POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering II

Requirement Analysis and Specification Document

Project: Best Bike Paths

Authors: Ianosel Bianca Roberta, Gholami Vajihe, Errigo Simone
Version: 1.0
Date: 23.12.2025
Project Link: Errigo-Gholami-Ianosel (github.com)

Contents

Contents **i**

1 Introduction **1**
1.1 Purpose . . . 1
1.1.1 Goals . . . 1
1.2 Scope . . . 2
1.2.1 World phenomena . . . 3
1.2.2 Shared phenomena . . . 3
1.3 Definitions, Acronyms, Abbreviations . . . 5
1.3.1 Definitions . . . 5
1.3.2 Acronyms . . . 6
1.3.3 Abbreviations . . . 6
1.4 Revision history . . . 6
1.5 Reference Documents . . . 6
1.6 Document Structure . . . 7

2 Overall Description **9**
2.1 Product Perspective . . . 9
2.1.1 Scenarios . . . 9
2.1.2 Domain Class Diagram . . . 13
2.1.3 State Diagrams . . . 14
2.2 Product functions . . . 20
2.3 User Characteristics . . . 23
2.4 Assumptions, dependencies and constraints . . . 24
2.4.1 Dependencies . . . 24
2.4.2 Constraints . . . 24
2.4.3 Domain Assumptions . . . 24

3 Specific Requirements **27**
3.1 External Interface Requirements . . . 27
3.1.1 User Interfaces . . . 27
3.1.2 Hardware Interfaces . . . 27
3.1.3 Software Interfaces . . . 27
3.1.4 Communication Interfaces . . . 28
3.2 Functional Requirements . . . 28
3.2.1 Requirements . . . 28
3.2.2 Use Case Diagrams . . . 31
3.2.3 Use Cases . . . 33
3.2.4 Requirement Mapping . . . 72
3.3 Performance Requirements . . . 72
3.4 Design Constraints . . . 72
3.4.1 Standards Compliance . . . 72
3.4.2 Hardware Limitations . . . 72
3.5 Software System Attributes . . . 73
3.5.1 Reliability . . . 73
3.5.2 Availability . . . 73
3.5.3 Security . . . 73
3.5.4 Maintainability . . . 73
3.5.5 Portability . . . 74

4 Formal analysis using alloy **75**

5 Effort Spent **97**

Bibliography **99**

List of Figures **101**

List of Tables **103**

1 | Introduction

In recent years, cycling has become an increasingly popular and sustainable means of transportation. However, cyclists often struggle to find safe routes and reliable information about bike path conditions. The **Best Bike Paths (BBP)** system aims to address this issue by offering a platform where users can record their rides, explore optimal routes, and contribute with updated information about cycling paths and obstacles.

1.1. Purpose

The purpose of this document is to define, analyze, and specify the requirements of the **Best Bike Paths (BBP)** system in a precise and complete manner. It establishes a shared understanding of the system's goals, requirements, and boundaries, describing both the functional behavior and the assumptions about the real-world environment in which the system operates. This document serves as a common reference for all project stakeholders, ensuring a clear and shared understanding of what the system must accomplish and under which constraints.

1.1.1. Goals

The following **goals** outline the desired properties of the real world that the Best Bike Paths (BBP) system is intended to achieve. They reflect the stakeholders' expectations about how cycling activities and information should be managed and shared, independently of any technical or implementation detail.

[G1] **Users can access accurate and up-to-date information about bike paths**, including their status and possible obstacles, to support safe and informed cycling.

[G2] **Users can discover optimal and safe biking routes** between two locations based on the quality and effectiveness of available bike paths.

[G3] **Users can safely navigate along selected bike paths**, being informed in real-time about their position, nearby obstacles, and path closures.

[G4] **Logged-in users can track their cycling activities over time**, allowing logged-in users to monitor their trips and review their past performances.

[G5] **Logged-in users can contribute to the community knowledge** by creating, updating, or confirming information about existing bike paths.

[G6] **Logged-in users can view general and per-trip statistics** derived from their recorded cycling activities, supporting awareness and motivation in their performances.

1.2. Scope

Best Bike Paths (BBP) is a software system designed to support users in **creating**, **exploring**, and **sharing** information about bike paths and cycling activities. Users can **register** on the platform and may optionally provide personal data to enable personalized statistics for their cycling activities.

Logged-in users have access to a personal history of their trips and can view general **statistics** accumulated over time, as well as detailed **information** about each specific **trip**, such as the total distance covered, average speed, and other performance metrics. When available, **additional information** such as weather conditions, temperature, and wind speed can be **enriched** through **external services**, provided the logged-in user owns a compatible device capable of capturing such data. Furthermore, logged-in users can contribute to the community by **publishing** information about bike paths, **reporting** their current **status** and the presence of obstacles, to help other users identify the best and safest routes. These data can be entered **manually**, by specifying details such as the streets composing the path and their condition, or **automatically**, through the acquisition of data from the user's mobile device (e.g., GPS, accelerometer, and gyroscope sensors). In automatic mode, BBP can infer that a user is biking based on their speed, collect GPS information to reconstruct the followed path, and detect irregular movements that may suggest the presence of potholes or surface issues. Since this process may generate false positives, logged-in users will have to confirm or correct the automatically acquired data before storing them.

The condition of a bike path can assume several states, such as "optimal," "medium," "sufficient," "requires maintenance," or "closed." BBP **aggregates** and **merges** publishable information about paths obtained from multiple users, taking into account the **freshness** of the data and the number of consistent confirmations received. For instance, if a path receives multiple reports within a similar timeframe, BBP determines its status based on

the most supported and up-to-date information.

Any user can search for and visualize optimal bike paths between two points on a map. Paths are **ranked** according to their status and overall effectiveness in connecting the selected origin and destination. While following a route, the user can visualize their position on the map. Additionally, the system may provide **notifications** about obstacles or particular conditions along the path.

1.2.1. World phenomena

The phenomena relevant to the system can be grouped into two main categories. This section focuses on non-shared world phenomena, that is, phenomena occurring in the external environment which are beyond the machine's direct control.

[W1] - Availability and accuracy of external services (e.g., weather data providers, map services).

[W2] - The user has an external device to measure additional statistics.

[W3] - The road's physical condition can evolve.

[W4] - The user may deviate from the suggested path.

[W5] - The user may experience unexpected events during the ride (e.g., accidents, weather changes).

[W6] - The user rides a bike.

1.2.2. Shared phenomena

- **World controlled:** Shared phenomena triggered by the world

[SP1] - A guest user registers to the system.

[SP2] - A logged-in user logs into the system.

[SP3] - A logged-in user logs out of the system.

[SP4] - A logged-in user creates a new bike path.

[SP5] - A logged-in user decides to make a created bike path public or private.

[SP6] - A logged-in user deletes one of their created bike paths.

[SP7] - A user selects an origin and a destination to explore bike paths.

[SP8] - A user selects a specific bike path to view its details.

[SP9] - A logged-in user sends reports about a bike path's condition or obstacles.

[SP10] - A logged-in user confirms or rejects a bike path's condition report.

[SP11] - A logged-in user updates personal information.

