POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering II

Acceptance Testing Document

Project: Best Bike Paths

Authors: Ianosel Bianca Roberta, Gholami Vajihe, Errigo Simone
Version: 1.0
Date: 08.02.2026
Project Link: Errigo-Gholami-Ianosel (github.com)

Contents

1 | Introduction

1.1. Tested Project

The analyzed project is the **Best Bike Paths (BBP)** System, a web-based crowdsourcing application aimed at supporting cyclists in discovering safer and more comfortable routes, developed by **Huang Shijie, Li Yuqing, and Ma Zeyao** (github.com).
The delivered material includes both the frontend and backend components of the system, as well as a deployable JAR package for the backend.
For the purposes of this Acceptance Test Document, the following reference documents provided by the authors were considered:

- **RASD**: Requirements Analysis and Specification Document
- **DD**: Design Document
- **ITD**: Implementation and Test Document

1.2. Acronyms

This section provides definitions of the acronyms used through out the document, making it easier for readers to understand and reference them.

1.2.1. Acronyms

- **BBP:** Best Bike Paths.
- **RASD:** Requirements Analysis and Specification Document.
- **DD:** Design Document.
- **ITD:** Implementation and Test Document.
- **API:** Application Programming Interface.
- **GPS:** Global Positioning System.

- **JWT:** JSON Web Token.
- **UI:** User Interface.

1.3. Revision History

- Version 1.0 (08 February 2026);

1.4. Document Structure

This document is divided into six chapters, which are organized as follows:

1. **Introduction:** provides an overview of the document, including the project under evaluation, the goals of the analysis, and how the document is organized.
2. **Installation:** describes the installation process, including the steps followed to install and run the system, as well as any issues encountered.
3. **Testing:** details the acceptance test cases executed on the system and their outcomes.
4. **Missing Features, Design Deviations, and Discrepancies:** identifies and discusses any missing features, deviations from the original design, or discrepancies between the intended functionality and the actual implementation.
5. **Conclusions:** summarizes the results of the installation and testing phases, highlighting significant findings or issues.
6. **References:** lists the references and resources used in the creation of the document and the project.
7. **Effort Spent:** details the distribution of work and time spent by each team member throughout the project.

2 | Installation

This section reports the installation process we executed to run the prototype delivered by the group under analysis. We followed the Installation Instructions provided in the ITD (Chapter 6), and we report both the documented procedure and the issues encountered during the setup, together with the corrective steps required to successfully run the system.

2.1. Prerequisites

The ITD lists the following software requirements:

- Java (JDK 8 or higher, JDK 11 recommended)
- MySQL 8.0+ with spatial support,
- Node.js (>= 14),
- npm (>= 6).

During the installation, we confirmed that these prerequisites are necessary.
Moreover, the document states that a Google Maps API key is required, and it explicitly lists several required Google APIs (Maps JavaScript, Routes, Directions, and Geocoding). However, during functional testing, we encountered a minor inconsistency between the documented API requirements and the actual implementation. The location search functionality relies on the Google Places API (Autocomplete), which is not mentioned in the installation requirements section. This dependency is documented elsewhere in the ITD (specifically in Chapter 3 and 4), but it is missing from the explicit list of required APIs in the installation prerequisites, which caused an initial functional failure of the search bar until the Places API was manually enabled in Google Cloud Console.

2.2. Database Setup and Issues

The ITD provides a SQL script located at `backend/sql_file/bbp.sql` and suggests executing it using: `mysql -u root -p < backend/sql_file/bbp.sql`

While the script correctly creates the schema and the tables, the documented procedure is not sufficient to set up the database from scratch in a clean environment. The backend configuration expects a dedicated database user (through environment variables such as `DB_USERNAME` and `DB_PASSWORD`). However, the installation guide does not describe the creation of such user, nor the privilege grants required to import the schema and let the application connect.
When executing the import using the configured application user, the setup fails with a MySQL authentication error (ERROR 1045, access denied), because the user does not exist yet or does not have the required privileges.
To complete the installation, we can set the DB_USERNAME environment variable to `root` or those administrative steps should be performed which were not documented in the ITD:

```
cd backend
sudo mysql

CREATE DATABASE IF NOT EXISTS BestBikePathDB;
CREATE USER IF NOT EXISTS 'tester'@'localhost' IDENTIFIED BY 'bbp';
GRANT ALL PRIVILEGES ON BestBikePathDB.* TO 'tester'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

After creating the user and granting privileges, we imported the schema by executing:

