POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

ITD

Best Bike Paths

Authors: **Matteo Caldognetto – Claudia Salone**

Student IDs: 11097854 – 10827565
Academic Year: 2025-26

Contents

1 | Introduction

A. Purpose

Best Bike Paths (BBP) is a community-driven web platform designed for cyclists to discover safe routes and report bike path conditions through crowdsourced data.

The system addresses the lack of centralized, up-to-date information about cycling infrastructure quality by enabling users to record bike trips, report path conditions (including obstacles such as potholes), and search for routes between locations.

A key feature is the aggregation algorithm that merges multiple user reports into a single, freshness-weighted path quality score.

B. Scope

Included in scope

- User registration and authentication (email/password, session-based).
- Trip recording with both manual and simulated (OSRM-based) data collection modes.
- Manual path condition reporting with obstacle marking.
- Path discovery and route search with interactive map visualization.
- Crowdsourced data aggregation with freshness-weighted scoring algorithm.
- Trip publishing as community paths.
- Obstacle lifecycle management.

Excluded from scope

- Real-time features (live tracking, WebSocket notifications).
- Social features (profiles, following, comments).
- Turn-by-turn navigation and custom pathfinding.
- Native mobile applications.

- Admin dashboard and analytics.
- Email verification and password reset.

Target users

Guest User (unregistered) Read-only access to path search, route visualization, and community map.

Registered Cyclist Full access including trip recording, path reporting, and data contribution.

C. Acronyms

BBP	Best Bike Paths
SPA	Single Page Application
tRPC	TypeScript Remote Procedure Call
ORM	Object-Relational Mapping
OSRM	Open Source Routing Machine
RASD	Requirements Analysis and Specification Document
DD	Design Document
ITD	Implementation and Testing Document
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CI/CD	Continuous Integration / Continuous Deployment
E2E	End-to-End
JSONB	JSON Binary (PostgreSQL data type)
GeoJSON	Geographic JSON
CSRF	Cross-Site Request Forgery
ACID	Atomicity, Consistency, Isolation, Durability
MVP	Minimum Viable Product

D. Reference Documents

The Software Engineering 2 course held by Prof.s Di Nitto Elisabetta, Camilli Matteo and Rossi Matteo Giovanni.

The provided document describing the project: Assignment IT AY 2025-2026

Project of last year provided as an assignment.

The provided document describing the project: Assignment RDD AY 2025-2026

2 | Implemented Functionalities and Requirements

A. Functionalities

Refer to the RASD document for detailed descriptions under section 2.A.1

Functionality	**Status**
Registration and authentication	Implemented
Automated route recording	Simulated
Manual route recording	Implemented
Path visualization between two points	Implemented
Display of the registered paths in the personal inventory	Implemented
Reporting another path	Implemented

B. Requirements

Refer to the RASD document for detailed descriptions under section 3.B

R1	The system shall allow a new user to create an account by providing necessary information.	Implemented
R2	The system shall allow a registered user to log in by providing their credentials.	Implemented
R3	The system shall permit users who are not logged in (guests) to use core viewing functions.	
R4	The system shall provide an interface for a logged-in user to view and edit their profile information.	Implemented
R5	The system shall allow a user to log out of their account on a device.	Implemented
R6	The system should attempt to detect when the user is cycling and start recording.	Implemented
R7	While recording, the system shall log the sequence of GPS coordinates that represent the user's path.	Not implemented
R8	The system shall automatically compute summary statistics for the trip.	Implemented

R9	After stopping the recording and computing stats, the system shall show the user a summary of their ride.	Simulated
R10	The system shall save the recorded trip data to the user's account.	Implemented
R11	When a trip is recorded, the system shall attempt to retrieve weather data for the	Not implemented
R12	If weather data was successfully obtained for a trip, the system shall display it along with the other stats on the trip summary view.	Not implemented
R13	The system shall provide a feature for users to add a bike path entry manually.	Implemented
R14	The system shall allow the user to mark any known obstacles or hazards along the path.	Implemented
R15	The system shall prompt the user whether to publish this information to the community or keep it private.	Implemented
R17	The system shall support continuing recording even if the user switches apps or turns off the screen, once a recording is started.	Implemented
R18	While a trip recording is active (started by the user), the system shall collect accelerometer and gyroscope data continuously in the background.	Not implemented
R19	The system shall analyze the sensor data in real-time (or in short batches) to detect possible road anomalies.	Not implemented
R20	For each detected potential obstacle or anomaly during the ride, the system shall log it internally in the context of the current recording.	Not implemented
R21	When the trip is finished, if the system has any automatically detected events, it shall prompt the user to review them.	Not implemented
R22	The system shall present the detected events in a user-friendly way.	Not implemented
R23	The user shall be able to confirm, reject or modifies the events.	Not implemented
R24	After the user has finished reviewing and confirming the data from an automated collection ride, the system shall save the verified information.	Not implemented
R25	The system shall maintain a central repository of all published bike path entries.	Implemented
R26	The system shall allow updates to existing path entries.	Implemented
R27	Whenever new path information is published, the system shall integrate it into the used for route queries.	Implemented
R28	The system shall ensure that published path data does not include personal information about cyclist	Implemented
R29	The system shall provide an interface for the user to specify a start and end location for their desired trip (route query).	Implemented

R30	The system shall compute a score or rating for each route found, to communicate its quality to the user.	Implemented
R31	The system shall display the calculated route(s) on the map UI for the user.	Implemented
R32	For each route, the system shall provide a summary of key details.	Implemented
R33	The system shall provide a section where a logged-in user can view their past recorded trips.	Implemented

3 | Adopted Development Frameworks

The system follows a three-tier architecture: a React-based presentation layer communicates with a Hono/tRPC application layer, which persists data to a PostgreSQL data layer. The frontend and backend share TypeScript types end-to-end via tRPC, eliminating runtime type mismatches. The entire codebase is organized as a Bun monorepo using workspaces.

