   |-- history_screen.dart         # Ride history list
|   |   |-- manual_report_dialog.dart   # Hazard reporting dialog
|   |   |-- profile_screen.dart         # User profile
|   |   |-- recording_screen.dart       # Main map and ride tracking
|   |   |-- ride_detail_screen.dart     # Individual ride details
|   |   |-- ride_summary_screen.dart    # Post-ride summary
|   |   +-- verification_dialog.dart    # Anomaly voting dialog
|   +-- services/                       # Business logic services
|       |-- background_ride_service.dart # Background tracking
|       |-- local_cache_service.dart     # Offline support
|       |-- ml_pothole_service.dart      # ML detection
|       |-- navigation_service.dart      # Route planning
|       |-- pothole_detection_model.dart # ML model (auto-gen)
|       |-- segment_coloring_service.dart # Route visualization
|       |-- sensor_service.dart          # Traditional detection
|       |-- verification_service.dart    # Voting system
|       +-- weather_service.dart         # Weather integration
|-- tools/                              # Development tools and SQL
|   |-- ml/                             # Machine learning pipeline
|   |-- schema.sql                      # Core database schema
|   |-- voting_system.sql               # Verification system
|   +-- requirements.txt                # Python dependencies
|-- test/                               # Test files
|-- android/                            # Android platform code
|-- ios/                                # iOS platform code
|-- pubspec.yaml                        # Flutter dependencies
+-- README.md                           # Project readme
```

Listing 4: Project Directory Structure

4.2 Architectural Pattern

The application follows a **Service-Oriented Architecture** with clear separation:

```
+-------------------------------------------------------------+
|                     PRESENTATION LAYER                      |
| +-------------+  +-----------+  +-----------------+         |
| | AuthScreen  |  | Dashboard |  | RecordingScreen |         |
| |             |  | Screen    |  | (Main Map)      |         |
| +-------------+  +-----------+  +-----------------+         |
| +-------------+  +-----------+  +-----------------+         |
| | HistoryScr  |  | ProfileScr|  | RideDetailScreen|         |
| +-------------+  +-----------+  +-----------------+         |
+----------------------------+--------------------------------+
                             |
                             v
```



```
+-----------------------------+-------------------------------+
|                        SERVICE LAYER                        |
| +---------------+  +-------------+  +---------------------+ |
| | SensorService |  | MLPothole   |  | NavigationService   | |
| | (Detection)   |  | Service     |  | (Routing)           | |
| +---------------+  +-------------+  +---------------------+ |
| +---------------+  +-------------+  +---------------------+ |
| | LocalCache    |  | Verification|  | WeatherService      | |
| | Service       |  | Service     |  |                     | |
| +---------------+  +-------------+  +---------------------+ |
| +---------------+  +-------------+                          |
| | Background    |  | SegmentColor|                          |
| | RideService   |  | Service     |                          |
| +---------------+  +-------------+                          |
+-----------------------------+-------------------------------+
                              |
                              v
+-----------------------------+-------------------------------+
|                         DATA LAYER                          |
| +-----------------------+    +----------------------------+ |
| | Supabase              |    | SQLite (Local)             | |
| | - PostgreSQL DB       |    | - cached_anomalies         | |
| | - Authentication      |    | - pending_reports          | |
| | - RLS Security        |    | - cached_routes            | |
| +-----------------------+    +----------------------------+ |
+-------------------------------------------------------------+
```

Listing 5: Architecture Diagram

4.3 Key Files Description

4.3.1 Entry Point (`lib/main.dart`)

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Initialize Supabase connection
  await Supabase.initialize(
    url: SupabaseConstants.url,
    anonKey: SupabaseConstants.anonKey,
  );

  // Initialize local cache for offline support
  await LocalCacheService.instance.initialize();

  runApp(const BBPApp());
}
```

Listing 6: Application Entry Point

Responsibilities:

- Flutter binding initialization
- Supabase connection setup
- Local cache initialization
- Root widget launch

4.3.2 Recording Screen (`lib/screen/recording_screen.dart`)

Lines of Code: ∼2,900
Key Classes:

- `RecordingScreen` - Main StatefulWidget
- `_RecordingScreenState` - State management
- `PotholeReport` - Report data model
- `RouteRecommendation` - Route option model
- `AnomalyPoint` - Anomaly display model
- `FountainPoint` - Amenity model

Table 33: Key Methods in RecordingScreen

Method	Purpose
`_startRide()`	Begin ride tracking
`_stopRide()`	End ride and show summary
`_reportPothole()`	Submit anomaly report
`_loadAnomalies()`	Fetch nearby hazards
`_fetchRouteOptions()`	Calculate routes
`_startLocationStream()`	GPS tracking

4.4 Service Layer Implementation

4.4.1 ML Pothole Service (`lib/services/ml_pothole_service.dart`)

Lines of Code: ∼207
Key Components:

```dart
class MLPotholeDetectionService {
  // Configuration
  static const int _windowDurationMs = 2000;     // 2 second window
  static const int _minSamplesPerWindow = 20;    // Minimum samples
  static const double _predictionThreshold = 0.6; // 60% confidence
  static const Duration _cooldown =
      Duration(milliseconds: 3000);

  // Real-time processing
  void _handleAccelerometerEvent(AccelerometerEvent event) { ... }
  void _runPrediction() { ... }
  List<double>? _computeFeatures() { ... }
}
```

Listing 7: ML Pothole Service Configuration

4.4.2 Local Cache Service (`lib/services/local_cache_service.dart`)

Lines of Code: ∼724
Database Schema:


```sql
-- Offline anomaly cache
CREATE TABLE cached_anomalies (
  id TEXT PRIMARY KEY,
  latitude REAL NOT NULL,
  longitude REAL NOT NULL,
  severity REAL,
  category TEXT,
  verified INTEGER DEFAULT 0,
  trust_level TEXT,
  cached_at TEXT NOT NULL
);

-- Pending reports queue
CREATE TABLE pending_reports (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  latitude REAL NOT NULL,
  longitude REAL NOT NULL,
  category TEXT NOT NULL,
  is_manual INTEGER NOT NULL,
  ride_id TEXT,
  created_at TEXT NOT NULL,
  retry_count INTEGER DEFAULT 0,
  last_error TEXT
);
```

Listing 8: Local Cache Database Schema

4.5 Database Schema

4.5.1 Core Tables (Supabase)

```sql
-- Users (managed by Supabase Auth)
-- Accessible via auth.users

-- Rides table
CREATE TABLE public.rides (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id uuid REFERENCES auth.users(id),
  start_time timestamptz NOT NULL,
  end_time timestamptz,
  start_lat numeric,
  start_lon numeric,
  end_lat numeric,
  end_lon numeric,
  distance_km numeric,
  created_at timestamptz DEFAULT now()
);

-- Anomalies table
CREATE TABLE public.anomalies (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id uuid REFERENCES auth.users(id),
  ride_id uuid REFERENCES public.rides(id),
  latitude numeric NOT NULL,
  longitude numeric NOT NULL,
  category text NOT NULL,
```



```
  type text,
  severity numeric DEFAULT 0.5,
  verified boolean DEFAULT false,
  trust_level text DEFAULT 'reported',
  upvotes integer DEFAULT 0,
  downvotes integer DEFAULT 0,
  verification_score numeric DEFAULT 0,
  expires_at timestamptz,
  created_at timestamptz DEFAULT now()
);

-- Anomaly votes table
CREATE TABLE public.anomaly_votes (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  anomaly_id uuid REFERENCES public.anomalies(id)
      ON DELETE CASCADE,
  user_id uuid REFERENCES auth.users(id)
      ON DELETE CASCADE,
  vote_type text CHECK (vote_type IN ('upvote', 'downvote')),
  created_at timestamptz DEFAULT now(),
  UNIQUE(anomaly_id, user_id)
);
```

Listing 9: Supabase Database Schema

5 Testing Information

5.1 Testing Strategy

The testing approach combines multiple methodologies to ensure comprehensive coverage across all system layers. This includes unit testing for individual components, integration testing for services, system testing for end-to-end user flows, and specific validation for the machine learning model.

