POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

ATD

Best Bike Paths

Authors: **Matteo Caldognetto – Claudia Salone**

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

Contents

1 | Analyzed Project

A. Authors

The authors of the analyzed prototype are:

- Kaifei Xu
- Shinuo Yan
- Yanglin Hu

B. Repository

The repository of the analyzed project can be found at this link

C. Reference documents

The following documents were considered during the acceptance test:

- Assignment document A.Y. 2025/2026 ("Requirement Engineering and Design Project: goal, schedule, and rules"),
- Assignment document A.Y. 2025/2026 ("I&T assignment goal, schedule, and rules"),
- Requirement Analysis and Specification Document - BBP: Best Bike Paths Software, version 2 (referenced as "RASD"),
- Design Document - BBP: Best Bike Paths Software, version 1 (referenced as "DD"),
- Implementation and Test Deliverable - BBP: Best Bike Paths Software, version 1 (referenced as "ITD")

2 | Installation setup

A. Installation guide

The installation guide presents a dual approach: a recommended Cloud deployment and a local installation.

A.1. Cloud Testing

The cloud version is the recommended method for immediate testing as it requires no local installation. You can just click on a link. However, since it uses a free hosting service, the backend server goes into a "sleep" mode after being inactive. This means every time you access the site, it may take some minutes to start up. It might look like the site is broken, but it's just slow to start.

A.2. Local Installation

The backend runs correctly when following the provided instructions. However, even when strictly following the documented steps, we encountered the following error while starting the frontend:

```
sh: .../frontend/node_modules/.bin/vite: Permission denied
```

Although not mentioned in the documentation, we resolved the issue by granting execute permissions:

```
chmod +x node_modules/.bin/vite
```

After applying this fix, we were able to install dependencies and run the frontend.

A.3. Discrepancies

We observed consistent differences between the cloud and local versions, both in terms of UI/UX and business logic. This makes the evaluation less reliable, as the same test case may yield different outcomes depending on the deployment.

3 | Acceptance test cases

A. Test strategy

Since the cloud version is the officially recommended approach, we used it as our primary test environment. We derived the acceptance tests from the use cases defined in the RASD and, for the most relevant ones, attempted to reproduce the expected flow of events, verifying outcomes and the handling of exceptions and edge cases.

We did not find automated tests (e.g., unit/integration tests) that could be executed to validate the system behavior.

A.1. Registration and Login

The app doesn't have separate pages for signing up or logging in; you just enter a username to get started. A major problem we found is that the system doesn't check if a username is already being used. This means multiple people can use the exact same name, which is a big security risk as it could lead to users seeing or overwriting each other's data.

The authors justified this by stating the system was simplified for the prototype and that username-based identification is enough for a demo in their ITD. However, even though the project plan and the RASD describe a "guest mode" for people who want to view the map without an account , the app currently forces everyone to provide a username. This means the required access for unregistered users is completely missing.

Additionally, after refreshing the page, the user is required to enter the username again (no persistent session).

A.2. Record Trip

According to the requirements, a registered user should be able to track their cycling activities by simply hitting a "Start Trip" button, allowing the app to compute live metrics like distance and average speed . When we tested the application, we spent a significant amount of time looking for this feature, but we could not find any button or workflow to actually initiate a recording. This is a major gap because trip recording is the primary way the system is supposed to gather data from the real world. Without this function, the application fails to serve its main purpose of keeping track of cycling activities for registered users.

A.3. Automatic Obstacle Detection

The RASD explains that the system should automatically detect potholes and other hazards using the smartphone's sensors while a trip is being recorded . Since we were unable to even start a trip session, we could not test this feature in a realistic scenario. Furthermore, the "Auto detection" area we found in the app seems to be completely disconnected from the concept of a "trip." It appears to be linked to individual road segments in a static way, which makes the whole feature very confusing and totally different from the automated, sensor-based logic described in the documentation.

A.4. Confirm/Correct Detected Obstacles

A key part of the BBP system is the validation process, where a user is asked to confirm or reject an obstacle that was automatically detected to avoid false positives. This is supposed to happen either during or after the ride. However, because the trip recording and the automatic detection workflows are missing or broken, this entire confirmation screen is also nowhere to be found. We had no way to validate if the system correctly handles user feedback or if it updates the database as promised in the manual.

A.5. Insert Manual Path Information

The documentation describes a feature where users can manually report the status of a bike path by specifying streets and selecting a quality level like "Optimal" or "Requires Maintenance" . In reality, the application does not allow for this kind of path-based reporting. Instead, you can only pick individual, tiny segments and give them a status one by one. There is no mechanism to link these pieces together into a coherent path, which contradicts the RASD's definition of a path as an ordered sequence of segments.