A. Adopted Programming Language

A.1. Typescript

TypeScript is the sole programming language used across the entire stack — frontend, backend, database schema, tests, and build scripts. It was chosen for its strong static typing, which catches errors at compile time and provides end-to-end type safety when combined with tRPC and Drizzle ORM. TypeScript compiles to JavaScript and runs on the Bun runtime.

A.2. SQL (Postgres Dialect)

SQL is used indirectly through Drizzle ORM for database schema definitions and migrations. The generated migration files contain raw SQL DDL statements.

A.3. HTML/CSS

The frontend uses JSX (embedded in TypeScript .tsx files) for markup and Tailwind CSS v4 for styling. No raw HTML or CSS files are authored; all UI is composed declaratively through React components and utility-first CSS classes.

B. Adopted Middleware

B.1. Hono

Hono is a lightweight, ultra-fast web framework designed for edge runtimes. It serves as the HTTP layer for the API server, handling request routing, middleware composition, and response serialization. Hono was chosen because it runs natively on Cloudflare Workers (the target production deployment) while also supporting standard Bun/Node.js servers for local development.

B.2. tRPC

tRPC provides end-to-end type-safe RPC between the frontend and backend without code generation or schema files. Procedures defined on the server are automatically typed on the client, ensuring that API contract changes are caught at compile time. tRPC is mounted on Hono at /api/trpc/* and supports request batching for performance.

B.3. React

React 19 is the UI library for the frontend SPA. It was chosen for its mature ecosystem, strong TypeScript support, concurrent rendering capabilities, and wide industry adoption. Functional components with hooks are used exclusively (no class components).

C. Utilized APIs not Present in the DD

OpenStreetMap Nominatim (Geocoding & Street Search) Convert street names to geographic coordinates and search for cyclable streets.

OSRM (Open Source Routing Machine) Calculate cycling routes between two geographic points with realistic geometry.

OpenWeatherMap (Weather Enrichment) Enrich trip data with meteorological conditions.

D. Platforms and Libraries

D.1. Better Auth

Better Auth is the authentication framework handling user registration, login, session management, and CSRF protection. It integrates directly with Drizzle ORM for session storage in PostgreSQL. For the demo prototype, authentication is simplified to email/-password only (no email verification, no OAuth, no password reset).

D.2. Drizzle ORM

Drizzle ORM provides type-safe SQL query building and schema management for PostgreSQL. It was chosen over Prisma for its lighter footprint, direct SQL mapping, and native PostgreSQL feature support. Drizzle Kit handles migration generation and application.

D.3. TanStack Router

TanStack Router provides file-based, type-safe routing for the React application. Route groups (auth) and (app) separate public and protected pages. Route guards via beforeLoad hooks enforce authentication requirements.

D.4. TanStack React Query

React Query manages server state (API data caching, refetching, and synchronization). It integrates with tRPC to provide automatic query invalidation, optimistic updates, and offline-first behavior. Configuration includes a 2-minute stale time and 3-attempt retry with exponential backoff.

E. Runtime and Environment

E.1. Bun

Bun has been adopted as the primary JavaScript runtime and toolchain for the project, effectively replacing Node.js to streamline the development environment. It was selected for its ability to execute TypeScript natively, which eliminates the need for external transpilers or complex build configurations during the development phase, ensuring that the running code mirrors the source files without intermediate steps.

Beyond its role as a runtime, Bun serves as the project's package manager, orchestrating the entire codebase structure. It manages the monorepo configuration through workspaces, handling dependency installation and symbolic linking between the separate frontend and backend applications efficiently. Furthermore, the integrated test runner is utilized to execute the unit and component tests described in Section 5, providing high-performance execution for the test suites directly from the project root.

4 | Source Code Structure

The project is organized as a Bun monorepo with workspaces. The architectural pattern is a three-tier layered architecture with clear separation between presentation, application logic, and data persistence.

A. Components in the Source Code

The system is composed of three main runtime components that communicate over HTTP/tRPC.

The frontend is a single-page application built with React 19. It communicates exclusively with the API server via tRPC HTTP calls. The Leaflet map library renders geographic data (paths, routes, obstacles) on interactive OpenStreetMap tiles. State is managed through a combination of TanStack React Query (server state), Jotai (global UI state), and React useState (local component state).

The API server exposes six tRPC routers (user, trips, path, pathReport, street, routing) and additional HTTP endpoints (health check, OSRM proxy, auth). Each tRPC procedure is either publicProcedure (no auth required) or protectedProcedure (session required). The server also hosts the aggregation engine, which recalculates street status and path scores whenever a publishable report is created or modified.

The database stores 10 tables across three domains: authentication (user, session, identity, verification), bike trip management (trip, tripRoute, tripRating), and path/reporting (path, pathSegment, pathReport, obstacleReport). All geographic data is stored as GeoJSON in JSONB columns. The Drizzle ORM provides type-safe query building and migration management.