[SP12] - A logged-in user starts, pauses, resumes, or stops recording a bike trip.

[SP13] - A logged-in user browses a trip history or views statistics.

[SP14] - The sensors transmit their data to the system.

[SP15] - The external services provide additional data (e.g., weather conditions).

- **Machine controlled:** Shared phenomena triggered by the machine

[SP16] - The system recommends a bike path based on its ranking.

[SP17] - The system merges reports about bike path's conditions.

[SP18] - The system updates bike path rankings based on new feedback.

[SP19] - The system presents detected information from sensors to the logged-in user for confirmation, generating a new report only if the user validates it.

[SP20] - The system notifies users about significant changes in bike path conditions.

[SP21] - The system confirms the successful creation or deletion of a bike path to the logged-in user.

[SP22] - The system publishes or hides a bike path according to the logged-in user's sharing preference.

[SP23] - The system visualizes on a map the bike path(s) between a specified origin and destination.

[SP24] - The system reads the GPS positions of a biking user to track their trip.

[SP25] - The system reads data from the device's sensors (e.g., accelerometer, gyroscope) to detect obstacles during a bike trip.

[SP26] - The system retrieves additional data (e.g., weather conditions) from external services when available.

[SP27] - The system provides general and per-trip statistics to logged-in users.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **Bike Path:** A route, defined by collected data, where a proper bike track exists or where road conditions are generally compatible with cycling safety and speed. Path quality is determined by its aggregated status.
- **Path Segment:** A portion of a Bike Path defined between two consecutive waypoints. Each segment may have specific attributes.
- **Path Status:** The overall condition of a Bike Path or Path Segment, as determined by the system's aggregation and merging process. Statuses include Optimal, Medium, Sufficient, and Requires Maintenance.
- **Path Score:** The final numerical ranking value computed by the system, based on the path's aggregated status and effectiveness, used to prioritize and recommend path alternatives.
- **Trip:** A cycling activity recorded by a logged-in user through the BBP application. Each trip includes temporal, spatial, and contextual data (e.g., duration, distance, speed).
- **Report:** A submission of path information by a logged-in user. A report contains details on path status and obstacles and can be either Manually Entered (user-entered) or Automatically Acquired (sensor-acquired).
- **Freshness:** A metric used in the path merging algorithm that measures the recency of a Path Report's submission. Older reports carry less weight in the aggregation.
- **Obstacles:** Features on a path that negatively impact cycling conditions, such as potholes or flooding, as identified by users or automated sensors.
- **Automatic Mode:** The system functionality where the BBP application acquires real-time accelerometer and gyroscope data from a logged-in user's device to automatically generate a Path Report.
- **Manual Mode:** The system functionality where the logged-in user manually enters information about a Bike Path, including its conditions and obstacles, through the application interface.

1.3.2. Acronyms

- **API:** Application Programming Interface.
- **BBP:** Best Bike Paths (The name of the software system).
- **DA:** Domain assumption.
- **DD:** Design Document.
- **GPS:** Global Positioning System.
- **RASD:** Requirement Analysis and Specification Document.
- **UI:** User Interface.
- **UML:** Unified Modeling Language.

1.3.3. Abbreviations

- **[Gn]** - The n-th goal of the system.
- **[Wn]** -The n-th world phenomena.
- **[SPn]** -The n-th shared phenomena.
- **[UCn]** -The n-th use case.
- **[Rn]** -The n-th functional requirement.

1.4. Revision history

- Version 1.0 (23 December 2025);

1.5. Reference Documents

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

- IEEE Standard for Software Requirement Specifications [1];
- Assignment specification for the RASD and DD of the Software Engineering II course, held by professors Matteo Rossi, Elisabetta Di Nitto, and Matteo Camilli at the Politecnico di Milano, A.Y. 2025/2026 [3];
- Slides of the Software Engineering 2 course available on WeBeep [4];

- The paper *Deriving Specifications from Requirements: An Example* by Jackson and Zave [2].

1.6. Document Structure

Mainly the current document is divided into six chapters:

1. **Introduction**: aims to describe the environment and the demands taken into account for this project. In particular, it focuses on the reasons and goals that will be achieved through its development, as well as the context in which the system will operate.
2. **Overall Description**: provides a high-level overview of the system, with a domain model and state/process views that outline the system's behavior without implementation details.
3. **Specific Requirements**: details the functional and non-functional requirements necessary for the system to achieve its goals. In addition, it contains including external interface requirements (UI, hardware, software, communication) and design constraints.
4. **Formal Analysis**: presents a formal description of selected aspects of the world/shared phenomena using the Alloy modeling language.
5. **Effort Spent**: provides a breakdown of time spent on each section of the document.
6. **References**: lists all sources (including Software) and documents used in the preparation of this document.

2 | Overall Description

2.1. Product Perspective

The Best Bike Paths (BBP) system operates within an environment that includes users, mobile devices, and external services. The following sections describe its main interactions and contextual role.

2.1.1. Scenarios

Representative scenarios illustrate how different types of users interact with BBP in realistic situations, showing the system's main functionalities and objectives.

1. **A user wants to use the system**:

   Giuseppe is a former professional cyclist who wants to start cycling again to stay fit. Among the various options available, he decides to use Best Bike Paths (BBP) to track his cycling activities and discover new routes. He chooses BBP because it offers cyclist-oriented features such as creating personalized paths and sharing the current conditions of bike tracks with other users.
   Once he opens the BBP app, Giuseppe realizes that he can already use some functionalities of the system without creating an account, such as searching for bike paths and receiving suggestions based on other users' rankings. However, he soon notices that he cannot record his own cycling activities without registering.
   Therefore, he decides to create an account to take full advantage of all BBP's features. The registration process is simple and quick, requiring only basic information such as name, email, and password. After successfully registering, Giuseppe gains access to the full set of functionalities, including creating personalized routes, recording his cycling sessions, and sharing information about the condition of bike paths with the BBP community.

2. **A user starts to browse and create paths**:

   Irene accesses the BBP app to plan her next cycling trip. From the main page, she

specifies a starting point and a destination. BBP displays on a map all available bike paths connecting the two points. Each path is shown with its current condition based on feedback provided by other users. The paths are ranked according to an overall score computed by the system, taking into account their condition and effectiveness in connecting the selected origin and destination.
After reviewing the suggested routes, Irene realizes that none of them fully meet her preferences. Since she is logged into her BBP account, she decides to create a new custom path.
BBP offers her either a manual or an automatic mode for path creation. She chooses the manual mode, where she can specify the names of the street segments composing her desired route and provide initial information about their condition. Irene carefully enters the street names that form her preferred path, adds a brief description, and saves it to her personal collection.
The newly created path is now available in Irene's account and can be selected for future trips. If she decides to make it publishable later, other users will also be able to discover and use it.

3. **A logged-in user starts a trip**:

   Cesare decides to start a new bike trip. After selecting a suggested bike path, BBP asks him which data acquisition mode he prefers: manual or automatic. Cesare chooses the automatic mode and presses the Start button to begin his ride.
   As he starts, BBP begins collecting data from the sensors available on his mobile device. The GPS continuously updates his position, allowing the system to reconstruct the exact path he follows. At the same time, the accelerometer and gyroscope detect small irregular movements that may indicate the presence of bumps or potholes along the road.
   On the screen, BBP dynamically shows his current position. The system also enriches Cesare's trip data with weather conditions such as temperature, humidity, and wind speed retrieved from an external meteorological service.
   Once he reaches his destination BBP processes the collected data and generates a detailed summary, including total distance, duration, average speed, and other performance metrics that Cesare can review later in his personal activity history.

