BestBikePaths

Ianosel Bianca Roberta, Errigo Simone, Gholami Vajihe

System Scope and Actors

READ-ONLY INTERACTION
Guests can search and navigate paths without modifying any system data.

NO DATA STORAGE
Trip data is processed locally and is not saved once the session ends.

GUEST USER
(Read-Only Access)

ACTIVE CONTRIBUTION
Users can report obstacles, validate community reports, and create new paths.

DATA PERSISTENCE
Full lifecycle management of trip history, personalized statistics, and saved path libraries.

LOGGED-IN USER
(Read-Write Access)

System Goals & World Interactions

System Goals

[G1] Access Accurate Safety Information

Users receive real-time updates on path status and potential obstacles for informed cycling.

[G5] Active Community Crowdsourcing

Logged-in users drive the system by creating paths and submitting or confirming reports.

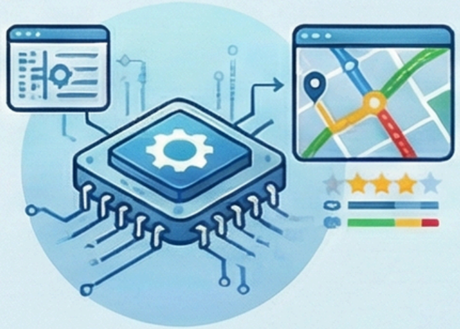
World Interactions

[W3] Physical Road Evolution

The physical state of the road is a world phenomenon that changes independently.

[SP9] World-to-Machine: Manual Reporting

Users share real-world observations with the machine by submitting digital condition reports.



[SP18] Machine-to-World: Ranking Updates

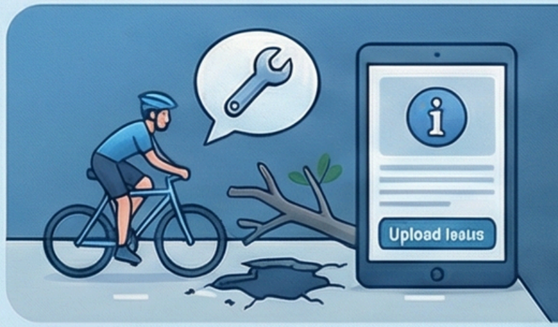
The system processes data to update path rankings, influencing how users navigate reality.

Key Requirements and Assumptions

ESSENTIAL SYSTEM REQUIREMENTS

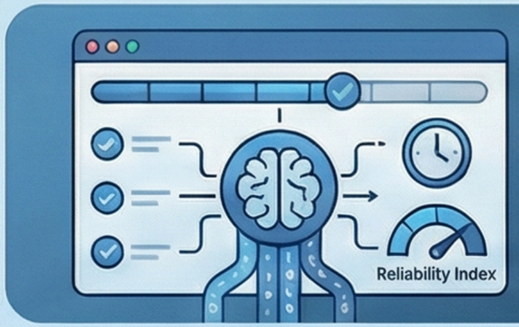
[R35 & R38] Public Browsing & Intelligent Ranking

Users can browse public paths ranked by safety and quality metrics.



[R26] Manual Obstacle Reporting

Logged-in users report path problems manually during active cycling trips.



[R32] Path Reliability Evaluation

System evaluates segment reliability based on user confirmations and data freshness.

FOUNDATIONAL DOMAIN ASSUMPTIONS

[D5] User Input Trustworthiness

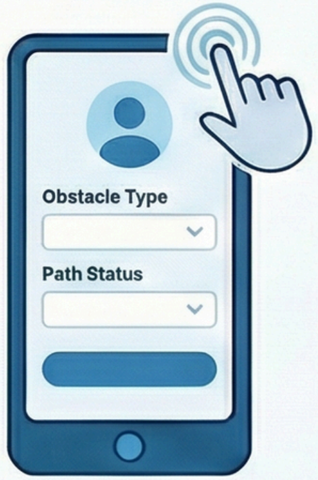
The system assumes manual user input is truthful, accurate, and complete.

[D8] Freshness Decay Logic

Information becomes obsolete over time; newer reports carry significantly more weight.

Meaningful use case

[UC17] Make a Report in Manual Mode



Initiate & Populate Report

User opens the report form and provides details ilke obstacle type and path status.