```
mysql -u tester -p BestBikePathDB < sql_file/bbp.sql
```

This procedure successfully completed the database setup and allowed the backend to connect correctly.

2.3. Backend Setup and Environment Variables

The backend is delivered as a runnable Spring Boot JAR. The ITD instructs to export environment variables (`DB_USERNAME`, `DB_PASSWORD`, `GOOGLE_API_KEY`) and then run the JAR from the backend directory using: `java -jar best_bike_path-0.0.1-SNAPSHOT.jar`.
In our execution, a critical usability issue is that the documentation implies a straightforward execution, but it does not clearly emphasize that the environment variables must be loaded in the same shell session where the JAR is launched. In our setup, we used a `.env` file and manually sourced it before starting the backend, otherwise the application fails due to missing configuration values.

Final working procedure:

```
source .env
java -jar best_bike_path-0.0.1-SNAPSHOT.jar
```

The backend starts on port 8080 as stated in the ITD.

2.4. Frontend Setup and External API Inconsistency

The ITD instructs to install frontend dependencies in `frontend/` and start the development server with `npm start`. The ITD also specifies that the Google Maps API key must be placed inside `js/config.js`.
Following the documented procedure, the frontend setup completed successfully. The application loaded as expected in the browser, and the overall frontend initialization process can be considered smooth and straightforward.

2.5. Repository Structure and Configuration Management

Upon inspecting the delivered source code, we identified issues regarding repository organization and configuration management.
The project is split into two main directories, `backend/` and `frontend/`, coherently with the ITD description of the repository-level structure. However, the repository lacks a `.gitignore` file. This omission introduces two major problems.

- **Security risk:** without ignore rules, sensitive configuration files such as `.env` (containing database credentials and API keys) can be accidentally committed to version control. Even if such files were not committed in the delivered snapshot, the repository configuration does not prevent accidental leaks in future commits.
- **Repository bloat and noise:** without excluding folders such as `node_modules/`, dependency files can be tracked by Git, increasing repository size and raising the probability of merge conflicts and irrelevant diffs.

3 | Testing

In this chapter, we present the results of the functional tests performed on the delivered prototype. We focused on the main user stories described in the RASD. For each feature, we report the Expected Behavior, the Steps performed, the Result, and a detailed list of Anomalies and Notes found during execution.

3.1. User Registration and Authentication

These tests assess the implementation of user registration, authentication, access control, and basic profile management functionalities. The tests focus on the correctness of the registration and login flows, input validation, duplicate handling, session management, and access to protected resources.

Expected Behavior

The system should allow new users to register using valid credentials (username, email, and password), prevent the creation of accounts with duplicate identifiers, and enforce basic input validation constraints. Registered users should be able to authenticate, access protected routes, update their profile information, and log out securely. Unauthenticated users should be prevented from accessing restricted areas of the system.

Steps

1. Open the registration page.
2. Register a new user using valid credentials.
3. Attempt to register users with duplicated email and duplicated username.
4. Attempt registration using invalid email formats and weak passwords.
5. Perform login attempts with valid credentials, invalid credentials, and non-existing users.

6. Access protected routes both before and after authentication.
7. Edit profile information and attempt updates with duplicated data.

Result: PARTIAL SUCCESS

Core authentication flow: The core authentication mechanisms are correctly implemented. User registration with valid credentials works as expected, and registered users can successfully log in. Login attempts with non-existing users or incorrect passwords are consistently rejected, and the system always returns a generic "invalid email or password" message. This behavior represents a good security practice, as it prevents user enumeration.

Duplicate handling and access control: Duplicate email addresses are correctly detected and rejected during registration, but duplicate usernames are allowed. Access control for protected routes is properly enforced, prompting unauthenticated users to login while allowing authenticated users to access restricted sections.

Inconsistent validation rules during registration: During the registration phase, the system allows the creation of multiple users with the same username, despite enforcing username uniqueness during profile updates.

Missing input validation: Input validation during registration is severely lacking. The system accepts invalid email formats and extremely weak credentials (e.g., username: `a`, email: `a`, password: `a`), both on the client side and on the server side. The absence of minimal validation constraints represents a significant weakness in the user management implementation and exposes the system to potential security and data quality issues.