The API server includes a reverse proxy to a public OSRM instance that enables realistic route generation for the simulated trip mode. This component validates the routing profile (bike, car, foot) and coordinate format before forwarding the request.

5 | Testing

A. Integration Testing

Integration testing focused on verifying the correct interaction between the different modules of the system, specifically the database, the backend, and the frontend.

Database Layer: The integration of the PostgreSQL database and the Drizzle ORM schema was verified during the initial development phases. The correctness of the tables and their relationships was confirmed by monitoring the Docker container logs and utilizing Drizzle Studio for visual schema inspection.

API Layer: The communication between the React frontend and the Hono backend, managed via tRPC, was tested by verifying the data flow during application usage. Rather than using external tools, the integration was validated directly within the browser environment.

B. Automated Testing

To ensure reliability and prevent regression, the project includes both Unit Tests and End-to-End (E2E) tests.

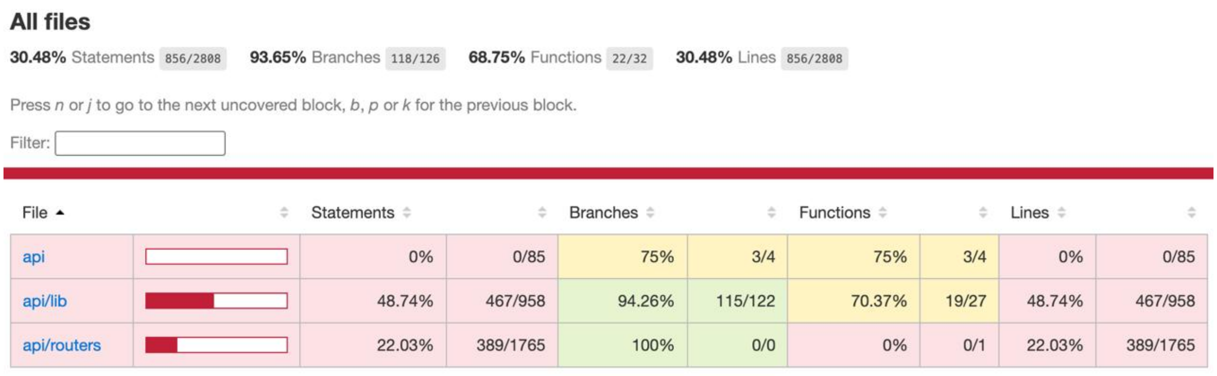
B.1. Unit and Component Testing

The test suite includes 392 tests across 19 files, split into two main areas.

API (apps/api/) — 15 files, 310 tests

These tests focus on core backend business logic, including the aggregation engine (weighted status calculation, the path score formula with weights, freshness decay, and a recency bonus). They also cover routing behavior and the OSRM adapter (response parsing plus error handling using mocked `fetch`).

Validation and security are extensively tested as well: Zod schemas for all tRPC endpoints, authentication enforcement, XSS/SQL injection sanitization, rate limiting, and OSRM proxy parameter validation. The obstacle state machine is verified end-to-end, including allowed and disallowed transitions across PENDING/CONFIRMED/REJECTED/CORRECTED/EXPIRED, along with tests for path publishing eligibility, spatial search (bounding boxes, corridor filtering, ranking), rating persistence and aggregation, and HTTP endpoints exercised against the real Hono app (health check, API info, OSRM proxy).



All files

30.48% Statements `856/2808` **93.65%** Branches `118/126` **68.75%** Functions `22/32` **30.48%** Lines `856/2808`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

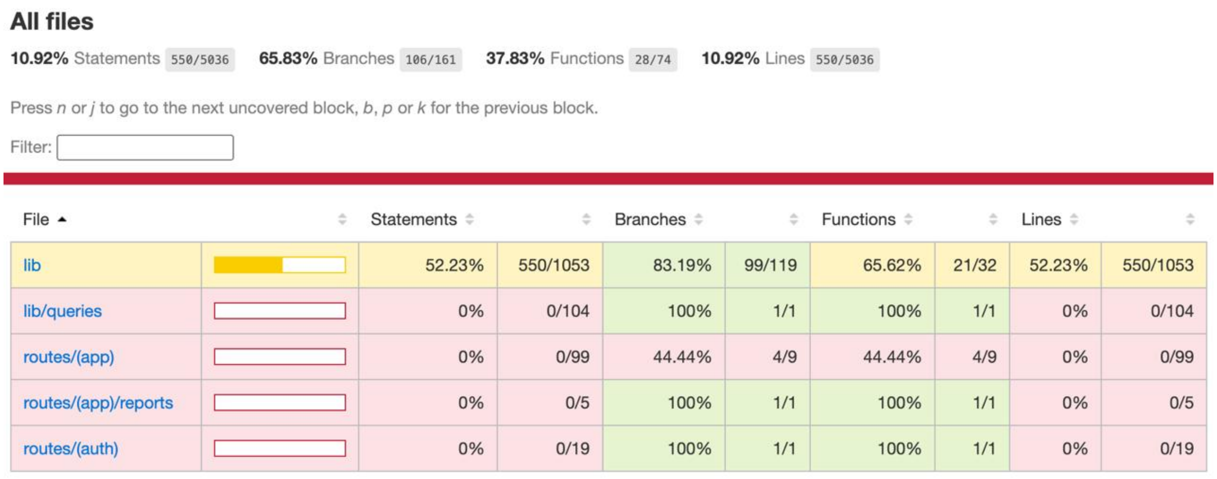
Filter:

File ▲		Statements		Branches		Functions		Lines	
api		0%	0/85	75%	3/4	75%	3/4	0%	0/85
api/lib		48.74%	467/958	94.26%	115/122	70.37%	19/27	48.74%	467/958
api/routers		22.03%	389/1765	100%	0/0	0%	0/1	22.03%	389/1765

Figure 5.1: Unit Test Coverage Report for the API Workspace

App (apps/app/) — 4 files, 86 tests

These tests cover frontend utility logic extracted into testable modules, including autocomplete (suggestion filtering, query validation, keyboard navigation), formatting helpers (duration, distance, average speed), and map utilities (status-to-color mapping, bounding box calculation). They also validate the report form, trip simulation (route decomposition, GPS track generation, weather data), OSRM client utilities (coordinate generation, route result structure), and Overpass helpers (street snapping and street distance calculations).



All files

10.92% Statements `550/5036` **65.83%** Branches `106/161` **37.83%** Functions `28/74` **10.92%** Lines `550/5036`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
lib		52.23%	550/1053	83.19%	99/119	65.62%	21/32	52.23%	550/1053
lib/queries		0%	0/104	100%	1/1	100%	1/1	0%	0/104
routes/(app)		0%	0/99	44.44%	4/9	44.44%	4/9	0%	0/99
routes/(app)/reports		0%	0/5	100%	1/1	100%	1/1	0%	0/5
routes/(auth)		0%	0/19	100%	1/1	100%	1/1	0%	0/19

Figure 5.2: Unit Test Coverage Report for the App Workspace

Execution: To run the unit tests for the specific workspaces, the following commands are used from the project root:

- **API Tests:**

```
        bun api:test

```

- **App (Frontend) Tests:**

```
        bun app:test

```

B.2. E2E Testing

Playwright acts as an automated user, launching a real browser instance (Chromium) and interacting with the application exactly as a human would (clicking buttons, filling forms, navigating pages). This ensures that the Database, Backend, and Frontend work correctly together in a production-like environment.

The following table describes the test cases considered for the system validation and their respective outcomes:

Table 5.1: System Tests: Authentication & Access Control

Requirement	Input / Action	Expected Outcome
R1 - Registration Form *Allow a new user to create an account by providing necessary information.*	User navigates to the sign-up page.	Name, email, and password inputs are visible; Sign-up title is displayed.
R1 - Valid Registration *Allow a new user to create an account by providing necessary information.*	User fills valid name, email, and password.	User is redirected to the authenticated area ('/routes').
R1 - Duplicate Email *Allow a new user to create an account by providing necessary information.*	User attempts to register with an existing email.	Error message displayed (matching "already exists" or "registered").
R1 - Password Validation *Allow a new user to create an account by providing necessary information.*	User enters a password shorter than 8 characters.	Validation text "Must be at least 8 characters" is visible.
R1 - Input Validation *Allow a new user to create an account by providing necessary information.*	User enters an invalid email format.	Browser validation blocks submission (checkValidity returns false).

Requirement	**Input / Action**	**Expected Outcome**
R2 - Sign In Form *Allow a registered user to log in by providing their credentials.*	User visits the login page.	Email and password fields are visible; Sign-in title is displayed.
R2 - Valid Login *Allow a registered user to log in by providing their credentials.*	User enters valid email and password.	Redirects to '/routes' (Dashboard).
R2 - Login Error *Allow a registered user to log in by providing their credentials.*	User enters incorrect email or password.	Error message displayed (matching "invalid", "incorrect", or "not found").
R3 - Redirect Logic *Permit users who are not logged in (guests) to use core viewing functions.*	Unauthenticated user tries to access a protected page (e.g., '/trips'), then logs in.	System redirects the user back to the originally requested page.
R2 - Auth Guard *Allow a registered user to log in by providing their credentials.*	Authenticated user attempts to visit the login page.	System automatically redirects the user to '/routes'.
R5 - Logout *Allow a user to log out of their account on a device.*	User clicks the Logout button from the Settings page.	Session is cleared; User is redirected to the login page.
R5 - Access After Logout *Allow a user to log out of their account on a device.*	User attempts to access a protected route after logging out.	Access denied; User is redirected to the Login page.
R3 - Guest Paths View *Permit users who are not logged in (guests) to use core viewing functions.*	Unauthenticated user navigates to '/paths'.	Page loads successfully; Map and titles are visible.
R3 - Guest CTA *Permit users who are not logged in (guests) to use core viewing functions.*	Guest user views the Paths page when data exists.	"Call to Action" banner is displayed inviting the user to sign in.
R3 - Guest Route Guard *Permit users who are not logged in (guests) to use core viewing functions.*	Guest user attempts to access '/trips', '/trip-record', or '/report'.	System intercepts the request and redirects to '/login'.

Requirement	Input / Action	Expected Outcome
R3 - Guest Search *Permit users who are not logged in (guests) to use core viewing functions.*	Unauthenticated user visits the '/routes' page.	Search inputs and map are fully visible and interactive.