Real-Time GPS Capture

The mobile app automatically retrieves the current GPS coordinates to tag the report location.

Local Validation Check

The app ensures all mandatory fields are filled before attempting to contact the backend.

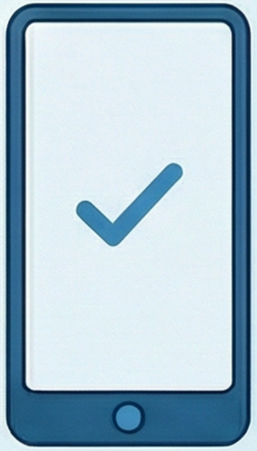
- **Report Matadata** (Manual User input)
- **GPS Coordinates** (Device GPS Module)
- **User Identifier** (Authenticated Session)

API Transmission

The validated payload is sent via a POST request to the backend API entry point.

Manager-Driven Storage

ReportManager delegates data persistence to the QueryManager to update the system database.



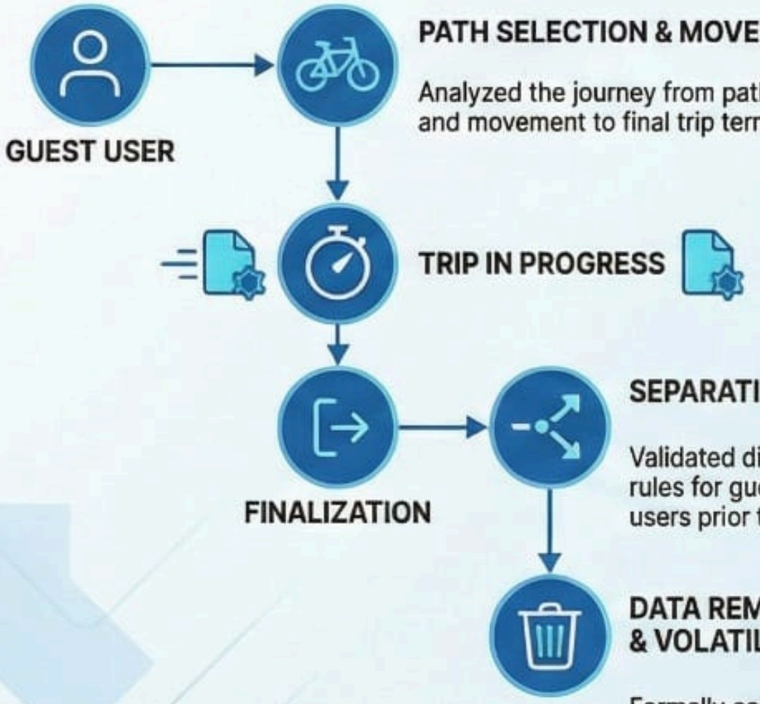
User Confirmation

Upon successful saving (201 Created), the app displays a success message to the cyclist.