4. **A logged-in user browse his cycling activities**:

   Luca, a logged-in user of BBP, opens the app to view his previous trips. From the main page, he accesses the "My Activities' section", where BBP displays a list of his recorded trips over time.

Each activity is accompanied by summary information such as the date, distance traveled, duration, and primary location. Luca can sort trips by date or distance and scrolling through the list, he recognizes some of his favorite routes and notices that BBP automatically updates aggregate statistics such as total number of trips, kilometers traveled, and weekly average giving him a general overview of his progress over time.
After identifying a recent trip he's interested in analyzing in more detail, Luca decides to open it to view the details.

5. **A logged-in user access his trip and per trip statistics**:

   Pietro selects his latest trip from his activity list. BBP opens the route details page, displaying the entire route on a map. In the same page, the app presents a complete summary of the recorded data: total distance, duration, average and maximum speed. Thanks to integration with the device's sensors, Pietro can also view the weather conditions he encountered during the trip, including average temperature and wind direction.
   Scrolling through the map, Pietro notices some points where the accelerometer detected abnormal movement.
   Satisfied with his analysis, Pietro closes the trip tab and returns to the general overview of his activities.

6. **A logged-in user wants to insert information about a bike path condition**:

   Giulia is cycling along a local bike path using the manual reporting mode. During her ride, she notices that a section of the route is partially blocked by some fallen branches. Since she wants to help other cyclists avoid potential issues, she decides to report the current condition of the path to the BBP community.
   From the app menu, Giulia selects Report Path Condition and once the interactive map is opened, selects the affected segment, and writes a short description of the obstacle, choosing "requires maintenance" as the current status of that path. After confirming the report, BBP stores the information and sends it for aggregation with other users' data about the same route.

7. **A logged-in user wants to confirm the condition of a bike path**:

   While cycling through a familiar route, Marco receives a notification with a small pop-up alert which states that there is a possible obstacle ahead reported by another user. Marco slows down and observes the road segment. He notices that the obstacle is indeed still present, so he decides to confirm the report. Directly from

the notification, he presses Confirm, and BBP registers his feedback as an additional confirmation for that path condition.

Later, while continuing his ride, Marco receives another notification about a reported bump on the same road, but this time the road surface looks perfectly fine. He therefore selects Reject, indicating that the problem has been resolved or no longer exists.

BBP updates the internal data accordingly. Each confirmation or rejection contributes to refining the aggregated information that determines the official status of that bike path. This way, Marco's participation, along with that of other cyclists, helps the system keep path conditions accurate and up to date for the entire community.

2.1.2. Domain Class Diagram

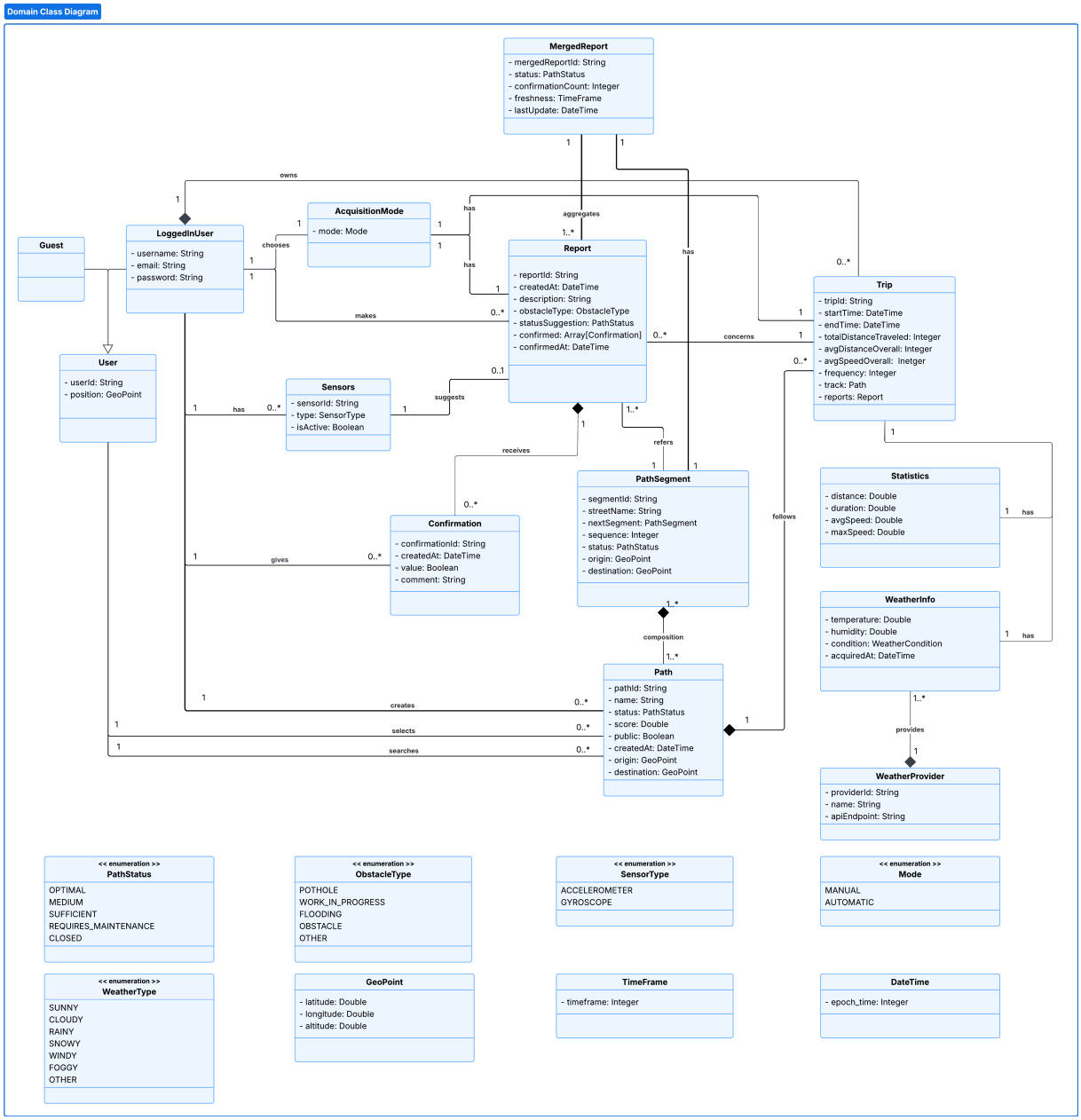


Figure 2.1: Domain Class Diagram

The Domain Class Diagram for the Best Bike Paths (BBP) system illustrates the key entities and their relationships for managing bike paths, trips, and associated information in a unified manner.

The core functionality revolves around the User, who can either register to record activities and contribute data, or remain a guest and only browse path information. A

User is defined by identifiers such as userId and email, along with their current position represented as a GeoPoint.