3.2. Authentication, Session Handling, and Access Control

This set of tests evaluates authentication enforcement for guest users, session persistence, logout behavior across navigation and browser tabs, and authorization checks on protected backend resources. In addition, we performed targeted security tests to verify that users cannot access resources belonging to other users.

Expected Behavior

Guest users should be prevented from using limited features and should be prompted to log in. After logout, the session must be invalidated and the user should not be able to access or view protected content through browser back navigation. Sessions should

behave consistently across tabs (logging out in one tab should invalidate the session in all tabs). On the backend, authenticated users must only be able to access and modify their own trips and reports. Any attempt to access another user's resources (e.g., by guessing an ID in the URL) should be rejected with an authorization error (e.g., 403).

Steps

1. As a guest user, attempt to access limited features and observe the system behavior.
2. Log in and navigate across protected pages (e.g., profile, record).
3. Log out and use the browser back button to check whether protected pages can still be viewed.
4. Verify session persistence by closing and reopening the browser after login.
5. Open two tabs with an authenticated session. Log out from one tab and observe the other.
6. Perform backend authorization checks:
   (a) Call protected endpoints without a Bearer token.
   (b) Call resource endpoints using a valid token belonging to a different user (e.g., `GET /api/reports/{reportId}` with another user's token).

Result: PARTIAL SUCCESS

Guest gating and basic session handling: The system correctly prompts guest users to log in when attempting to use limited features. This behavior is coherent with access control expectations. However, selecting the *Stay* button causes a full page refresh, which is functional but negatively impacts usability, since the page content is reloaded unnecessarily.
The login session is persistent: after authenticating, closing and reopening the page preserves the logged-in state. Logging out in one tab also logs the user out in another tab, which is correct and indicates consistent session invalidation across browser contexts.
Logout and navigation edge case: After logout, using the browser back button may still display previously visited protected pages (e.g., the profile page) even though the session has been invalidated. From those pages, attempting protected actions triggers authentication-related failures. This indicates that logout invalidation works at the backend level, but client-side state and/or navigation handling is incomplete (e.g., missing cache/state clearing or missing redirect logic after logout).

Backend access control on protected endpoints: Calls to protected endpoints without a Bearer token are correctly rejected, confirming that authentication is enforced at the API level for those routes.
Critical security issue: Horizontal Privilege Escalation (IDOR) ***(FAIL)*** Authorization checks are missing for multiple endpoints that expose user-owned resources. In particular, requesting a report by ID (e.g., `GET /api/reports/{reportId}`) with a valid token belonging to a different user returns the full report details instead of rejecting the request. This indicates an Insecure Direct Object Reference (IDOR) vulnerability: any authenticated user can read other users' reports if the numeric ID is known or can be guessed.
Similar missing-authorization patterns were identified for additional endpoints related to trips and reports, where operations are performed based only on the provided `tripId` or `reportId`.
This vulnerability breaks data confidentiality and allows unauthorized access to other users' data. It is a high-severity access control flaw.

3.3. Path Search and Navigation

The following tests evaluate the path search functionality and route visualization workflow, including the correctness of route computation, the availability of multiple route options, UI responsiveness, and robustness of the Google Maps integration.

Expected Behavior

The system should allow the user to search for a route between two locations and visualize the computed route(s) on the map, including route details (e.g., distance, duration, score). If multiple routing strategies are provided, the UI should present them consistently and avoid invalid or empty results. The Google Maps interface should load reliably during normal usage. User location visualization should be coherent with the application's permission model.

Steps

1. Use the search interface to search for a path between two locations.
2. Click the displayed route to inspect route details.
3. Compare the routes returned by the available strategies (balanced, fastest, safest).

4. Search for very short routes and inspect the returned results.
5. Repeat address searches multiple times to assess robustness of the autocomplete and map loading.
6. Perform simultaneous searches from different browsers/accounts to check concurrency.

Result: PARTIAL SUCCESS

Core route search and visualization: Path search works correctly in normal conditions. The map is displayed with the computed path, and clicking on the route shows route details. The UI provides three routing strategies (balanced, fastest, safest).