Model Verification: Persistence & Consistency

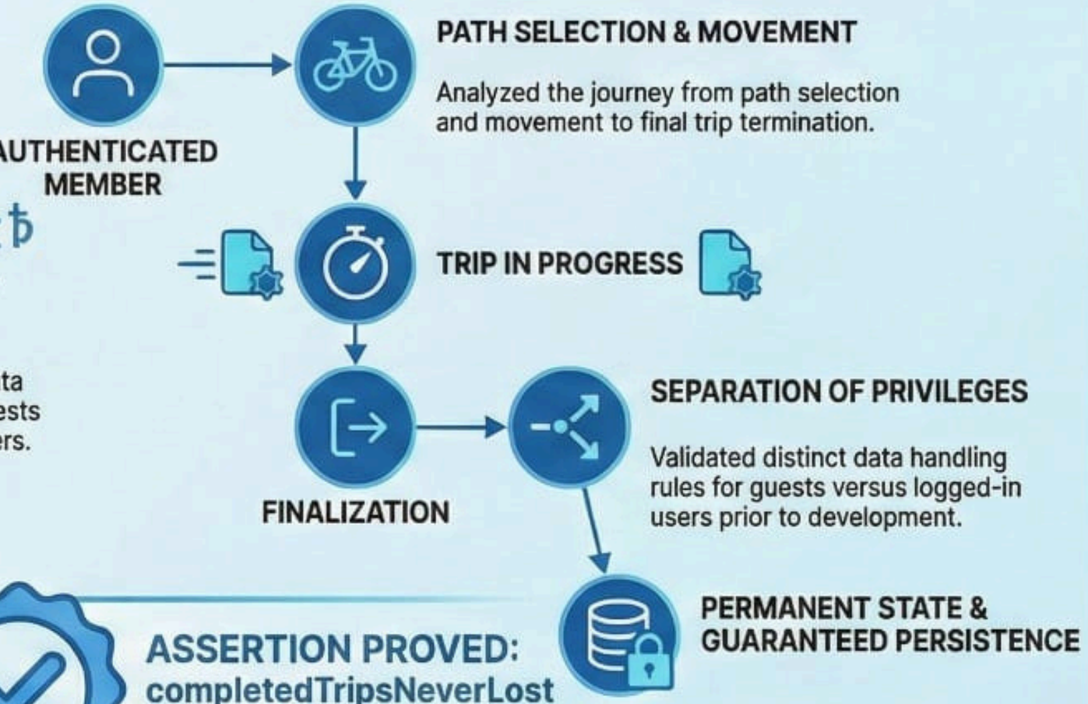
```
predicate guestCompleteCycle {}
```

GUEST COMPLETE CYCLE




```
predicate loggedCompleteCycle {}
```

LOGGED COMPLETE CYCLE



ASSERTION PROVED: completedTripsNeverLost

Mathematically proved that trips by logged-in users transition to a permanent state.

SYSTEM ARCHITECTURE & COMPONENTS

MODULAR BACKEND MANAGERS

USER MODULE

- AuthManager
- UserManager

OPERATIONAL MANAGERS

- Path
- Trip
- Report
- Stats

INTEGRATED EXTERNAL SERVICES

- Geocoding
- Snapping
- Weather
- Cache

DATA ACCESS

UNIFIED QUERYMANAGER INTERFACE

Isolates all database access logic to ensure consistency & modularity.

DBMS

MOBILE-CENTRIC THIN CLIENT

Handles UI rendering & sensor data acquisition, delegates logic to backend.

SINGLE API GATEWAY ENTRY POINT

Centralizes all routing, security middleware, & input validation for modular backend.