The central object of the system is the Path, which is defined as a sequence of PathSegments linking an origin to a destination GeoPoint. Each Path has a status (from the PathStatus enumeration), a computed score, and a boolean flag indicating if it is public. A PathSegment is an atomic portion of a route, identified by a segmentId and a streetName, and maintains a reference to the next segment in the path.

Logged-in Users may record a Trip that follows a specific Path. A Trip includes temporal information (startTime, endTime) and metrics such as totalDistanceTraveled. Performance for each trip is summarised in the associated Statistics entity, which records values such as avgSpeed, maxSpeed, and other relevant indicators. To enrich the user experience, a Trip may incorporate WeatherInfo, including temperature, humidity, and a weather condition (WeatherType), which is obtained from an external WeatherProvider.

Data about paths, especially regarding obstacles such as potholes, is collected through Report entities. A User creates a Report using a specific AcquisitionMode (MANUAL or AUTOMATIC). In the automatic mode, the system uses raw data from Sensors (e.g., accelerometer, gyroscope) to detect and suggest possible issues. Each Report specifies an ObstacleType and may include a statusSuggestion.

To maintain accuracy and reduce false positives, each Report must receive one or more Confirmations from users, which validate the information with a boolean value. Confirmations may also be used to validate individual PathSegments.

Finally, the system aggregates multiple Reports about a Path into a MergedReport. This entity provides a consolidated view of the path's condition, including the resulting status, the confirmationCount that supports it, and a freshness interval (TimeFrame). All temporal data within the system is represented using the DateTime entity, which stores values in epoch time for simplicity.

This integrated design ensures a comprehensive system for creating, recording, validating, and sharing bike path information.

2.1.3. State Diagrams

The UML State Diagrams shown below describe the system's main behavioral processes, focusing on the most relevant user interactions. These include starting a trip, creating paths (manually and automatically), and reporting obstacles (manually and automatically).

Starting a trip

The following diagram illustrates the possible states involved when a user starts a trip through the system. This process begins when the user opens the app and decides to start a new trip by inserting the desired start and end points. Once these points are defined, the system activates the Path Suggestion System, which is responsible for generating and ranking candidate bike paths that best fit the user's request.

Inside the Path Suggestion System, the process goes through several internal states. First, the system enters the Waiting for Data state, where it collects environmental and contextual information (such as existing paths and reports). When all required data have been gathered, the system transitions to the Evaluating state to verify their validity and consistency. If the data are valid, the system proceeds to Scoring, where each segment is analyzed and assigned a score based on its quality and reliability. Afterward, in the Merging and Building Candidates states, segments are merged and combined to form complete candidate paths. Finally, in the Ranking state, the system orders these candidates according to computed scores.

If an error occurs at any stage (for example, insufficient data, invalid inputs, or scoring issues), the process moves to a failure path and returns to the Idle state. Otherwise, once the ranking is completed, the system exits the Path Suggestion System, displaying the generated recommendations to the user. At this point, the user enters the Browsing state, where they can inspect all suggested paths. If a suitable path is found, the user can select it, triggering the transition to the next phase of the trip initialization. If no suitable option is available, the system returns to the idle state, ready for a new search request.

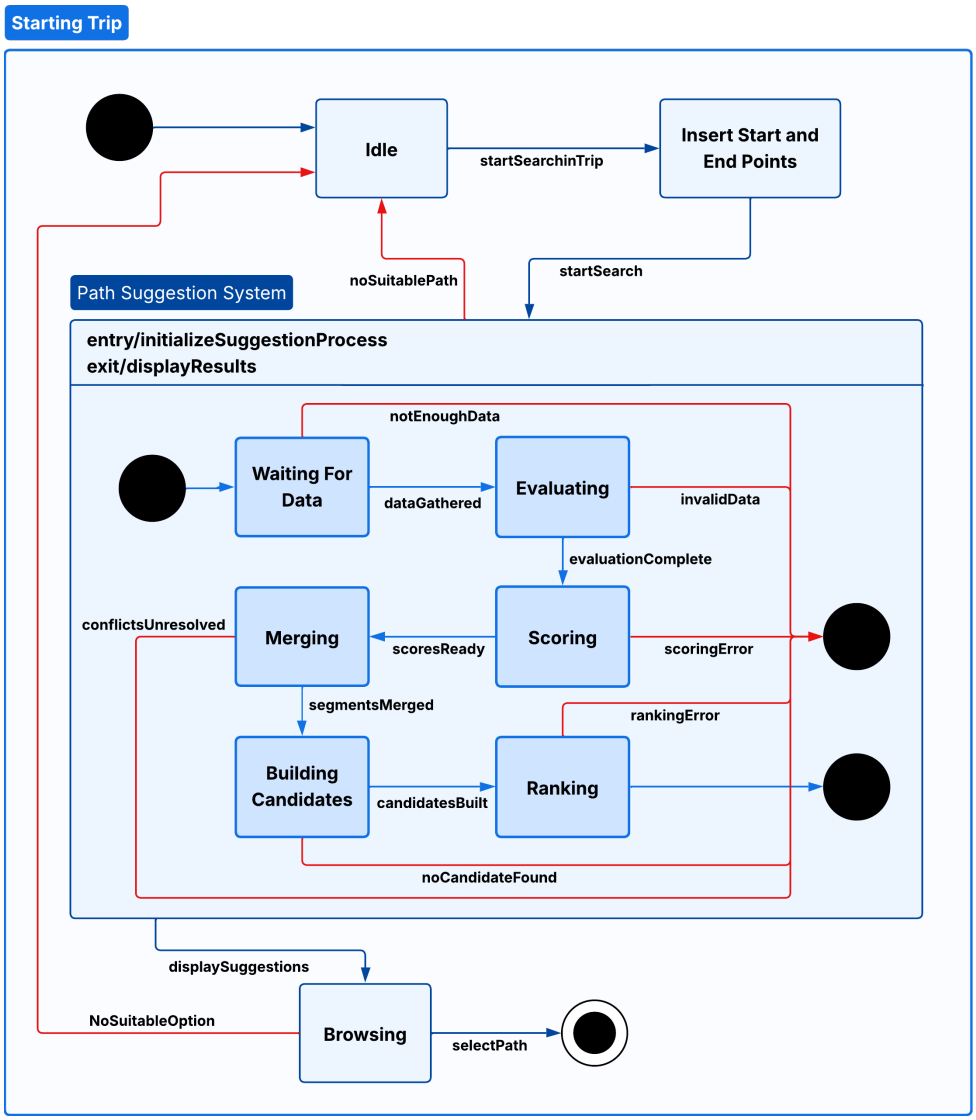


Figure 2.2: Starting Trip State Diagram

Manually creating a new path

The following diagram illustrates the states related to the manual creation of a new path by a logged-in user. A logged-in user can start the manual creation process by selecting the "Create Path" option. Once this action is triggered, the system leaves the idle state and enters the Editing Metadata state, where the user provides general information about the path, such as its name, description, visibility, and other optional details.