Ghost results on short routes: When searching for very short routes, the system sometimes returns invalid "ghost" results mixed with valid ones. These entries show 0.0 km distance, 0 minutes duration, and a score of 0. This indicates missing filtering or validation of route results before presenting them to the user.
Intermittent Google Maps loading error: During address search (autocomplete dropdown), an intermittent Google Maps error was observed: "Questa pagina non carica correttamente Google Maps". The error appears non-deterministically, as the same functionality may work correctly moments before failing. This reduces robustness of the search experience and suggests instability in the Maps integration layer.
User position visibility and permission model: While the search page does not consistently show the user's current position, this behavior aligns with the application's permission flow, since location access is requested only when starting a trip recording in the Record feature. After granting location permissions in the recording workflow, the user position becomes visible on the map, making orientation clearer. Therefore, the absence of a position marker during pure search should be considered a UX limitation rather than a functional bug.
Route selection and workflow clarity: Selecting a route allows preview on the map. Starting a trip from the search view is not supported, but this appears to be an intentional design choice, as the search feature is used for visualization only, while trip recording is handled in a dedicated section.
Concurrency: Searching for routes simultaneously from two different browsers/accounts works correctly without interference.

3.4. Trip Recording

These tests analyze the trip recording functionality, including location permission handling, path tracking, trip termination, and post-trip report generation. The focus is on correctness of recorded paths, robustness in edge cases, and usability of the reporting workflow.

Expected Behavior

The system should request location permissions before starting a trip and record the user's movement accurately from the initial GPS position. Trip recording should handle edge cases such as very short or zero-length trips gracefully. At the end of a trip, the system should allow users to review and report encountered obstacles in a way that preserves data accuracy and usability.

Steps

1. Navigate to the recording page.
2. Press the "Start" button and observe location permission handling.
3. Grant location permissions when requested.
4. Observe the recorded path on the map while stationary or moving.
5. Stop the trip and inspect the generated report draft.
6. Start and immediately stop a trip without moving.

Result: PARTIAL SUCCESS

Location permission handling: The application requests location permissions only when the user presses the "Start" button.
Initial position handling and map artifact: After granting location permissions, the recorded path does not consistently start from the actual initial GPS position. Instead, the map sometimes draws a straight artifact line connecting the previous map center (random default point) to the first retrieved GPS coordinate after authorization. This occurs because the application initializes the trip polyline before having access to the user's real location, and then connects the initial placeholder point to the real position once it becomes available. This behavior creates a misleading visual representation of the trip and indicates incorrect initialization of the starting point.

Continuous tracking with simulated GPS: Using a browser extension to simulate GPS movement (Route Pilot on Chrome), the system is able to record a moving path and append multiple points over time. This confirms that the core tracking pipeline works for non-trivial trips, and that the main issues are concentrated around trip initialization and edge-case handling.

Trip termination and report generation: Stopping an active trip correctly generates a draft report, allowing the user to add obstacles. However, obstacles can only be added at the end of the trip by selecting points on the map. This post-trip reporting approach increases cognitive load and may reduce data accuracy, as users are required to remember obstacles encountered earlier during the trip.

Map cleanup after trip completion: After ending a trip and submitting the related report, the previously recorded polyline remains visible on the map instead of being cleared. However, when another trip is started, the old path is removed and the new trip is recorded cleanly.

Zero-length trip handling *(FAIL)*: Starting and immediately stopping a trip without any movement results in a client-side JSON parsing error. This indicates that the backend does not handle empty or zero-length trip data gracefully, leading to a system failure instead of a controlled empty result.

Observed inconsistencies in score impact: After submitting multiple reports on the same path, no observable change was detected in the route score shown in the search feature (e.g., remaining constant at 80).

3.5. Manual Reports

These tests focus on the manual report functionality, including report creation, map interaction, input validation, and consistency of user statistics. In particular, one of the main goals of this test is to verify that manual reports can be created correctly and that they have the expected impact on route scores, as described in the requirements documentation.