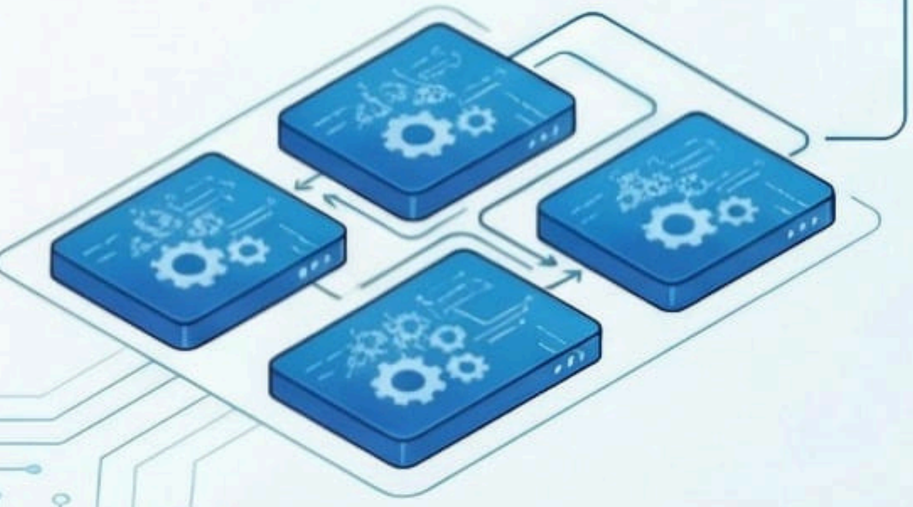
Architectural Styles and Patterns

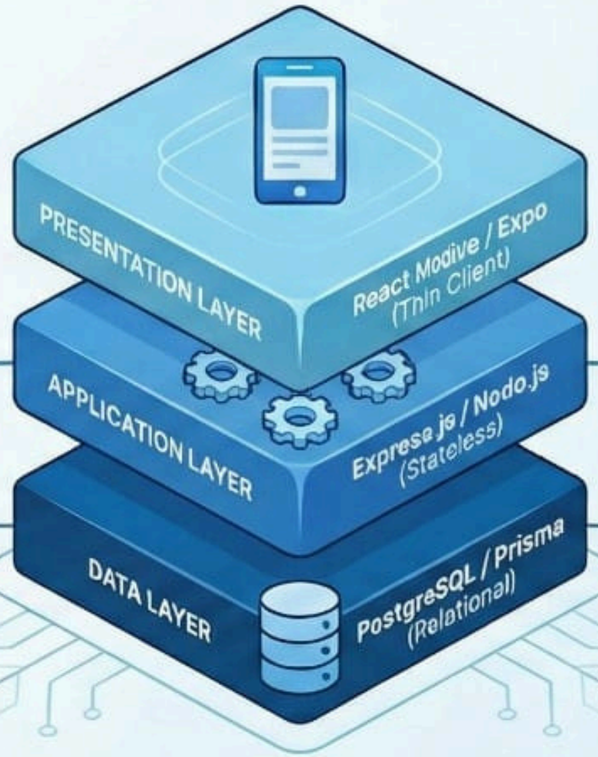
RESTful & Stateless Communication

Backend nodes remain stateless by storing session info client-side via short-lived tokens.

Modular Backend Structure

Organized into independent **Managers** encapsulating vertical slices of functionality.





Three-Tier Layered Architecture

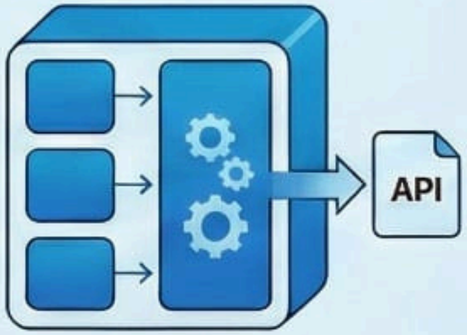
Separates software into presentation, application, and data layers to simplify system evolution.

DAO (Data Access Object) Pattern

The **QueryManager** isolates data access logic from business logic to ensure persistence consistency.

Service Layer Pattern

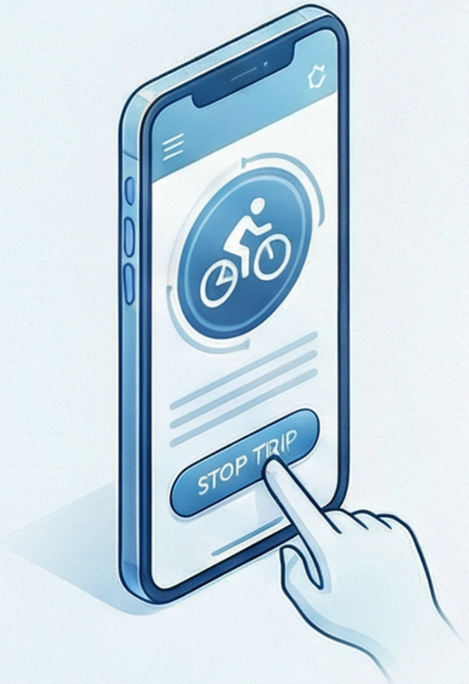
Managers act as service boundaries that hide internal logic and expose stable APIs.



JWT-Based Security Patterns

Employs JSON Web Tokens for stateless authentication and ownership-based authorization checks.

Meaningful Interaction: Trip Finalization Flow [UC16]



TripManager

Coordinates the entire data aggregation process once the mobile app sends the final payload.

WeatherService

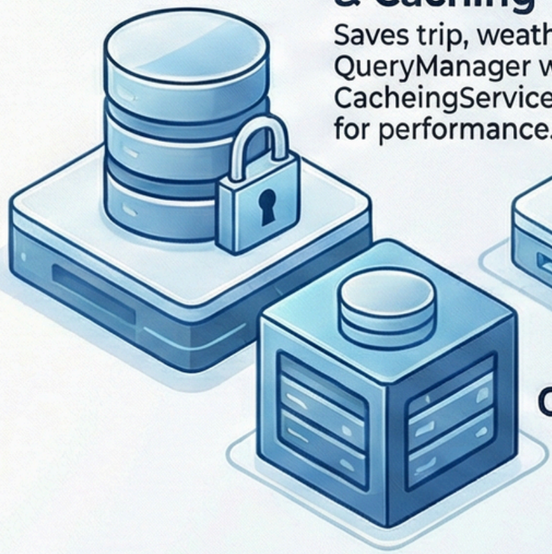
External Environmental Integration

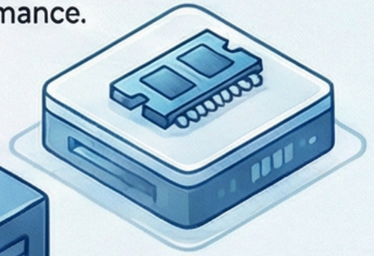
Fetches real-time weather context via the WeatherService to enrich the trip summary.