After the metadata are confirmed and saved, the process continues in the Adding Segments state. In this phase, the user manually draws or selects the path segments on the map to define the complete route. When all segments have been added, the user proceeds to the Validating Path state, where the system checks the consistency and continuity of

the selected route. If the validation succeeds, the process transitions to the Saving Path state, during which the system stores the new path in the database. In case of a validation failure or a save error, the system moves to an error path and returns to the idle state, allowing the user to retry or discard the creation.

If the saving process completes successfully, the system reaches the final state, confirming that the new path has been correctly created and stored. Throughout the process, the user can cancel the creation, which immediately returns the system to the idle state.

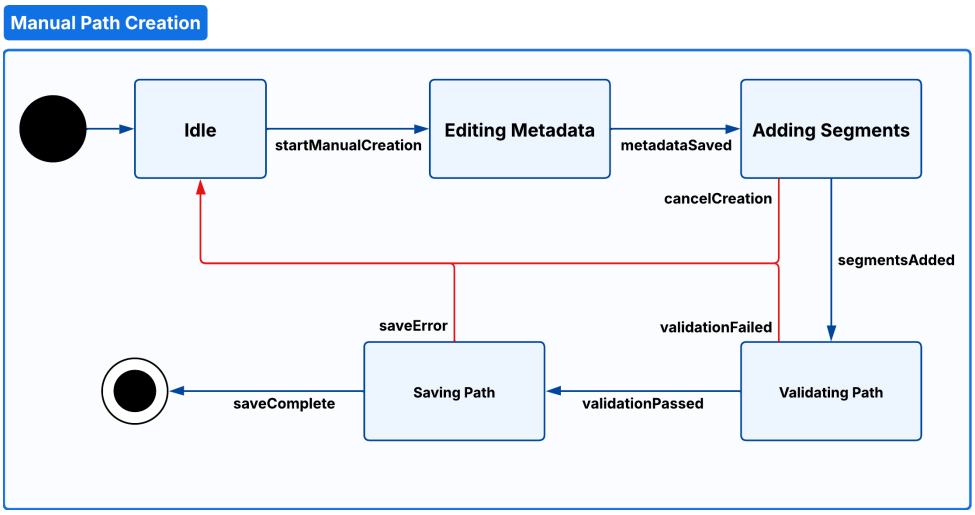


Figure 2.3: Manual Path Creation State Diagram

Automatically creating a new path

The following diagram illustrates the states involved in the automatic creation of a new path by a logged-in user using GPS tracking. After accessing the application, the user can start the automatic creation process by selecting the "Automatic Mode" option. Once this action is triggered, the system leaves the idle state and enters the Editing Metadata State, where the user can provide information about the path. After confirming the metadata, the system transitions to the Waiting GPS Lock state, where it requests location permissions and attempts to establish a stable GPS connection. If the GPS signal is successfully acquired, the system transitions to the Recording state, during which the user's movements are continuously tracked and converted into path segments.

While recording, the user can manually stop the tracking (userStop), or the process can automatically stop if a long period of inactivity is detected. If the GPS connection is lost for a prolonged time or the user denies permission, the process transitions to the failed path, ending unsuccessfully and returning to the idle state. When the user stops the recording or the system detects the end of movement, the process enters the Validating

Path state, where the recorded route is checked for continuity, consistency, and potential GPS anomalies.

If the validation succeeds, the system moves to the Saving Path state, where the path data are stored in the database. Any validation or saving errors cause a transition to a failure path, returning to the idle state. Once the path has been successfully saved, the process reaches its final state, confirming that the automatically tracked path has been correctly recorded and stored.

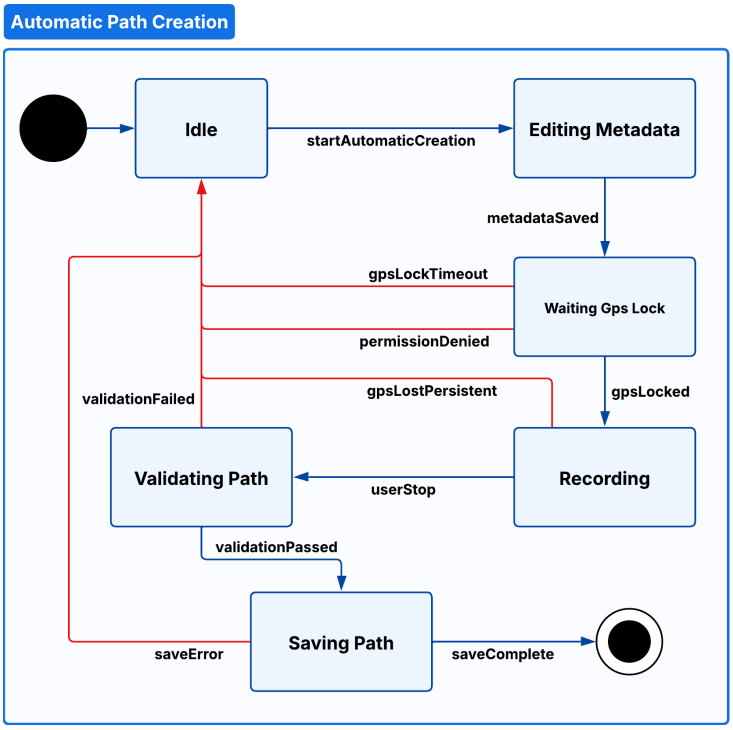


Figure 2.4: Automatic Path Creation State Diagram

Manually inserting a report about an obstacle

The following diagram illustrates the states related to the manual reporting process, which allows a logged-in user to create a report manually. Starting from the Idle state, the user initiates the manual report creation process by selecting the appropriate option from the interface. Once the process begins, the system moves to the Editing Metadata state, where the user can enter or modify relevant information, such as the description, and obstacle type.

After the metadata are saved, the process continues in the Waiting for GPS state. In this phase, the system attempts to obtain the user's current position to associate the report with a specific geographic location. If the GPS lock is successfully acquired, the process

transitions to the Saving Report state, where the report is stored in the database.

If any error occurs during the GPS acquisition or the saving phase (GPSError, saveError), or if the user decides to cancel the operation (reportCanceled), the process terminates prematurely and returns to the Idle state. When the report is successfully saved, the system reaches the final state, confirming that the manual report has been correctly created and recorded.

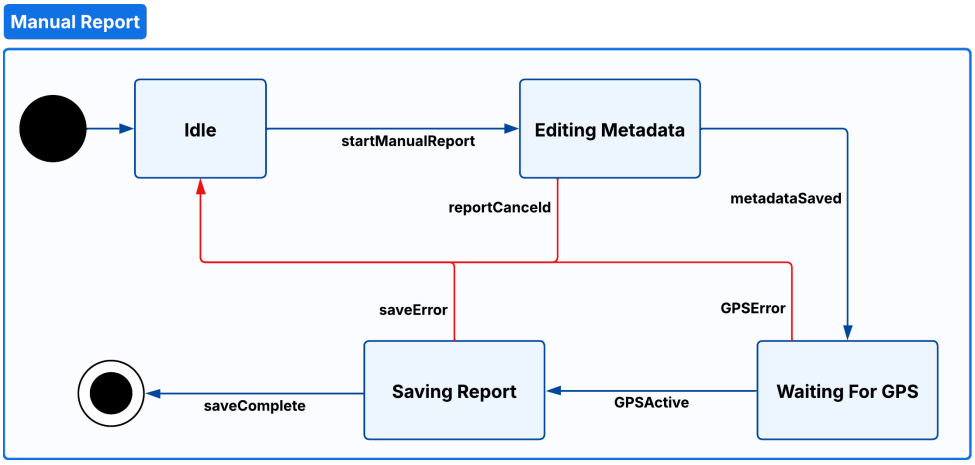


Figure 2.5: Manual Report State Diagram

Automatically detecting and reporting an obstacle

The following diagram represents the states involved in the automatic reporting process, where a logged-in user can generate a report automatically when a relevant event or sensor input is detected. The process starts from the Idle state. When the system detects a sensor signal or an anomaly (SensorDetection), it transitions to the Confirming Report state, where the user is asked to confirm whether the detected event should be reported.