Expected Behavior

The system should allow users to manually create reports by defining a path and selecting one or more obstacles on the map.

Steps

1. Create a manual report by defining a path on the map.
2. Attempt to submit the report with and without selecting obstacles.
3. Submit a report with obstacles but without selecting a valid map point.

Result: PARTIAL SUCCESS

Manual report creation workflow: The manual report workflow allows the user to define an origin and a destination, manually adjust the path on the map, and then add obstacles along that path. The path creation phase works as expected and the route can be visually defined through map interaction.
Report submission behavior: When attempting to submit a manual report with obstacles, the system enforces the selection of a point on the map. However, even after correctly selecting obstacle positions, the report submission fails with a generic saving error. Conversely, submitting a manual report without selecting any obstacle succeeds without issues. The empty report appears to be intended behavior, to signal a "positive" confirmation of a path and increase its score. However, no observable effect on the route score was detected in the search feature (e.g., the score remained constant at 80 after multiple submissions).

3.6. Profile Page and User Statistics

This set of tests evaluates the profile page implementation, with a focus on the visibility and usefulness of user-related information and statistics.

Expected Behavior

According to the RASD, the profile-related use cases indicate that the system should present meaningful user information, including aggregated statistics such as total distance traveled and number of submitted reports. Such information is expected to provide users with an overview of their activity history.

Steps

1. Log in as an authenticated user.
2. Navigate to the profile page.

3. Inspect the information and statistics displayed for trips and reports.

Result: PARTIAL SUCCESS

Minimal profile information: The profile page is accessible and displays user-related data. However, the interface only presents simple lists of items (e.g., "My Trips" and "My Reports") without providing detailed views or interactive exploration of individual entries.
Missing aggregated statistics: Aggregated user statistics such as total distance traveled or meaningful summaries of report activity are not displayed. The interface does not provide any form of aggregation, visualization, or historical overview beyond a flat list of elements.
Low informational value and engagement: The current implementation offers limited informational value to the user. The lack of statistics, summaries, or visual elements reduces the perceived usefulness of the profile page. Overall, the profile page appears to be implemented at a minimal level and does not fully reflect the expectations set by the requirements documentation.

3.7. Test Infrastructure and Coverage

This section analyzes the testing approach adopted by the development team, focusing on the presence (or absence) of automated testing infrastructure and test coverage.

3.7.1. Absence of Automated Test Suite

We found no evidence of automated unit tests, integration tests in the delivered codebase. The backend repository contains a single placeholder test class:

`BestBikePathApplicationTests.java`

This class contains only the default Spring Boot context load test generated by the framework initializer, which verifies that the application starts successfully but provides no functional validation.
No custom test classes were found for any of the core services, controllers, repositories, or utility classes, despite the implemented business logic (path segmentation, data merging, scoring algorithms).
Similarly, the frontend code contains no test files, test configurations, or references to testing frameworks.

3.7.2. Manual Testing via Postman

Based on the ITD documentation and the absence of automated tests, we infer that the development team relied exclusively on manual testing using tools like Postman for API validation.
While manual testing is acceptable for rapid prototyping and exploratory validation, the complete absence of automated tests represents a significant quality assurance gap for several reasons:

- **Difficult bug detection:** Without automated tests, there is no mechanism to detect when code changes introduce bugs in previously working functionality. This is particularly problematic for complex algorithms like path segmentation and score calculation.
- **Limited edge case coverage:** Manual testing is typically focused on happy paths and common scenarios. Edge cases, boundary conditions, and error handling paths are often under-tested, as evidenced by the zero-length trip crash and ghost route results we discovered during functional testing.
- **No validation of business logic correctness:** Core algorithms such as the Data Merging weighted average, the Freshness Algorithm (which is not implemented), and the Score Calculation penalty system cannot be verified for mathematical correctness without dedicated unit tests.
- **Poor documentation of expected behavior:** Well-written tests serve as executable specifications that document how components should behave. Without tests, understanding intended behavior requires reading implementation code and inferring intent.