Table 5.2: System Tests: Automated Data Collection & Sensors

Requirement	Input / Action	Expected Outcome
R6 - Automatic Mode *Attempt to detect when the user is cycling and start recording.*	User visits the Trip Record page.	"Automatic Mode" button is visible and interactive.
R7 - Route Generation *Log the sequence of GPS coordinates that represent the user's path.*	User fills trip info and clicks "Automatic Mode".	System simulates GPS data; Route count is greater than 0.
R8 - Metadata Display *Automatically compute summary statistics for the trip.*	User generates routes in Automatic Mode.	Generated route cards display calculated distance information.
R12 - Auto-fill Data *Display weather data along with other stats if successfully obtained.*	User initiates Automatic Mode.	Trip details and weather section are automatically populated.
R6 - Multiple Routes *Attempt to detect when the user is cycling and start recording.*	User requests automatic route generation.	System returns one or more route options (count ≥ 1).
R18/R19 - Sensor Sim *Collect and analyze accelerometer and gyroscope data continuously.*	Automatic mode runs with simulated sensor inputs.	"Detected Obstacles" section becomes visible in the UI.
R11 - Weather Integration *Attempt to retrieve weather data for the ride.*	System fetches environmental data during recording.	Weather section is visible; "Weather Condition" label is displayed.

Requirement	**Input / Action**	**Expected Outcome**
R21/R23 - Obstacle Verification *Prompt user to review detected events; allow confirm/reject.*	User accesses the obstacle management interface.	"Mark Obstacles" button is visible to allow user verification.
R24 - Data Persistence *Save verified information after user review and confirmation.*	User rates, reviews, and saves the auto-generated trip.	Trip data is persisted; User is redirected to the '/trips' list.

Table 5.3: System Tests: End-to-End User Journeys

Scenario	**Input / Action**	**Expected Outcome**
New User Onboarding	Guest lands on paths page, signs up, records first trip (auto mode), saves it.	Account created; Trip saved successfully; User redirected to '/trips' history.
Guest Transition	Guest browses map, attempts to access '/trips', signs up.	Redirected to login, then automatically back to the originally requested '/trips' page.
Daily Commute	User searches work route, records trip, marks obstacles, rates and saves.	Trip saved with obstacle data; New entry appears in personal history.
Publishing Cycle	User records a trip ("Scenic Route"), rates it 5 stars, selects "Publish".	Trip becomes public; Verified visible in the community paths list.
Reporter Flow	Full cycle: Search route → Record trip → Save → Browse Community.	User completes the flow of consuming data (search) and contributing data (record/publish).

Table 5.4: System Tests: Manual Path Recording & Obstacles

Requirement	**Input / Action**	**Expected Outcome**
R13 - Manual Entry Mode *Provide a feature for users to add a bike path entry manually.*	User visits Trip Record page (default view).	Trip Name input and manual entry elements are displayed.

Requirement	**Input / Action**	**Expected Outcome**
R13 - Autocomplete *Provide a feature for users to add a bike path entry manually.*	User accesses the street search field.	Street autocomplete component is visible and functional.
R6 - Mode Switching *Attempt to detect when user is cycling (Automatic vs Manual).*	User clicks the "Automatic Mode" button.	Interface switches mode; Route generation is triggered (loading indicator/routes appear).
R13 - Validation *Provide a feature for users to add a bike path entry manually.*	User attempts to review trip without entering a name.	"Review" button remains disabled or validation prevents progression.
R14 - Obstacle Mode *Allow the user to mark any known obstacles or hazards along the path.*	User clicks "Enable Obstacle Mode" (or Mark Obstacles).	UI indicator confirms mode activation.
R14 - Marking Obstacles *Allow the user to mark any known obstacles or hazards along the path.*	User clicks on the map while in Obstacle Mode.	Dialog opens allowing input of obstacle details (type, description).
R14 - Obstacle Types *Allow the user to mark any known obstacles or hazards along the path.*	User interacts with the obstacle type selection.	Dropdown menu displays multiple obstacle types (count $>$ 1).
R15 - Publish from Review *Prompt user whether to publish to the community or keep private.*	User records a trip, rates it, and clicks "Publish".	Publish action completes (confirmation dialog handled); Trip is made public.
R15 - Publish from List *Prompt user whether to publish to the community or keep private.*	User clicks "Make Public" on a saved private trip in the list.	Button updates to "Published" or trip status changes in the list.
R15 - Rating Requirement *Prompt user whether to publish to the community or keep private.*	User checks publish availability without a rating.	System ensures rating is present or handles the validation logic before publishing.

Table 5.5: System Tests: Path Repository & Updates

Requirement	**Input / Action**	**Expected Outcome**
R25 - Community Access *Maintain a central repository of all published bike path entries.*	User navigates to the Paths page ('/paths').	Page title is displayed; Map component loads successfully showing community data.
R29 - Search Interface *Provide interface for user to specify start and end location (route query).*	User visits the Routes page.	Start and End street input fields are visible and interactive.
R29 - Route Query *Provide interface for user to specify start and end location (route query).*	User initiates a search between two streets (e.g., "Via Torino" to "Corso Buenos Aires").	Search completes successfully; URL updates to reflect search parameters.
R3 - Guest Search *Permit users who are not logged in (guests) to use core viewing functions.*	Unauthenticated guest visits the Routes page.	Search functionality is fully available without requiring login.
R26 - Report Access *Allow updates to existing path entries.*	Authenticated user navigates to the Report page ('/report').	User is granted access; Path search input for reporting is visible.
R26 - Condition Reporting *Allow updates to existing path entries.*	User views the report interface.	Controls for rating paths (stars) and adding notes/descriptions are present.
R15 - Private Storage *Prompt user whether to publish to the community or keep private.*	User saves a trip without publishing it.	Trip remains visible only in the personal '/trips' list (Private) and checks verify it's saved.
R27 - Public Availability *Integrate new path information into the dataset used for route queries.*	User publishes a trip.	Trip becomes visible on the public '/paths' community map.