If the user confirms, the process continues in the Editing Metadata state, allowing the user to optionally edit or add information related to the report, such as a short description or obstacle type. Once the metadata are saved, the system moves to the Waiting for GPS state, where it tries to obtain the current position to associate it with the report.

If the GPS signal becomes active, the system transitions to the Saving Report state, where the report data are finalized and stored in the database. If at any point the user cancels the report (reportCanceled), or the GPS fails to provide a valid location (GPSError), the process terminates prematurely and returns to the idle state.

Finally, if the saving procedure completes successfully (saveComplete), the system reaches its final state, confirming that the automatic report has been correctly generated and

saved.

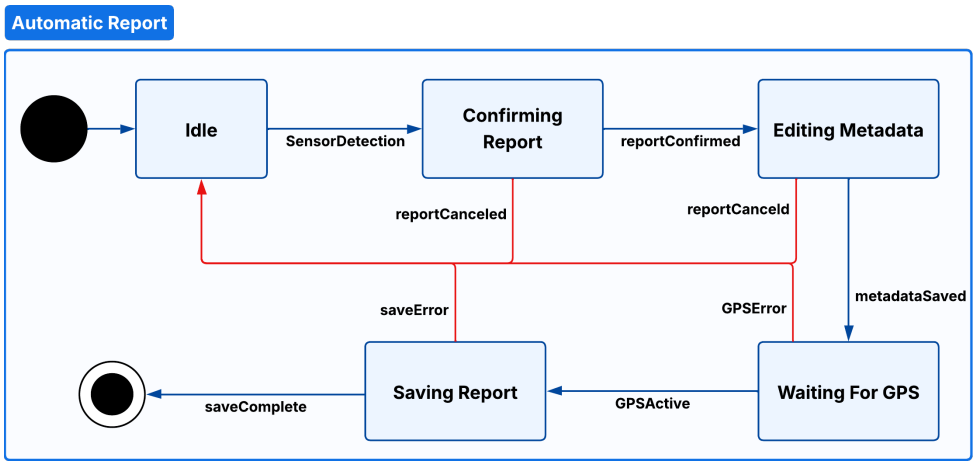


Figure 2.6: Automatic Report State Diagram

2.2. Product functions

The following points outline the main functionalities offered by the system from the user's perspective.

- **User Registration and Login**:

  The system allows new users to register and existing users to log in through a dedicated interface accessible from the main page of BBP. Registration requires only basic personal information such as name, email, and password. Once logged-in, users gain access to additional functionalities that are not available to guest users, including the ability to record and track personal cycling trips, create personalized bike paths, view personal activity history and performance statistics, and contribute information to the community by reporting or confirming bike path conditions. Guest Users can browse available bike paths, search for routes between two locations, and receive route suggestions based on community rankings, but they cannot record trips, create new paths, or contribute reports.

- **Bike Path Search and Visualization**:

  Any user can search for bike paths by specifying an origin and a destination. BBP displays on a map all available routes connecting the two selected points. Each path has information about its current condition, derived from aggregated feedback provided by logged-in users. The displayed paths are ranked according to an overall score computed by the system. This score takes into account both the physical

condition of the path (e.g., optimal, medium, requires maintenance) and its effectiveness in connecting the specified origin and destination. Users can explore the visualized paths, view their details, and select the most suitable route for their needs. However, only logged-in users can start recording a trip or contribute new information about the selected paths.

- **Bike Path Creation**:

  Logged-in users can create new custom bike paths using two different approaches. In manual mode, the user specifies the names of the street segments that compose the desired route and provides information about the characteristics of each segment. This allows cyclists to define personalized routes that may not yet be present in the system or that better match their preferences. Alternatively, in automatic mode, the system dynamically creates a new path by recording GPS data while the user follows their desired route during an actual trip. As the user bikes, BBP collects positional information to reconstruct the exact path in real time. This approach enables users to create paths simply by cycling along them, without needing to manually specify street names or segments beforehand. Regardless of the creation method, the newly created path can remain private to the user or be made publishable for the entire community.

- **Trip Recording and Data Acquisition**:

  A logged-in user starts a trip by choosing a route on the map and selecting the data acquisition mode: manual (distance, time, and average speed only, without automatic detections) or automatic. In automatic mode, BBP uses GPS, an accelerometer, and a gyroscope to track the route and identify potential irregularities in the road surface. Pressing Start begins the trip: the app displays the user's real-time location and provides navigation to the destination. The activity stops automatically upon arrival or manually with Stop, then the system processes the data and generates a summary. If recording is automatic, BBP sends pop-ups to confirm or correct any detected anomalies before the data is made available to the community.

- **Cycling Activity History and Aggregate Statistics**:

  Logged-in users can access the "My Activities' section" to review their personal cycling history. This section displays both a chronological list of all recorded trips and aggregate statistics summarizing the users overall performance over time. The aggregate statistics are presented on the same page as the trip list and include

metrics such as total number of trips, total kilometers traveled, average speed across all activities, and weekly or monthly riding frequency. Each trip entry in the list includes summary information such as the date, distance covered, duration, and primary route followed. Users can sort the list by date or distance to easily find specific activities. The trip list and aggregate statistics remain securely linked to the users account, allowing them to track their performance and monitor improvements over time.

- **Trip Specific Statistics and Environmental Data**:

  Logged-in users can select any trip from their activity list to access detailed trip specific statistics. For each individual trip, BBP provides total distance, duration, average and maximum speed. The trip page also displays a map showing the complete route followed by the user. If environmental data were available during the ride either from the users external device or from an external meteorological service they are presented alongside the trip statistics. These environmental data may include average temperature, humidity levels, and wind speed and direction. Additionally, either if the user recorded the trip in automatic mode or manual mode, the map highlights points where the accelerometer detected abnormal movements, indicating possible obstacles or rough surfaces encountered along the route or points where the user manually reported issues.

- **Manual Reporting of Bike Path Conditions**:

  BBP enables logged-in users to manually report the condition of any bike path they encounter. This feature helps keep route information accurate, especially when users spot issues like obstacles or closures. By selecting Report Path Condition in the app, users can mark the affected segment on a map, describe the issue, and assign a status such as “optimal,” “medium,” “requires maintenance,” or “closed.”