StatsManager

Real-Time Metric Calculation

Triggers the StatsManager to compute distance, speed, and elevation metrics immediately.





CacheingService

QueryManager

Atomic Persistence & Caching

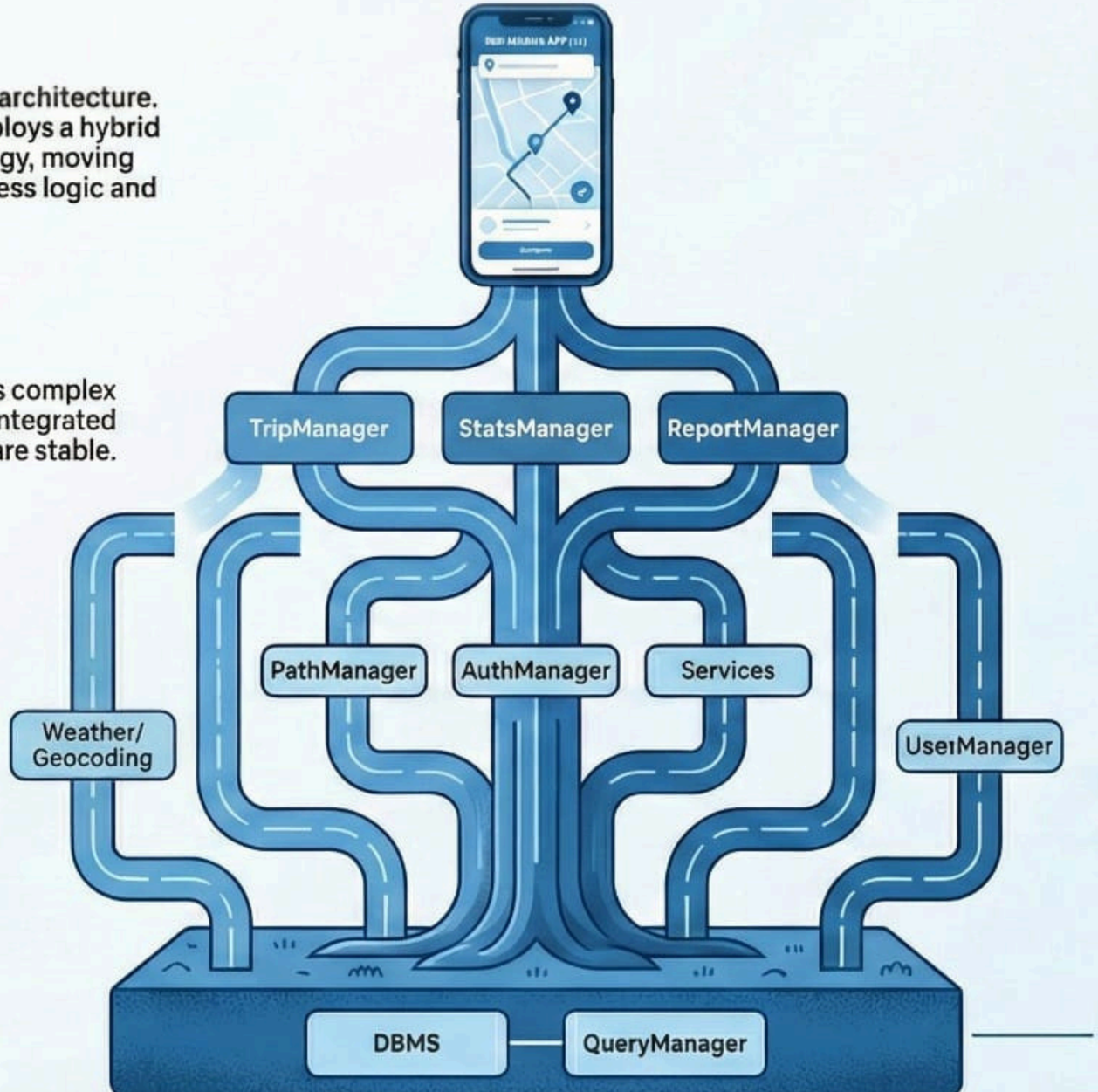
Saves trip, weather, and stats via QueryManager while updating the CacheingService for performance.