Table 5.6: System Tests: Personal Management & Community Map

Requirement	Input / Action	Expected Outcome
R33 - My Paths Access *Provide a section where a logged-in user can view their past recorded trips.*	Authenticated user navigates to ’/trips’.	"My Paths" page title is visible.
R33 - Trip Metadata *Provide a section where a logged-in user can view their past recorded trips.*	User views the trip history list.	Trip cards are displayed containing distance, duration, and average speed metadata.
R6 - Record Navigation *Attempt to detect when the user is cycling and start recording.*	User clicks the "Record Path" button from the history page.	System redirects the user to the ’/trip-record’ page.
R25 - Map Display *Maintain a central repository of all published bike path entries.*	User visits the Paths page (’/paths’) or Routes page.	The community map component (Leaflet container) loads and is visible.
R3 - Guest Viewing *Permit users who are not logged in (guests) to use core viewing functions.*	Unauthenticated user visits the map page.	Map is fully visible; A "Want to contribute?" CTA banner is displayed.
R2 - Auth View *Allow a registered user to log in by providing their credentials.*	Authenticated user visits the map page.	Map is visible; The guest CTA banner is NOT displayed.
R31 - Map Interactivity *Display the calculated route(s) on the map UI for the user.*	User interacts with the map (hover/click).	Map container remains visible and receives user events.

Table 5.7: System Tests: Route Finder & Visualization

Requirement	**Input / Action**	**Expected Outcome**
R29 - Search Interface *Provide interface for user to specify start and end location (route query).*	User visits the Routes Finder page.	Start/End street inputs, Search button, and Map are fully visible.
R29 - Initial State *Provide interface for user to specify start and end location (route query).*	User visits the page without performing a search.	An info box explaining how to use the feature is visible.
R29 - Route Query *Provide interface for user to specify start and end location (route query).*	User fills valid Start/End streets and clicks Search.	System queries the database; URL updates with ‘startStreet‘ and ‘endStreet‘ parameters.
R29 - URL Loading *Provide interface for user to specify start and end location (route query).*	User navigates directly to a URL with search parameters.	Search is automatically triggered; Results are loaded without manual input.
R30 - Route Scores *Compute a score or rating for each route found, to communicate quality.*	System returns multiple routes for a query.	Each route displays a calculated score; Routes are sorted by score (Best first).
R30 - Result Limit *Compute a score or rating for each route found, to communicate quality.*	Search returns many potential paths.	System limits the display to a maximum of 5 top-scoring results.
R31 - Visualization *Display the calculated route(s) on the map UI for the user.*	User selects a specific route card from the results list.	The selected path geometry (polyline) is highlighted on the map.
R31 - Multi-Select *Display the calculated route(s) on the map UI for the user.*	User switches selection between different route options.	Map updates dynamically to show the newly selected path.

Requirement	Input / Action	Expected Outcome
R29 - No Results *Provide interface for user to specify start and end location (route query).*	User searches for disconnected or invalid locations.	"No routes found" message or info box is displayed; Map remains in a neutral state.

Table 5.8: System Tests: Trip Creation, Statistics & Details

Requirement	Input / Action	Expected Outcome
R6 - Trip Recording *Attempt to detect when the user is cycling and start recording.*	User navigates to '/trip-record'.	Trip recording form, including Name input and "Automatic Mode" button, is displayed.
R6 - Automatic Creation *Attempt to detect when the user is cycling and start recording.*	User fills trip info, selects "Automatic Mode", sets rating, and saves.	Trip is created; User is redirected to trips list; New trip appears in the list.
R6 - Name Validation *Attempt to detect when the user is cycling and start recording.*	User attempts to review/save without a trip name.	Action is blocked; "Review" button is disabled or validation error appears.
R7 - Route Generation *Log the sequence of GPS coordinates that represent the user's path.*	User initiates Automatic Mode.	System simulates route data; Route count becomes greater than 0; Map markers (Start/End) appear.
R8 - Review Statistics *Automatically compute summary statistics for the trip.*	User completes recording and enters review screen.	Trip statistics (Distance, Duration) are calculated and displayed.
R8 - Distance Calculation *Automatically compute summary statistics for the trip.*	User reviews a recorded trip.	Total distance is displayed (non-zero value with correct units km/m).
R8 - Duration Track *Automatically compute summary statistics for the trip.*	User completes a trip.	Total elapsed time is tracked and displayed in the review summary.

Requirement	Input / Action	Expected Outcome
R8 - Speed Calc *Automatically compute summary statistics for the trip.*	User reviews trip statistics.	Average speed is calculated and displayed.
R12 - Weather Auto-fill *Display weather data along with other stats if successfully obtained.*	User uses Automatic Mode for recording.	Weather section is visible; Temperature and conditions are automatically populated.
R33 - Empty State *Provide a section where a logged-in user can view their past recorded trips.*	User visits '/trips' with no saved history.	Empty state message or placeholder is displayed correctly.
R33 - Card Display *Provide a section where a logged-in user can view their past recorded trips.*	User views the trip list.	Trips are shown as cards containing summary statistics (Distance, Rating).
R33 - Delete a trip *Provide a section where a logged-in user can view their past recorded trips.*	User clicks "Delete" on a trip and confirms.	Trip is removed from the list; Total trip count decreases.
R9 - Trip Detail View *Show the user a summary of their ride after stopping recording.*	User clicks "View" on a trip card.	Navigates to '/trip-detail'; Full map visualization and detailed stats are shown.