- **Real Time Confirmation and Rejection of Reported Path Conditions**:

  During a ride, logged-in users may receive pop-up alerts about nearby issues reported by others, such as “Possible pothole ahead” or “Path requires maintenance.” As they approach the location, users can confirm that the issue persists or reject it if the path is clear. Their feedback is instantly recorded by BBP and influences the reliability of reports-confirmed ones gain credibility, while frequently rejected ones are downgraded or removed.

- **Trip Confirmation of Automatically Detected Anomalies**:

  When a logged-in user records a trip in automatic mode, BBP uses the device’s

sensors to detect irregularities such as potholes or obstacles. To prevent false positives, detected anomalies are not immediately published. Instead, the user is shown a confirmation screen to review and validate or correct each detection. Only confirmed data becomes publishable, ensuring accuracy and reliability of shared path information.

- **Aggregation and Merging of Bike Path Condition Reports**:

  BBP combines multiple user reports about the same bike path into a single, updated condition overview. The system prioritizes recent and consistent reports, giving more weight to those confirmed by several users. In case of conflicting information, BBP determines the path's status based on the most recent and widely supported data, ensuring an up-to-date representation of path conditions.

2.3. User Characteristics

The Best Bike Paths (BBP) platform is designed to support two main user categories: **guest users** and **logged-in users**. Each group has distinct goals, permissions, and levels of interaction with the system.

- **Guest Users**:

  Guest users interact with the system without performing any authentication. They may either be users who have not created an account, or users who have an existing account but are currently logged out. Guest users can explore existing bike paths, view global information such as distance, rankings, and general statistics, and consult public reports about obstacles or road conditions. Their interaction with the system is limited to browsing and visualization; they are not allowed to create, edit, or report any paths or obstacles.

- **Logged-in users**:

  Logged-in users have full access to the system's functionalities after creating an account and logging in. They can create new bike paths using GPS tracking, or by manually selecting points on a map interface. They can also manually or automatically submit obstacle reports, and validate or confirm existing ones shared by other users. If the automatic mode is used, the user can confirm the detected obstacles before submission. Furthermore, logged-in users can record trips, which are then stored within their personal profiles, allowing them to view historical data and general and per-trip statistics such as distance covered, average speed, and total time

spent cycling.

2.4. Assumptions, dependencies and constraints

2.4.1. Dependencies

The BBP application depends on several external systems and services to provide its core functionalities:

- The system requires a connection with the remote database through the backend APIs to retrieve, store, and update user and path data.
- The application depends on a reliable Internet connection to access the backend, synchronize trip data, and retrieve updated path information.
- A geographic map service is required to display maps, compute routes, and process geocoding and reverse-geocoding requests.
- External meteorological APIs are used to collect environmental data (e.g., temperature, humidity, wind speed) associated with trips.

2.4.2. Constraints

The design and implementation of the BBP system are subject to the following constraints:

- The system must fully comply with the principles and requirements established by the GDPR (General Data Protection Regulation, EU 2016/679), ensuring that all collected and processed data respect user privacy, security, and transparency standards.
- The application must conform to accessibility and usability guidelines defined by major mobile platforms (Android and iOS).
- Any third-party APIs used for maps, weather, or notifications must be employed in compliance with their respective terms of service and usage quotas.

2.4.3. Domain Assumptions

This subsection lists the assumptions about the real world that are taken as true for the requirements to hold.

[D1] The system has access to a geographic map service and geocoding capabilities to

process origin/destination and render paths.

[D2] External meteorological services are reachable and operational with acceptable latency during normal operation.

[D3] Users' mobile devices are equipped with GPS and have granted the BBP app access to location and motion sensors.

[D4] Motion sensors (accelerometer, gyroscope) and GPS provide data of sufficient accuracy and reliability for obstacle detection.

[D5] Manual input provided by logged-in users is assumed to be truthful, accurate, and sufficiently complete for a path/report entry.

[D6] A bike path is either a dedicated bicycle track or a low-traffic road where motor-vehicle speed limits are compatible with average cycling speed.

[D7] Notifications or pop-ups sent by the BBP system reach users within 5 minutes.

[D8] Reported conditions may become obsolete over time; data "freshness" decays and is considered by the aggregation/ranking logic.

3 | Specific Requirements

3.1. External Interface Requirements

The external interface requirements describe the main interfaces through which users, hardware, and external services communicate with the system.

3.1.1. User Interfaces

BBP will be implemented as a mobile application available for Android and iOS devices. The interface will be designed to be intuitive and minimal, allowing users to create, browse, and report bike paths directly from their smartphones. Main sections will include an interactive map, a reporting form, a trip history, and user statistics. Each user type (guest, logged-in user) will access specific features according to their role. Detailed mockups and design choices will be presented in the Design Document (DD).

3.1.2. Hardware Interfaces

The application will run on common smartphones equipped with GPS, internet connection, and basic sensors. The GPS module will be used to track routes and provide automatic path creation, while optional sensors (accelerometer or gyroscope) may improve data accuracy. A stable network connection will be required for map loading and data synchronization.

3.1.3. Software Interfaces

The BBP application will interact with a remote backend through RESTful APIs. These APIs will handle user authentication, map data, path storage, and report submission. External services may also be used to obtain weather data relevant to the user's current route.

3.1.4. Communication Interfaces

All communications between the mobile app and the server will occur via HTTPS to guarantee secure data exchange. The system will use JSON-formatted requests and responses.

3.2. Functional Requirements

3.2.1. Requirements

This subsection specifies all the functional requirements that the BBP system must fulfill to achieve the goals described in the previous chapters. The requirements are organized according to the main functionalities of the system and are expressed in a testable form, describing what the system shall do rather than how it shall be implemented.

1. **User Registration and Authentication**

   [R1] The system shall allow guest users to create an account by providing personal information and credentials.

   [R2] The system shall allow registered users to log into the application using valid credentials.

   [R3] The system shall allow logged-in users to view their profile and account settings.

   [R4] The system shall allow logged-in users to update their profile and account settings.

   [R5] The system shall allow logged-in users to log out of the application, ending their current session.

2. **Trip Recording and Management**

   [R6] The system shall allow logged-in users to start a cycling trip in manual or automatic mode.

   [R7] The system shall allow logged-in users to stop a currently active trip and save the recorded data.

   [R8] The system shall collect GPS data during trip recording.

   [R9] The system shall collect motion sensor data (accelerometer, gyroscope) during trip recording when available.

   [R10] The system shall allow logged-in users to view the list of their recorded trips.

[R11] The system shall allow logged-in users to view a summary of their overall cycling statistics (total distance, total time, average speed, etc.).

[R12] The system shall allow logged-in users to view statistics for each trip (distance, speed, duration, etc.).

[R13] The system shall display the route and reported obstacles associated with a recorded trip.

[R14] The system shall allow logged-in users to delete a recorded trip.

3. **Trip Data Enrichment**

[R15] The system shall communicate with external weather services to retrieve meteorological data related to the time and location of a trip.

4. **Automatic Data Processing and Detection**

[R16] The system shall detect when a user is cycling based on speed and acceleration patterns.

[R17] The system shall detect irregular movements from sensor data that may suggest potholes or surface defects.