Implementation Strategy

The BBP system utilizes a modular three-tier architecture. To ensure reliability and speed, the team employs a hybrid "Mixed Thread-Based and Bottom-Up" strategy, moving from low-level data access to complex business logic and final UI integration.

Bottom-Up for Stability: Ensures complex components like TripManager are only integrated once their underlying dependencies are stable.

Core Service Integration: Parallel integration of identity services (AuthManager) and external utilities (Weather/Geocoding).



INTERFACE LAYER:
FULL SYSTEM INTEGRATION

COMPLEX LAYER:
HIGH DEPENDENCY

Final System Convergence: Complex Managers (Trip/Report) are layered on top, ending with the BBP Mobile App integration.

CORE LAYER:
LOW DEPENDENCY

Thread-Based for Speed:
Enables early integration of independent features like PathManager before larger modules are finished.

BASE LAYER:
INDEPENDENT / FOUNDATION

Development Frameworks & Tools

The Best Bike Paths (BBP) system is built on a modern three-tier architecture designed for a mobile-centric user experience. The stack was selected to ensure cross-platform compatibility, robust data integrity for community-reported cycling data, and high developer efficiency through full-stack type safety.

The Core Tech Stack

Frontend

React Native + Expo

- Enables a single codebase for IOS and Android with seamless access to GPS and sensors.

Backend

Node.js + Express

- A flexible, high-performance framework for building modular RESTful APIs and middleware.

Data Layer

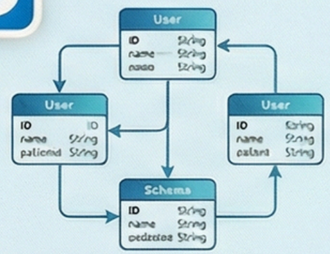
PostgreSQL + Prisma ORM

- Combines relational reliability with a type-safe client for consistent and safe data access.

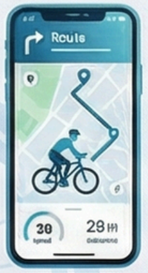
Language

TypeScript

- Used across the full stack to share types and eliminate common runtime errors.



Strategic Advantages



Mobile-First GPS Integration

Expo provides reliable location tracking and permission management for real-time trip recording.