A.6. View Best Bike Path

When a user asks for a route between two points, the system is supposed to rank the results based on a "Path Score" derived from community feedback and road quality . When we tried this, the routes displayed on the map looked like standard, generic directions you might get from any basic map service. They did not seem to be the specialized bike paths recorded and published by other cyclists, and there was no indication that the "best" path was chosen based on the actual road conditions reported by the community.

A.7. Publish Path Information

The RASD highlights that users should have control over whether their contributions are shared with others through a specific "Publish" workflow . This feature is essential for privacy and data quality. Unfortunately, this functionality is not implemented in the current prototype. There is no way for a user to manage their contributions or toggle the "publishable" status of their reports, which means the community-sharing aspect of the project is effectively missing.

A.8. Merge Multi-user Path Information

The functionality for merging/aggregating path information contributed by multiple users is not implemented.

4 | Additional comments

A. Quality of documentation

A.1. RASD

Upon reviewing the Requirements Analysis and Specification Document (RASD), our overall assessment is that it lacks the depth and level of detail expected for a project of this complexity. Despite being approximately 23 pages of actual content, the document appears rushed and does not provide sufficiently rigorous specifications to effectively support development, verification, and acceptance activities.

From a modelling perspective, the documentation is also weak. The number of diagrams is limited and their level of precision is often inadequate. In particular, the sequence diagrams are incorrect: instead of representing the interactions required to achieve a specific goal (i.e., the sequence of messages among participating components to realize a scenario), they mainly depict generic communications among User, App, Backend, and DB, without capturing the intended objectives of these diagrams.

In addition, key requirements engineering artifacts are missing. Functional requirements are not explicitly stated, and there is no requirements mapping/traceability that links requirements to design elements and implemented features.

Finally, we identified several inconsistencies between what is described in the RASD (in terms of functionality and design) and what is available in the current version of the application. The document also contains an editorial inconsistency in the headers, where the project is referred to as "Travlendar+" instead of "Best Bike Paths".

A.2. DD

Similar issues apply to the Design Document (DD). Overall, it is too high-level and, in multiple sections, it does not accurately reflect the current implementation, which significantly reduces its value as a reference for validation and consistency checks.

From an architectural documentation standpoint, the component view is problematic: it resembles a class diagram rather than a proper component diagram, and it uses unclear or inconsistent labels, making responsibilities and boundaries hard to understand.

Moreover, the DD contains statements that conflict with the actual system. In particular, Section 2.6 describes the backend as being implemented through microservices, but the delivered backend does not follow a microservice architecture. Similarly, the Pub/Sub pattern is described in the DD but does not appear to be implemented.

Several planned elements are also missing from the DD (or not supported by the implementation), such as the offline strategy and the Admin web dashboard, which are mentioned as part of the envisioned solution but are not presented with concrete design details.

Requirements traceability is also severely compromised: the traceability section is inaccurate, and it omits requirements stated in the RASD, making it impossible to properly map requirements to design choices and implemented features.

Finally, the test plan is overly short and approximate, lacking the level of detail expected to support systematic verification and acceptance activities.

Just like in the RASD, the project name continues to be incorrectly referred to as "Travlendar+".

A.3. ITD

The Implementation and Test Document (ITD) shows a significant disconnection between the design specifications and the delivered system. The implementation fails to reflect the architectural complexity outlined in previous documents.

Critically, the system contradicts the distributed architecture defined in the Design Document (DD), opting instead for a monolithic solution with volatile in-memory storage. This prevents the persistence of historical road quality data, a fundamental requirement of the project.

Functional coverage is also reduced compared to the RASD, with key features like background sensor collection and offline management largely omitted or simplified. Additionally, planned components such as the administrative dashboard are missing.

Testing is superficial, documenting mostly manual scenarios without code coverage metrics or load testing. This lack of rigor is particularly concerning as, upon practical inspection of the delivered application, most core features proved to be non-functional, failing to meet basic operational expectations. Furthermore, we didn't find automated tests that could be executed to validate the system behavior.

B. Quality of code

The backend is implemented as a single file of over 2000 lines, which significantly impacts readability, maintainability, and adherence to a modular/pattern-driven design approach.

On the frontend side, robustness is limited: input validation is weak, invalid inputs are often accepted, and the application can enter error states without clear recovery paths.

5 Effort Spent

	Cap. 1	Cap. 2	Cap. 3	Cap. 4	Revision
Caldognetto	1	2	1	1	1
Salone	1	1	2	1	2