Running the E2E Tests:

To execute the End-to-End test suite, the environment must be properly configured.

Ensure that Node.js is installed on the machine. It can be downloaded from the official website: https://nodejs.org/en/download.

- **Initial Setup (First Run Only):** Before running the tests for the first time, execute the following commands to install dependencies and generate the necessary authentication states:

  1. Install project dependencies:

  ```
      bun install

  ```

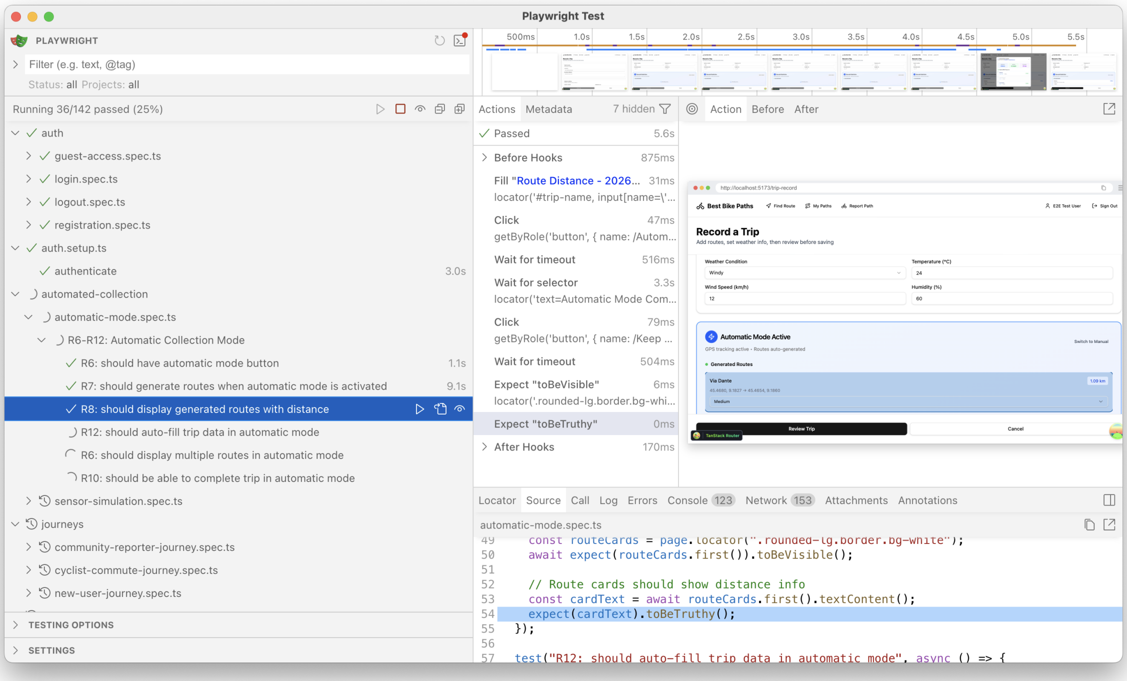
  2. Navigate to the e2e directory and run the authentication setup:

```
    npx playwright test tests/auth.setup.ts

```

- **Execution (UI Mode):** To run the tests, navigate to the e2e directory and launch the interactive UI mode. This opens the Playwright Inspector, allowing step-by-step execution and visual debugging.

```
    npx playwright test --ui

```

The E2E testing suite focuses on validating functional requirements and critical user journeys (R1 to R33) within a real browser environment. Unlike unit tests, which measure code-level coverage, these tests ensure that the integration between the frontend, backend, and database remains consistent during complex operations. The following figure demonstrates the Playwright execution environment, where each test step is visually verified against the application's UI.



Running 36/142 passed (25%)

```
✓ auth
  ✓ guest-access.spec.ts
  ✓ login.spec.ts
  ✓ logout.spec.ts
  ✓ registration.spec.ts
✓ auth.setup.ts
  ✓ authenticate                                             3.0s
automated-collection
  automatic-mode.spec.ts
    R6-R12: Automatic Collection Mode
      ✓ R6: should have automatic mode button                1.1s
      ✓ R7: should generate routes when automatic mode is activated   9.1s
      ✓ R8: should display generated routes with distance
      R12: should auto-fill trip data in automatic mode
      R6: should display multiple routes in automatic mode
      R10: should be able to complete trip in automatic mode
  sensor-simulation.spec.ts
journeys
  community-reporter-journey.spec.ts
  cyclist-commute-journey.spec.ts
  new-user-journey.spec.ts
```

```
✓ Passed                                          5.6s
Before Hooks                                      875ms
Fill "Route Distance - 2026...                    31ms
locator('#trip-name, input[name=\'...
Click                                             47ms
getByRole('button', { name: /Autom...
Wait for timeout                                  516ms
Wait for selector                                 3.3s
locator('text=Automatic Mode Com...
Click                                             79ms
getByRole('button', { name: /Keep ...
Wait for timeout                                  504ms
Expect "toBeVisible"                              6ms
locator('.rounded-lg.border.bg-whi...
Expect "toBeTruthy"                               0ms
After Hooks                                       170ms
```

http://localhost:5173/trip-record

Best Bike Paths · Find Route · My Paths · Report Path · E2E Test User · Sign Out

Record a Trip

Add routes, set weather info, then review before saving

Weather Condition: Windy
Temperature (°C): 24
Wind Speed (km/h): 12
Humidity (%): 60