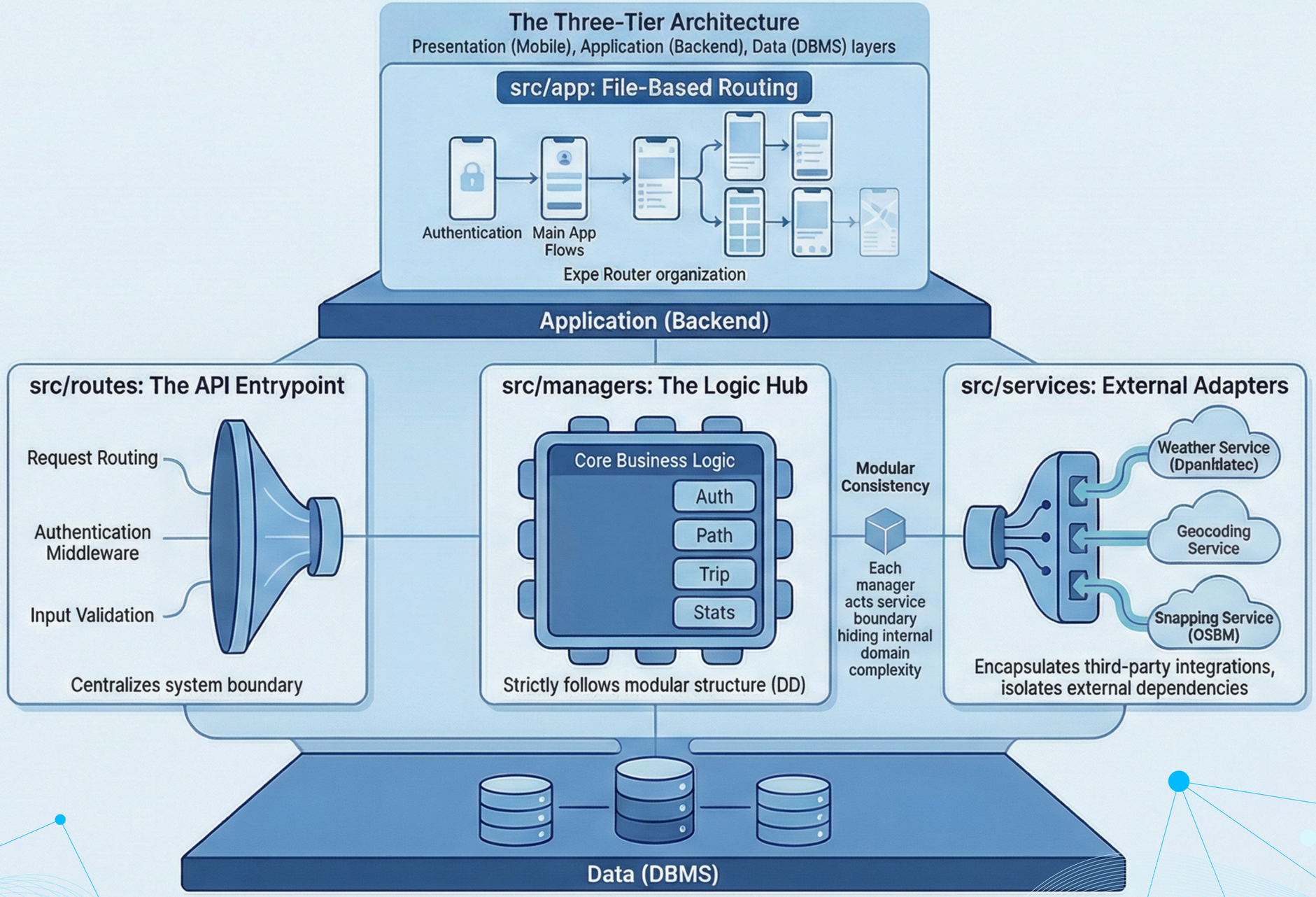
Schema Consistency

Prisma ORM ensures the database model remains perfectly aligned with TypeScript types during development.

Reproducible Environments

Docker Compose orchestrates the DB, Backend, and OSRM routing engine in isolated, consistent containers.

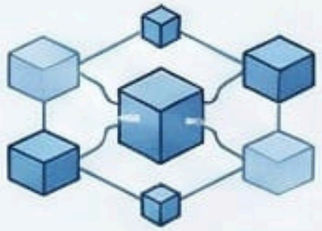
Code Structure & Design Alignment: From Design to Directory



A hybrid testing strategy combining 'Thread' and 'Bottom-Up' approaches ensures stable low-level data access before validating complex business logic across mobile frontend and Node.js backend.

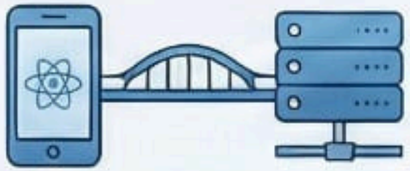
Testing Strategy & Selection

Unit Testing with Isolation



Individual modules are isolated using Jest and Mocks to verify internal logic.