3.8. Testing Summary

Overall, the tested prototype implements the main user-facing features described in the documents. However, multiple functionalities are only partially implemented or lack consistency between frontend behavior, backend validation, and data semantics.
Critical issues were identified in access control, trip recording edge case handling, and manual report submission logic. While the system demonstrates a functional baseline, these issues significantly impact data reliability, security, and user trust.

4 | Missing Features, Design Deviations, and Discrepancies

This chapter analyzes missing functionalities, partial implementations, and discrepancies between the Delivered Documets and the delivered prototype. The goal is to identify unjustified omissions and to highlight design-to-implementation gaps that affect correctness, completeness, or system quality.

4.1. Missing or Partially Implemented Features

This section highlights features that were presented as part of the system's functionality but are missing, incomplete, or only partially implemented in the delivered prototype, without a clear justification. The focus is on the gap between declared capabilities and the actual behavior and visibility of those features from a user and system perspective.

Aggregated User Statistics

The system is expected to provide users with an overview of their activity through aggregated statistics such as total distance traveled, total time, or number of reports.
In the delivered prototype, the profile page only displays raw lists of trips and reports, without any form of aggregation or summary.
As a result, users cannot obtain a meaningful overview of their historical activity. This represents a missing feature, since the aggregation logic is neither implemented nor explicitly excluded, and the current implementation only exposes unprocessed data.

Email Verification and Notification Service

The design documents describe an email verification step via a Notification Service. In the delivered prototype, however, no email verification mechanism is present and no email service is configured during installation. There's also no trace of the notification service, which should have processed notifications triggered by other components.
Moreover, the ITD does not justify the exclusion of email verification, despite describing

the registration flow as “complete”. As a result, accounts are created without verification, which is inconsistent with the planned design.

Weather Service Integration

The system is expected to associate meteorological data with recorded trips. In the delivered prototype, weather-related logic exists at the backend level. A WeatherService retrieves data from an external provider, and weather information is stored when a trip is completed.
However, this information is never exposed to the user. Weather data is not displayed in the profile, trip history, or any user-facing interface, making the feature effectively invisible from a functional perspective. As a result, the feature is partially implemented but not usable or observable by the user, and its presence does not provide any practical value.

4.2. Infrastructure Simplification

The development team initially proposed a deployment architecture oriented toward scalability and robustness, including components such as firewalls and load balancers to manage traffic and distribute requests across multiple instances.
In the delivered prototype, this architectural vision is not reflected. The system is deployed as a single, locally executed Spring Boot application, without any external security layer, request routing, or instance replication.
While this simplification is acceptable in the context of an academic prototype, it represents a clear deviation from the intended deployment model. The implemented infrastructure does not support horizontal scaling, load distribution, or fault tolerance as originally envisioned, and therefore does not align with the scalability and robustness goals stated during design.

4.3. Missing Core Logic and Algorithmic Discrepancies

Several algorithmic components were presented as central to the system’s value proposition, but are not realized in the delivered implementation.
In particular, the system was intended to evaluate route quality using a dynamic scoring approach, accounting for factors such as report freshness and continuous updates over time. In practice, no freshness-based or time-decay logic is implemented. Route scores

are computed using static weight combinations and fixed penalties, resulting in a simplified and non-adaptive evaluation model.
Similarly, the system was designed to automatically detect obstacles based on sensor data collected during trips, reducing the burden on users and increasing data objectivity. In the delivered implementation, sensor data is recorded but never analyzed to generate obstacles or reports automatically. All obstacle reporting remains entirely manual and post-trip.
Automated report generation from sensor data, which was meant to populate draft reports with pre-detected events, is also absent. Draft reports contain no automatically inferred information and rely solely on user input.