Automatic Mode Active — Switch to Manual

GPS tracking active • Routes auto-generated

Generated Routes

Via Dante — 1.09 km
45.4680, 9.1827 → 45.4654, 9.1860
Medium

Review Trip · Cancel · TanStack Router

automatic-mode.spec.ts

```
    const routeCards = page.locator(".rounded-lg.border.bg-white");
    await expect(routeCards.first()).toBeVisible();

    // Route cards should show distance info
    const cardText = await routeCards.first().textContent();
    expect(cardText).toBeTruthy();
  });

  test("R12: should auto-fill trip data in automatic mode", async () => {
```

Figure 5.3: Playwright UI Mode interface. The sidebar displays the organized test suites (authentication, automated collection, and user journeys), while the central panel shows the step-by-step execution and successful validation of the route generation logic (Requirement R8).

6 Installation Instructions

A. Prerequisites

Bun	= 1.2.0	TypeScript runtime and package manager
Docker	Latest stable	PostgreSQL container for local development
Git	Any recent	Version control
Node.js	18+ (optional)	Not strictly required; Bun replaces Node.js

B. Preparing the Project

Install all dependencies

```
bun install
```

This installs dependencies for all monorepo workspaces (apps, packages, db, e2e). The postinstall script automatically installs Playwright browsers.

Configure environment variables The project includes a default `.env` file pre-configured for local development:

```
APP_NAME=Best Bike Paths
APP_ORIGIN=http://localhost:5173
API_ORIGIN=http://localhost:8787
ENVIRONMENT=development
DATABASE_URL=postgres://postgres:postgres@localhost:5432/bbp
BETTER_AUTH_SECRET=<your-generated-secret>
PORT=8787
```

Generate a new `BETTER_AUTH_SECRET`:

```
bunx @better-auth/cli@latest secret
```

Replace the placeholder value in `.env` with the generated secret.

Start PostgreSQL

```
docker compose up -d postgres
```

Verify it is running:

```
docker compose ps
# Expected: bbp-postgres running on port 5432
```

(Optional) Seed the database with demo data

```
bun --filter @repo/db seed
```

C. Running the Project

Start all services at once

```
bun dev
```

This command automatically:

- Starts PostgreSQL via Docker Compose (if not running)
- Starts the API server on http://localhost:8787
- Starts the React frontend on http://localhost:5173

D. Interacting with the Project

Access URLs:

Frontend application	http://localhost:5173
API server	http://localhost:8787
API info endpoint	http://localhost:8787/api
Health check	http://localhost:8787/health
Drizzle Studio (DB GUI)	https://local.drizzle.studio (after `bun --filter @repo/db studio`)

Main user flows:

1. Register/Login: Navigate to http://localhost:5173/login. Enter name, email, and password to register, or email and password to log in.
2. Search for routes (guest or registered): Navigate to /routes. Enter a start and end street name (e.g., "Via Torino" and "Corso Buenos Aires"). The system searches for known paths and displays them on the map.
3. Record a trip (registered): Navigate to /trip-record.
4. Manual mode: Search for streets, pick start/end coordinates on the map, add weather and rating data, review, and save.
5. Automatic mode: Click "Simulate" to generate a trip with OSRM routing, random GPS points, and simulated sensor data. Review detected obstacles, confirm or reject them, then save.
6. View trips: Navigate to /trips to see all recorded trips. Click a trip to view its details and map route.

7. Publish a trip as a community path: From the trip list, click "Make Public" on a trip. The system requires a rating (1–5) before publishing. The trip's route is then available in the community path database.
8. Report path conditions: Navigate to /report. Search for an existing published path, select it, update street statuses, mark obstacles on the map, and submit the report.
9. Browse community paths: Navigate to /paths to see all published paths displayed on the map with their current condition status.

7 | Available acceptance testing infrastructure

Our project **does not** support running the application on:

- Android;
- iOS;
- Android emulators;
- iOS emulators.

As previously mentioned, the system we developed is a web application.

8 | Effort Spent

	Cap. 1	Cap. 2	Cap. 3	Cap. 4	Cap. 5	Cap. 6	Cap. 7	Revision
Caldognetto	1	6	1	1	5	2	1	3
Salone	1	1	2	1	10	1	1	4

9 | References

Hono framework documentation	https://hono.dev/
tRPC documentation	https://trpc.io/docs
Drizzle ORM documentation	https://orm.drizzle.team/
Better Auth documentation	https://www.better-auth.com/
React 19 documentation	https://react.dev/
TanStack Router documentation	https://tanstack.com/router/latest
TanStack React Query documentation	https://tanstack.com/query/latest
Jotai documentation	https://jotai.org/
Leaflet.js documentation	https://leafletjs.com/
Playwright documentation	https://playwright.dev/
Bun runtime documentation	https://bun.sh/docs
shadcn/ui component library	https://ui.shadcn.com/
Tailwind CSS v4 documentation	https://tailwindcss.com/
OSRM (Open Source Routing Machine) API	https://project-osrm.org/docs/v5.24.0/api/
Vitest testing framework	https://vitest.dev/
PostgreSQL 16 documentation	https://www.postgresql.org/docs/16/
Cloudflare Workers documentation	https://developers.cloudflare.com/workers/
Zod schema validation	https://zod.dev/