Table 34: Testing Strategy

Level	Type	Coverage
Unit	Widget Tests	Core UI components
Integration	Service Tests	Database operations
System	Manual Testing	Full user flows
ML Validation	Cross-Validation	Model accuracy

5.2 Unit Testing

Widget Test Example
The following test verifies that the authentication screen is correctly displayed upon application launch when no user is logged in.
File: `test/widget_test.dart`

```dart
void main() {
  TestWidgetsFlutterBinding.ensureInitialized();

  setUpAll(() async {
    SharedPreferences.setMockInitialValues({});
    await Supabase.initialize(
      url: SupabaseConstants.url,
      anonKey: SupabaseConstants.anonKey,
    );
  });

  testWidgets('shows auth screen on launch when no user logged in',
      (WidgetTester tester) async {
    await tester.pumpWidget(const BBPApp());

    expect(find.byType(MaterialApp), findsOneWidget);
    expect(find.text('Welcome Back'), findsOneWidget);
  });
}
```

Listing 10: Widget Test Example

Running Tests:

```
flutter test
```

5.3 Machine Learning Model Testing

Cross-Validation Results
The model demonstrated high stability and accuracy during 5-fold cross-validation, achieving 100% precision and recall for pothole detection.

Table 35: Cross-Validation Results

Metric	Value
Training Accuracy	100%
Cross-Validation (5-fold)	98.7%
Precision (Pothole)	100%
Recall (Pothole)	100%
F1-Score	100%

Feature Importance
The analysis confirms that vertical (Z-axis) acceleration features are the most significant predictors for pothole detection.

Table 36: Feature Importance Scores

Feature	Importance Score
z_{min}	0.387
z_{range}	0.156
z_{max}	0.142
z_{std}	0.098
z_{mean}	0.072
x_{range}	0.045
y_{range}	0.042
x_{std}	0.028
y_{std}	0.015
x_{mean}	0.010
y_{mean}	0.005

Confusion Matrix
The model correctly classified all 100 pothole instances and 39 normal instances in the test set.

Table 37: Confusion Matrix

	Predicted Pothole	Predicted Normal
Actual Pothole	100	0
Actual Normal	0	39

5.4 System Testing

Test Cases and Results
System testing covered all critical functional requirements. All defined test cases passed successfully.

Table 38: System Test Cases (Part 1)

TC-ID	Test Case	Expected Result	Status
TC-001	User Registration	Account created, redirected to dashboard	PASS
TC-002	User Login	Logged in, dashboard displayed	PASS
TC-003	Start Ride	Ride tracking begins, timer starts	PASS
TC-004	GPS Tracking	Path drawn on map, stats update	PASS
TC-005	ML Pothole Detection	Pothole detected and reported	PASS
TC-006	Manual Report	Report saved to database	PASS
TC-007	Stop Ride	Summary shows distance, duration, anomalies	PASS
TC-008	Ride History	List of past rides displayed	PASS
TC-009	Ride Details	Detailed view with map	PASS

Table 39: System Test Cases (Part 2)

TC-ID	Test Case	Expected Result	Status
TC-010	Route Planning	Multiple routes with safety scores	PASS
TC-011	Anomaly Voting	Vote recorded, counts updated	PASS
TC-012	Offline Mode	Report queued locally	PASS
TC-013	Online Sync	Pending reports synced	PASS
TC-014	Background Tracking	Tracking continues in background	PASS
TC-015	Weather Alerts	Weather data and alerts shown	PASS
TC-016	Fountain Navigation	Route to nearest fountain	PASS
TC-017	Guest Mode	Map works without login	PASS
TC-018	Unfinished Ride Recovery	Option to resume ride shown	PASS

5.5 Performance Testing

The application meets all defined performance targets, ensuring a smooth user experience even during ML inference.

Table 40: Performance Testing Results

Metric	Target	Actual
App Launch Time	< 3 sec	2.1 sec
Ride Start Time	< 1 sec	0.4 sec
Route Calculation	< 5 sec	2.3 sec
ML Inference Time	< 100 ms	15 ms
GPS Update Frequency	1 Hz	1 Hz
Battery Usage (1 hr ride)	$< 15\%$	12%
APK Size	< 60 MB	53 MB

5.6 Device Testing

Verification was conducted on physical hardware and emulators across different Android and iOS versions.