Finally, route visualization was intended to convey quality and safety information through visual cues such as color-coded segments. The current user interface renders all routes using a uniform visual style, preventing users from distinguishing between routes based on their underlying scores.
Overall, the implemented system reflects a significantly simplified version of the originally proposed logic. Several core mechanisms that were meant to differentiate the system—dynamic scoring, automation, and rich visual feedback—are either missing or reduced to static placeholders, resulting in a functional but substantially less ambitious solution than initially proposed.

4.3.1. Google Routes Dominance Over Crowdsourced Data

The system was designed as a crowdsourcing platform where community-contributed data should be the primary source of route recommendations, with Google Maps API serving as a fallback for areas without user coverage.
In the delivered implementation, Google routes receive hardcoded static scores (80 for primary route, 70 for alternatives) that are never adjusted based on obstacle data or community reports. This creates an unfair comparison where Google routes consistently outrank internally computed paths, even when the latter have higher quality scores derived from actual user feedback.
The RouteService sorts all routes by score in descending order, but since Google routes maintain their fixed score of 80, they almost always appear first in the result list, effectively making the system Google-first rather than crowdsourcing-first as intended.
Moreover, there is no mechanism to overlay Google polylines with internal BikePath segments to apply obstacle penalties or aggregate user-reported conditions. This means Google routes are presented as-is without any enrichment from the crowdsourced knowledge base, defeating the purpose of community data collection.

4.4. Security and Quality Issues

This section summarizes security-related and quality issues identified during code inspection and system usage. While some of these issues may be acceptable in a prototype context, they still represent deviations from good engineering practices and may introduce risks if the system were to be extended or deployed in a real-world scenario.

Hardcoded Secrets

Sensitive credentials are embedded directly in the source code and configuration files. In particular, the JWT secret key is hardcoded in the backend code, and the Weather Service API key is stored directly in the application configuration.
Although comments in the code acknowledge that this approach is not suitable for production, embedding cryptographic secrets and external service keys in the codebase represents a security anti-pattern. This practice weakens overall system security, makes secret rotation difficult, and increases the risk of credential leakage if the repository is shared or exposed.

UI Responsiveness Issues

The application exhibits several layout and usability problems when the screen resolution changes. On mobile-sized viewports, some interface elements become partially or completely hidden, overlap with other components, or are no longer clickable.
These issues reduce usability on mobile devices and suggest that responsive layout handling was not fully addressed in the delivered implementation.

5 | Conclusions

The system is clearly designed for cyclists and relies heavily on GPS data and mobile device sensors. For this reason, the application should be considered inherently *mobile-first*. Cyclists are expected to interact with the system through a smartphone during or immediately after a trip, rather than from a desktop environment.

From this perspective, ensuring full responsiveness, usability, and interaction correctness on mobile devices should have been a primary design and implementation concern. However, several features (manual report creation, map interaction, and general navigation) exhibit usability issues on small screens, and some interactions become difficult or impossible on mobile-sized viewports. This limits the practical usability of the system in its primary usage context.

Some design choices, while potentially intentional, reduce the intuitiveness of the user experience. For example, the path search feature is limited to static visualization and does not allow users to initiate a trip following a selected route. Although turn-by-turn navigation is not required, users may reasonably expect to view or follow a selected route dynamically while moving.

Similarly, report creation is restricted to the end of a trip. This post-trip-only reporting model negatively impacts the concept of data freshness, as obstacles may be removed or repaired shortly after being encountered. Additionally, requiring users to remember and manually pinpoint obstacle locations on the map after a delay increases cognitive load and reduces report accuracy. The same limitations apply to manual reports, which further weaken the temporal relevance of submitted data.

A significant portion of the identified issues stems not only from missing or partial implementations, but from a lack of explicit justification for these omissions. Several features described in the RASD, DD, and partially referenced in the ITD are either simplified, missing, or only partially exposed to the user.

While scope reductions and simplifications are acceptable in an academic prototype, they should be explicitly documented in the ITD. Leaving features partially implemented or silently removed introduces ambiguity and makes it difficult to assess whether the behavior is intentional or incomplete.