Multi-Platform Coverage



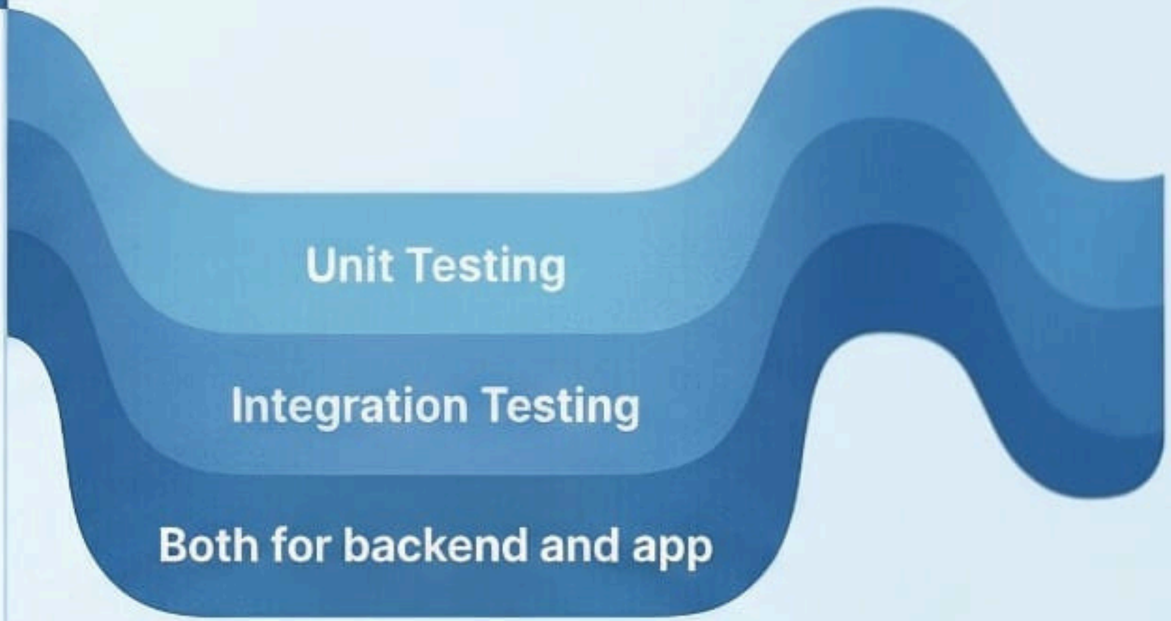
Frontend (React Native/Expo)

Backend (Express/Node)

Dedicated test suites cover both the React Native app and Express.js backend.

Selection Criteria

Tests prioritize API boundaries, authentication flows, and critical geospatial utilities.



Testing Scope Comparison across BBP System Tiers

	Backend (Express/Node)	Frontend (React Native/Expo)
Unit	Managers & Utilities (Jest)	Hooks & Utilities (Jest)
Integration	API Endpoints (Supertest)	Screen Interactions (Jest)

Path Status Evaluation: The Logic Behind the Route

PROBLEM: The Freshness Challenge

Static Data vs. Real-World Change

Road conditions evolve quickly, making old community reports obsolete and potentially misleading for cyclists.

SOLUTION: The Exponential Decay Model

The system applies an exponential decay formula to reduce the influence of older observations over time.

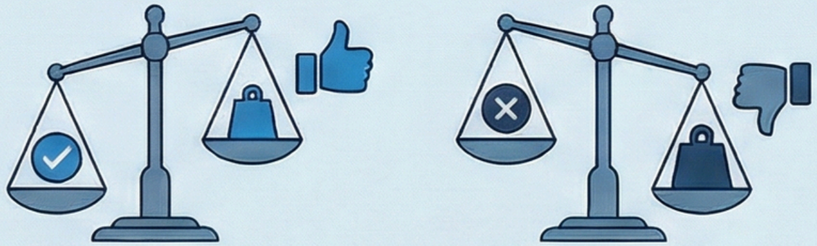
$$fresh_i = 2^{-ageMin_i/Hmin}$$

Time-Weighted "Freshness"

Newer reports carry significantly more weight to ensure the most current information dominates the score.

Reliability & Crowdsourcing Logic

Confirmations vs. Rejections



Confirmations (α=0.6)

Rejections (β=0.8)
More heavily penalized.

$$reportReliability = clamp(1 + \alpha \cdot confirmedScore - \beta \cdot rejectedScore, min, max)$$

Reliability Clamping

Reliability bounded between 0.1 and 2.5; scores below the threshold are discarded.

Path Aggregation

Path scores prioritize reported segments (70%) while factoring in the average of all segments (30%).

$$PathStatusScore = 0.7 \cdot score_{reportedSegments} + 0.3 \cdot score_{allSegments}$$

Path Status Mapping

Path Status	Numerical Score	Score Range
1. Optimal	5	[4.5, 5]
2. Medium	4	[3.5, 4.5)
3. Sufficient	3	[2.5, 3.5)
4. Requires Maintenance	2	[1.5, 2.5)
5. Closed	1	[1, 1.5)

Thank You!