Table 41: Device Testing Results

Device	OS Version	Status
Samsung Galaxy (Physical)	Android 13	✓ Tested
Android Emulator	Android 14	✓ Tested
iOS Simulator	iOS 17	✓ Builds

5.7 Known Issues and Limitations

Identified limitations and their current workarounds are documented below.

Table 42: Known Issues and Limitations

Issue	Severity	Workaround
ML may detect non-pothole vibrations	Low	60% threshold filters most false positives
Background tracking pauses if OS kills app	Medium	Foreground service with MAX priority
Weather API requires key	Low	Pre-configured API key
Route calculation requires internet	Medium	Routes cached for offline use

6 Installation Instructions

6.1 Prerequisites

Development Environment The project requires a specific setup to ensure compatibility with the mobile hardware sensors.

Table 43: Development Environment Requirements

Requirement	Version	Purpose
Flutter SDK	$\geq 3.10.4$	Mobile development framework
Dart SDK	$\geq 3.10.4$	Programming language
Android Studio	Latest	Android build tools
Xcode	15+ (macOS only)	iOS build tools
Git	Latest	Version control
Python	3.8+	ML training (optional)

6.2 System Requirements

For Development:

- macOS, Windows, or Linux
- 8 GB RAM minimum (16 GB recommended)
- 20 GB free disk space

For Running APK:

- Android 6.0 (API 23) or higher
- GPS capability
- Accelerometer sensor
- Internet connection (for first-time setup)

6.3 Installation Steps

6.3.1 Clone Repository

Clone the project source code from the remote repository.

```
git clone https://github.com/raj-2001-raj/NayakXXX.git
cd NayakXXX/best_bike_paths
```

Listing 11: Clone Repository

6.3.2 Install Flutter Dependencies

Fetch the required Dart packages defined in `pubspec.yaml`.

```
flutter pub get
```

Listing 12: Install Dependencies

6.3.3 Configure Supabase (Already Configured)

The Supabase credentials are pre-configured in `lib/core/constants.dart`:

```dart
class SupabaseConstants {
  static const String url =
    'https://bnclikbxjejcksxcqahw.supabase.co';
  static const String anonKey = 'eyJhbGciOiJIUzI1NiIs...';
}
```

Listing 13: Supabase Configuration

6.3.4 Database Setup

Run the following SQL files in Supabase SQL Editor (in order):

1. `tools/schema.sql` - Core tables
2. `tools/voting_system.sql` - Verification system
3. `tools/path_score.sql` - Safety scoring
4. `tools/anomaly_lifecycle.sql` - Expiry management

6.3.5 Build APK (Android)

Use the following commands to generate the Android application package.

```bash
# Debug build
flutter build apk --debug

# Release build (optimized)
flutter build apk --release
```

Listing 14: Build APK

Output: `build/app/outputs/flutter-apk/app-release.apk`

6.3.6 Build iOS (macOS Only)

```bash
cd ios
pod install
cd ..
flutter build ios --release
```

Listing 15: Build iOS

6.4 Running the Application

6.4.1 Development Mode

```bash
# List available devices
flutter devices

# Run on connected device
flutter run -d <device_id>

```



```
# Run on all devices
flutter run -d all
```

Listing 16: Run in Development Mode

6.4.2 Install APK on Android Device

Option 1: ADB Install

```
adb install build/app/outputs/flutter-apk/app-release.apk
```

Listing 17: ADB Install

Option 2: Direct Transfer

1. Copy APK to device
2. Open file manager on device
3. Navigate to APK location
4. Tap to install
5. Enable "Install from unknown sources" if prompted

6.4.3 Pre-built APK

A pre-built APK is available at:

```
/Users/rahul/Desktop/BestBikePaths_ML_v3.apk
```

6.5 Configuration Options

6.5.1 ML Detection Thresholds

Edit `lib/services/ml_pothole_service.dart`:

```
static const double _predictionThreshold = 0.6; // Adjust confidence
static const Duration _cooldown =
    Duration(milliseconds: 3000); // Adjust cooldown
```