The evaluation also highlighted the absence of automated testing. No evidence of unit

tests, integration tests, or automated validation frameworks (e.g., JUnit-based testing) was found. While manual testing was clearly performed, the lack of automated tests reduces confidence in the robustness of the implementation and makes regression detection difficult as the system evolves.

Despite the identified issues, the system demonstrates a clear and consistent separation between user roles. The distinction between guest users and registered users is correctly enforced. This aspect of the implementation closely follows the defined scope and actor responsibilities and represents a solid foundation for future extensions.

Overall, the delivered prototype provides a functional baseline that implements the core features of the system. With clearer scope documentation, a stronger focus on mobile-first design, and more rigorous testing practices, the project could be substantially strengthened and aligned more closely with its original design goals.

6 References

6.1. Reference Documents

The preparation of this document was supported by the following reference materials:

- Assignment specification for the ITD of the Software Engineering II course, held by professors Matteo Rossi, Elisabetta Di Nitto, and Matteo Camilli at the Politecnico di Milano, Academic Year 2025/2026 [7];
- Slides of the Software Engineering II course available on WeBeep [8].
- The delivered material of the Best Bike Paths project, including the RASD, DD, and ITD documents provided by the authors Huang Shijie, Li Yuqing, and Ma Zeyao [2–4].

6.2. Software Used

The following software tools have been used to support the development of this project:

- **Visual Studio Code**: editing of source code and documentation (LaTeX), with project-wide search and formatting support [6].
- **LaTeX**: typesetting system used to produce the final RASD document in a consistent format [5].
- **Git**: version control used to track changes and support collaborative development [9].
- **GitHub**: remote repository hosting and collaboration platform used for versioning, reviews, and issue tracking [1].

6.3. Use of AI Tools

AI tools were used during the project as supporting tools, in the same way as other software adopted in the development process. Their role was not to automatically generate

content, but to help improve the clarity, structure, and overall quality of the documentation.

6.3.1. Tools Used

The AI tools employed during the project were:

- Gemini
- ChatGPT

6.3.2. Typical Prompts

AI tools were queried using prompts such as:

- "Rephrase this description to make it clearer."
- "Does this explanation of the error encountered contain any ambiguities?"
- "Suggest alternative wording for this paragraph to improve readability."

6.3.3. Input Provided

The input given to AI tools consisted mainly of:

- Early drafts of paragraphs.
- Short text fragments requiring clarity checks.
- Sections with repeated structure where consistent wording was needed.

6.3.4. Constraints Applied

When using AI tools, the following constraints were strictly enforced:

- Preserve the intended meaning of the original text.
- Avoid introducing new design decisions or assumptions.
- Maintain terminology aligned with the definitions provided in this document.

6.3.5. Outputs Obtained

The interaction with AI tools resulted in:

- Clearer or more concise formulations of existing statements.
- Identification of potentially ambiguous sentences.
- Terminology suggestions to improve internal coherence.

6.3.6. Refinement Process

All AI-generated outputs were subject to a manual refinement process that included:

- Critical review of all suggestions.
- Verification against the original intent to avoid unintended changes.
- Manual integration to ensure consistency with the overall writing style.
- Alignment checks with established terminology and definitions.

7 | Effort Spent

This section provides a breakdown of the number of hours each group member dedicated to completing this document. The work distribution is tracked per section and task.

Section	Ianosel Bianca	Simone Errigo	Vajihe Gholami
Documents Inspection	2 hours	2 hours	2 hours
Installation Procedure	1 hours	1 hours	1 hours
Test Case Execution	4 hours	4 hours	4 hours
Document Preparation	3 hours	3 hours	3 hours

Table 7.1: Time spent

Bibliography

[1] GitHub Inc. Github. Online platform, 2025. https://github.com/.

[2] S. Huang, Y. Li, and Z. Ma. Design document, Academic Year 2025/2026.

[3] S. Huang, Y. Li, and Z. Ma. Implementation and test document, Academic Year 2025/2026.

[4] S. Huang, Y. Li, and Z. Ma. Requirements analysis and specification document, Academic Year 2025/2026.

[5] LaTeX Project Team. Latex: A document preparation system. Document preparation system, 2025. https://www.latex-project.org/.

[6] Microsoft. Visual studio code. Source code editor, 2025. https://code.visualstudio.com/.

[7] M. Rossi, E. Di Nitto, and M. Camilli. Software engineering 2 itd assignment specification, Academic Year 2025/2026.

[8] M. Rossi, E. Di Nitto, and M. Camilli. Slides of the software engineering 2 course. WeBeep platform, Academic Year 2025/2026.

[9] Software Freedom Conservancy. Git. Version control system, 2025. https://git-scm.com/.

List of Tables