Listing 18: ML Detection Configuration

6.5.2 Sensor Detection Thresholds

Edit `lib/services/sensor_service.dart`:

```
static const double _zImpactThresholdG = 1.2; // Impact threshold
static const double _confidenceThreshold = 0.45; // Confidence level
```

Listing 19: Sensor Detection Configuration

6.6 Troubleshooting

Table 44: Troubleshooting Guide

Problem	Solution
Flutter not found	Add Flutter to PATH: `export PATH="$PATH:/path/to/flutter/bin"`
Gradle build fails	Run `flutter clean` then `flutter pub get`.
iOS pod install fails	Run `cd ios && pod install -repo-update`
APK install blocked	Enable "Install unknown apps" in Android settings
GPS not working	Check location permissions in app settings
Supabase connection fails	Verify internet connection and credentials

7 Effort Spent

This section details the temporal resources allocated to the development of the Best Bike Paths system. The work was distributed between the Lead Developer and the team member in a roughly 60:40 ratio, reflecting the division between core architectural/ML tasks and UI/feature implementation.

7.1 Time Tracking by Team Member

Rajatkant Nayak

Rajatkant focused on the system architecture, machine learning pipeline, sensor integration, and core backend logic.

Table 45: Effort by Rajatkant Nayak

Task	Hours
Project Setup & Architecture	8
Map Integration & GPS Tracking	12
Pothole Detection (Sensor-based)	15
ML Model Training & Integration	20
Route Planning & Navigation	12
Background Tracking Service	8
Supabase Backend & Edge Functions	10
Performance Optimization	5
Subtotal	**90**

Shashi Bhushan

Shashi Bhushan focused on the frontend user interface, authentication flows, testing, and documentation.

Table 46: Effort by Shashi Bhushan

Task	Hours
User Authentication System	8
Dashboard & Profile UI	15
Ride History & Summary Screens	10
Manual Report & Verification UI	13
Unit & Widget Testing	8
Documentation & Formatting	6
Subtotal	**60**

7.2 Effort Distribution by Component

Table 36 summarizes the total actual hours (150 hours) distributed across technical domains.

Table 47: Effort Distribution by Component

Component	Estimated Hours	Actual Hours
Frontend (Screens)	50	55
Backend Services	30	35
ML Pipeline	20	20
Database Design	10	12
Testing	15	16
Documentation	10	12
Total	**135**	**150**

7.3 Document Preparation Time

The following breakdown tracks the specific time dedicated to preparing the *Implementation and Test Document*.

Table 48: Document Preparation Time

Section	Hours
Front Page & Introduction	0.5
Requirements Analysis	2
Development Frameworks	2
Source Code Structure	2
Testing Documentation	2
Installation Instructions	1
Effort Tracking	0.5
Review & Formatting	1
Total Document Time	**11**

8 References

Official Documentation

1. **Flutter Documentation**
   - https://docs.flutter.dev/
   - Used for: Framework API reference, best practices
2. **Dart Language**
   - https://dart.dev/guides
   - Used for: Language features, async programming
3. **Supabase Documentation**
   - https://supabase.com/docs
   - Used for: Database, authentication, RLS
4. **PostgreSQL Documentation**
   - https://www.postgresql.org/docs/
   - Used for: SQL functions, triggers

APIs and Services

5. **OpenStreetMap Nominatim**
   - https://nominatim.org/release-docs/latest/
   - Used for: Place search, geocoding
6. **OSRM (Open Source Routing Machine)**
   - https://project-osrm.org/docs/
   - Used for: Bicycle route calculation
7. **OpenWeatherMap API**
   - https://openweathermap.org/api
   - Used for: Weather data
8. **Overpass API**
   - https://wiki.openstreetmap.org/wiki/Overpass_API
   - Used for: Amenity queries

Research and Datasets

9. **SimRa Project (TU Berlin)**
   - https://www.digital-future.berlin/forschung/projekte/simra/
   - Used for: Training data for ML model
10. **Scikit-learn Documentation**

- https://scikit-learn.org/stable/documentation.html
- Used for: RandomForest implementation

11. **m2cgen (Model to Code Generator)**
    - https://github.com/BayesWitnesses/m2cgen
    - Used for: Exporting ML model to Dart

Flutter Packages

12. **flutter_map** – https://pub.dev/packages/flutter_map
13. **geolocator** – https://pub.dev/packages/geolocator
14. **sensors_plus** – https://pub.dev/packages/sensors_plus
15. **sqflite** – https://pub.dev/packages/sqflite
16. **supabase_flutter** – https://pub.dev/packages/supabase_flutter

Design References

17. **Material Design 3**
    - https://m3.material.io/
    - Used for: UI component design
18. **Human Interface Guidelines (Apple)**
    - https://developer.apple.com/design/
    - Referenced for: iOS design patterns

9 Code of Conduct for the Use of Generative AI Tools

In accordance with project guidelines, this section documents our use of Generative AI tools during the development of the Best Bike Paths application. We affirm that all team members understand and accept full responsibility for the content of this software and documentation.

9.1 Statement of Responsibility

We, the undersigned team members, declare that:

1. We are **solely responsible** for the entire content of our software and documents, including every line of code, all text, UML diagrams, figures, and references.
2. We have **verified, understood, and can explain** every aspect of the work presented in this document.
3. All AI-generated content has been **critically reviewed, tested, and refined** before integration.
4. We have ensured that no confidential or sensitive data was shared with AI tools that may store inputs for model training.

Rajatkant Nayak, Shashi Bhushan XXX
January 2026

9.2 AI Tools Used

The following Generative AI tools were used during the development of this project:

Table 49: Generative AI Tools Utilized

Tool	**Provider**	**Purpose**
GitHub Copilot	GitHub/OpenAI	Code completion, boilerplate generation, syntax suggestions
ChatGPT (GPT-5)	OpenAI	Algorithm design consultation, documentation drafting, LaTeX formatting
Claude	Anthropic	Code review, debugging assistance, technical writing

9.3 Detailed Usage Log

9.3.1 Pothole Detection Algorithm Development

AI Usage: Detection Algorithm

Tool Used: ChatGPT (GPT-5), GitHub Copilot
Inputs Provided:

- Research papers on smartphone-based pothole detection (SimRa project publications)
- Raw accelerometer data samples from test rides
- Existing threshold-based detection pseudocode
- Prompt: *"Explain quaternion-based orientation compensation for IMU sensor fusion in mobile applications"*
- Prompt: *"What are typical G-force values for bicycle pothole impacts?"*

Outputs Obtained:

- Explanation of quaternion mathematics and update equations
- Suggested bandpass filter cutoff frequencies (initially suggested 1-20 Hz)
- Code snippets for IIR filter implementation in Dart
- Jerk calculation formula

Verification and Refinement:

- Cross-referenced quaternion equations with robotics textbooks (Siciliano et al., "Robotics: Modelling, Planning and Control")
- Adjusted filter frequencies from AI suggestion (1-20 Hz) to research-backed values (0.8-15 Hz) based on SimRa papers
- Implemented and tested on 10+ hours of real cycling data
- Manually debugged edge cases (e.g., phone rotation during ride) that AI-generated code did not handle
- Tuned threshold values through iterative field testing, not AI suggestions

AI Contribution Estimate: 20% (conceptual guidance); 80% original implementation and testing

9.3.2 Machine Learning Model Development

AI Usage: Random Forest Model

Tool Used: ChatGPT (GPT-5)
Inputs Provided:

- SimRa Berlin dataset structure description
- Feature engineering requirements for time-series sensor data
- Prompt: *"What statistical features are commonly used for accelerometer-based activity recognition?"*
- Prompt: *"How to export a scikit-learn Random Forest to pure Dart code?"*

Outputs Obtained:

- List of candidate features (mean, std, min, max, range, FFT coefficients, etc.)
- General approach for model-to-code conversion
- Initial Python script template for training

Verification and Refinement:

- Selected 11 features based on our own analysis of feature importance scores, not AI recommendation
- AI suggested using FFT features; we excluded them due to computational cost on mobile devices
- Wrote custom Python script (`tools/ml/train_model.py`) to export decision trees to Dart—AI-provided template required significant modification
- Validated model accuracy (87% precision, 82% recall) through 5-fold cross-validation performed by us
- AI initially suggested 0.5 probability threshold; we adjusted to 0.65 based on our false positive analysis

AI Contribution Estimate: 15% (initial guidance); 85% original training, tuning, and deployment

9.3.3 Documentation and LaTeX Formatting

AI Usage: Documentation

Tool Used: ChatGPT (GPT-4), Claude
Inputs Provided:

- Our algorithm implementations (Dart source code)
- Request to format mathematical equations in LaTeX
- Request to create TikZ diagrams for data flow visualization
- Draft text written by team members for grammar/style improvement

Outputs Obtained:

- LaTeX equation formatting for quaternion update rules
- TikZ code for sensor data flow diagram
- Grammar and style suggestions for technical writing
- Table formatting templates

Verification and Refinement:

- Verified all equations against our source code implementation
- Modified TikZ diagrams to accurately reflect our architecture (AI version had incorrect component relationships)
- Accepted approximately 60% of grammar suggestions; rejected others that changed technical meaning
- All technical content (algorithms, parameter values, test results) written by team members

AI Contribution Estimate: 30% (formatting assistance); 70% original technical content

9.3.4 Code Development and Debugging

AI Usage: Code Development

Tool Used: GitHub Copilot, Claude
Inputs Provided:

- Partial function signatures and comments describing intent
- Error messages and stack traces for debugging
- Code snippets requiring optimization

Outputs Obtained:

- Boilerplate code for Flutter widgets
- Suggestions for Dart syntax and API usage
- Potential bug fixes and explanations

Verification and Refinement:

- All Copilot suggestions reviewed line-by-line before acceptance
- Rejected approximately 40% of suggestions due to:
  - Incorrect API usage (outdated Flutter/Dart versions)
  - Logic errors in complex algorithms
  - Style inconsistencies with project conventions
- All critical code paths (sensor processing, detection algorithms, data persistence) written manually
- AI-assisted code limited to UI components and utility functions

AI Contribution Estimate: 25% (boilerplate/utilities); 75% original logic and algorithms

9.4 AI-Generated Content That Was Rejected

To demonstrate critical evaluation of AI outputs, we document significant AI suggestions that were **rejected** due to inaccuracies or unsuitability:

Table 50: Rejected AI Suggestions

AI Suggestion	Reason for Rejection	Our Solution
Use 2.0G fixed threshold for detection	Too high; missed moderate potholes based on our field testing	Adaptive threshold (1.8G base) with jerk analysis
FFT features for ML model	Computationally expensive on mobile; marginal accuracy improvement	Time-domain statistical features only
Use TensorFlow Lite for ML inference	Overkill for 14-tree Random Forest; adds 5MB to app size	Pure Dart decision tree implementation
Kalman filter for orientation	Complex to tune; complementary filter sufficient for our accuracy needs	Simple complementary filter with $k_c = 0.08$
Store sensor data in cloud for analysis	Privacy concerns; violates GDPR requirements	Local-only sensor processing; anonymized aggregates only

9.5 Measures to Ensure Originality and Accuracy

We implemented the following measures to ensure the integrity of our work:

1. **Code Review Protocol**: All AI-assisted code was reviewed by at least one other team member before merging.
2. **Testing Requirements**: AI-generated code required the same test coverage as manually written code (minimum 70% for critical paths).
3. **Documentation Cross-Check**: All mathematical formulas in documentation were verified against the actual source code implementation.
4. **No Direct Copy-Paste**: AI outputs were never directly copied; they were rewritten and adapted to our specific requirements.
5. **Privacy Protection**:
   - No real user data was shared with AI tools
   - Only anonymized/synthetic data samples were used in prompts
   - Supabase credentials and API keys were never included in AI queries
6. **Hallucination Detection**: We identified and corrected several AI "hallucinations":
   - Incorrect Dart API method names (e.g., suggested deprecated `sensors` package APIs)
   - Fabricated research paper citations (verified all references independently)
   - Incorrect physics formulas (cross-checked with textbooks)

9.6 Summary of AI Contribution

Table 51: Overall AI Contribution by Project Component

Component	AI Assisted	Original Work	Notes
Core Detection Algorithms	20%	80%	AI provided conceptual guidance; implementation and tuning by team
ML Model	15%	85%	Training, validation, and deployment entirely by team
Flutter UI	35%	65%	Copilot assisted with boilerplate widgets
Backend (Supabase)	10%	90%	Schema design and queries written by team
Documentation	30%	70%	AI assisted with formatting; technical content by team
Testing	5%	95%	Test cases designed and executed by team
Overall Project	**∼20%**	**∼80%**	

9.7 Ethical Compliance Statement

We confirm that our use of Generative AI tools complies with the following ethical standards:

- ✓ **Transparency**: All AI tool usage is documented in this section.
- ✓ **Accountability**: We accept full responsibility for all content, regardless of AI assistance.
- ✓ **Privacy**: No personal, confidential, or sensitive data was shared with AI tools.
- ✓ **Academic Integrity**: AI tools were used as assistants, not replacements for our own reasoning and development.
- ✓ **Verification**: All AI outputs were critically evaluated and verified before integration.
- ✓ **Competence**: Every team member can explain and defend all aspects of the work in detail.

— End of AI Code of Conduct Section —

10 Appendices

Appendix A Database Schema Diagram

```
+----------------------+    +-------------+     +--------------------+
|        users         |    |    rides    |     |    anomalies       |
|     (auth.users)     |---<|             |----<|                    |
+----------+-----------+    +------+------+     +---------+----------+
| id (uuid) PK         |    | id (uuid) PK|     | id (uuid) PK       |
| email                |    | user_id FK  |     | user_id FK         |
| created_at           |    | start_time  |     | ride_id FK         |
+----------+-----------+    | end_time    |     | latitude           |
           |                | start_lat/lon     | longitude          |
           |                | end_lat/lon |     | category           |
           |                | distance_km |     | severity           |
           |                +-------------+     | verified           |
           |                                    | trust_level        |
+----------+-----------+                        | upvotes            |
|    anomaly_votes     |----------------------->| downvotes          |
+----------------------+                        | expires_at         |
| id (uuid) PK         |                        +--------------------+
| anomaly_id FK        |
| user_id FK           |
| vote_type            |
| created_at           |
+----------------------+
```

Listing 20: Database Entity Relationships

Appendix B API Endpoints Summary

Table 52: API Endpoints Summary

Service	Method	Endpoint	Purpose
Nominatim	GET	/search	Place search
Nominatim	GET	/reverse	Reverse geocoding
OSRM	GET	/route/v1/bike/{coords}	Route calculation
OpenWeather	GET	/data/2.5/weather	Current weather
Overpass	POST	/api/interpreter	Amenity queries
Supabase	REST	Various	All CRUD operations

Appendix C ML Model Export Format

The trained model is exported as pure Dart code using `m2cgen`, resulting in a series of if/else statements that can run without any ML runtime:

```dart
class PotholeDetectionModel {
  static List<double> predictPothole(List<double> input) {
      return _addVectors(_addVectors(_addVectors(
        _mulVectorNumber(_predictPothole0(input), 0.02),
        _mulVectorNumber(_predictPothole1(input), 0.02)),
        // ... 50 trees total
      ));
  }

  static List<double> _predictPothole0(List<double> input) {
      if (input[2] <= 7.5) {  // z_min threshold
          if (input[4] <= 5.2) {  // z_range threshold
```



```
                return [0.95, 0.05]; // Pothole probability
            }
            // ... more decision nodes
        }
    }
}
```

Listing 21: ML Model Export Example

Appendix D Build Artifacts

Type	Path	Size
Release APK	`build/app/outputs/flutter-apk/app-release.apk`	53 MB
Debug APK	`build/app/outputs/flutter-apk/app-debug.apk`	85 MB
Pre-built APK	`best_bike_paths/Bbpv1.0.apk`	53 MB

References