[R18] The system shall present automatically detected path and obstacle data to the user for manual confirmation before publishing.

5. **Path Information Management**

[R19] The system shall allow logged-in users to manually create a new bike path by drawing or specifying street segments.

[R20] The system shall allow logged-in users to manually report obstacles or problems on a bike path.

[R21] The system shall allow logged-in users to manually confirm or reject the presence of obstacles reported by other users.

[R22] The system shall allow logged-in users to create a new bike path in automatic mode using GPS tracking.

[R23] The system shall allow logged-in users to delete or edit their previously created paths.

[R24] The system shall allow logged-in users to set the visibility of their created paths as public or private.

6. **Path Data Aggregation and Status Evaluation**

[R25] The system shall aggregate multiple user reports referring to the same path segment.

[R26] The system shall evaluate the reliability of each path segment based on the number of confirmations and report freshness.

[R27] The system shall determine the current status of a path (optimal, medium, sufficient, requires maintenance, closed).

[R28] The system shall allow any user (guest or logged-in) to view the detailed status and latest reports of a selected bike path.

7. **Route Discovery and Search**

[R29] The system shall allow any user (guest or logged-in) to browse available public bike paths on a map.

[R30] The system shall allow any user to search for bike paths connecting two locations.

[R31] The system shall compute suggested routes based on path quality and distance.

[R32] The system shall rank suggested routes according to their safety and quality.

8. **Navigation**

[R33] The system shall display the user's current GPS position during navigation along a selected path.

9. **Notifications and Alerts**

[R34] The system shall send in-app notifications or pop-ups to warn users about nearby obstacles or closed path segments during navigation.

10. **External Services Integration**

[R35] The system shall interface with map and geocoding services to translate addresses into coordinates and render paths.

[R36] The system shall ensure that communication with all external services (map, weather) handles temporary unavailability gracefully.

3.2.2. Use Case Diagrams

This subsection illustrates the main interactions between actors and the BBP system. Each diagram focuses on a specific actor category, distinguishing what a guest user can do from the actions reserved to logged-in users.

Authentication and Profile Management Use Cases

This diagram shows how users access and manage their personal area within the system. Guest users can register to create an account and then log in to gain full access to BBP functionalities. Both the Register and Log-in cases include the Validate Inputs use case, ensuring that all user-provided data meet format and consistency rules. Once authenticated, users can log out, edit their personal information, and control the visibility of their profile data.

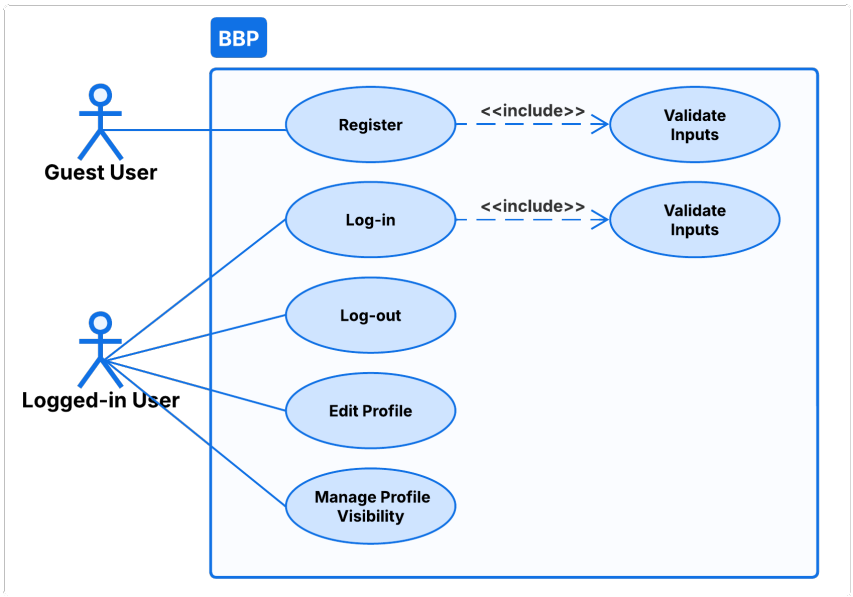


Figure 3.1: Authentication and Profile Management Use Case Diagram

Guest User Use Cases

Guest users can freely access public information related to the available bike paths without being authenticated. They can search for paths, browse their details, and check the overall condition of each segment. When a user selects a path, the system retrieves and displays the corresponding aggregated condition. Guests can also start and stop a trip, but the data is not stored or associated with a user account.

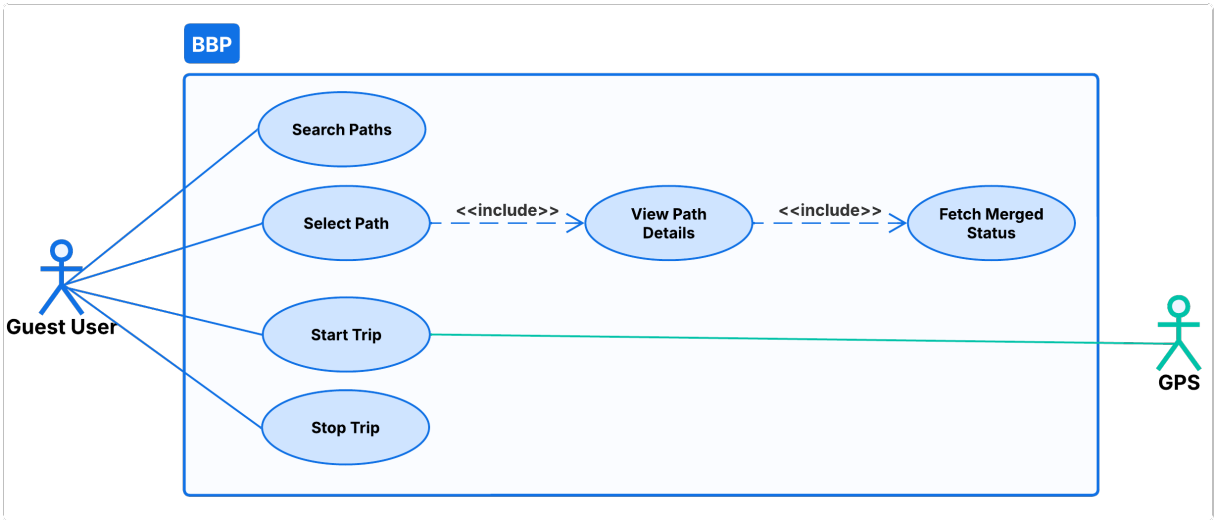


Figure 3.2: Guest User Use Case Diagram

Logged-in User Use Cases

Logged-in users have access to all interactive and community features of the BBP platform. They can create new paths manually or automatically, by recording their route or by manually adding segments. Each new path can be saved only after passing the Validate Inputs step and may optionally be made public. Users can also delete paths they have created. During trip recording, data from sensors are analyzed to automatically detect anomalies; these detections trigger the Confirm/Reject Detection extension, allowing the user to verify their validity. Users can also report path conditions manually and review other users' reports. Once a trip is stopped, a summary enriched with weather data is generated. Finally, users can view their activity history and per-trip or overall statistics.

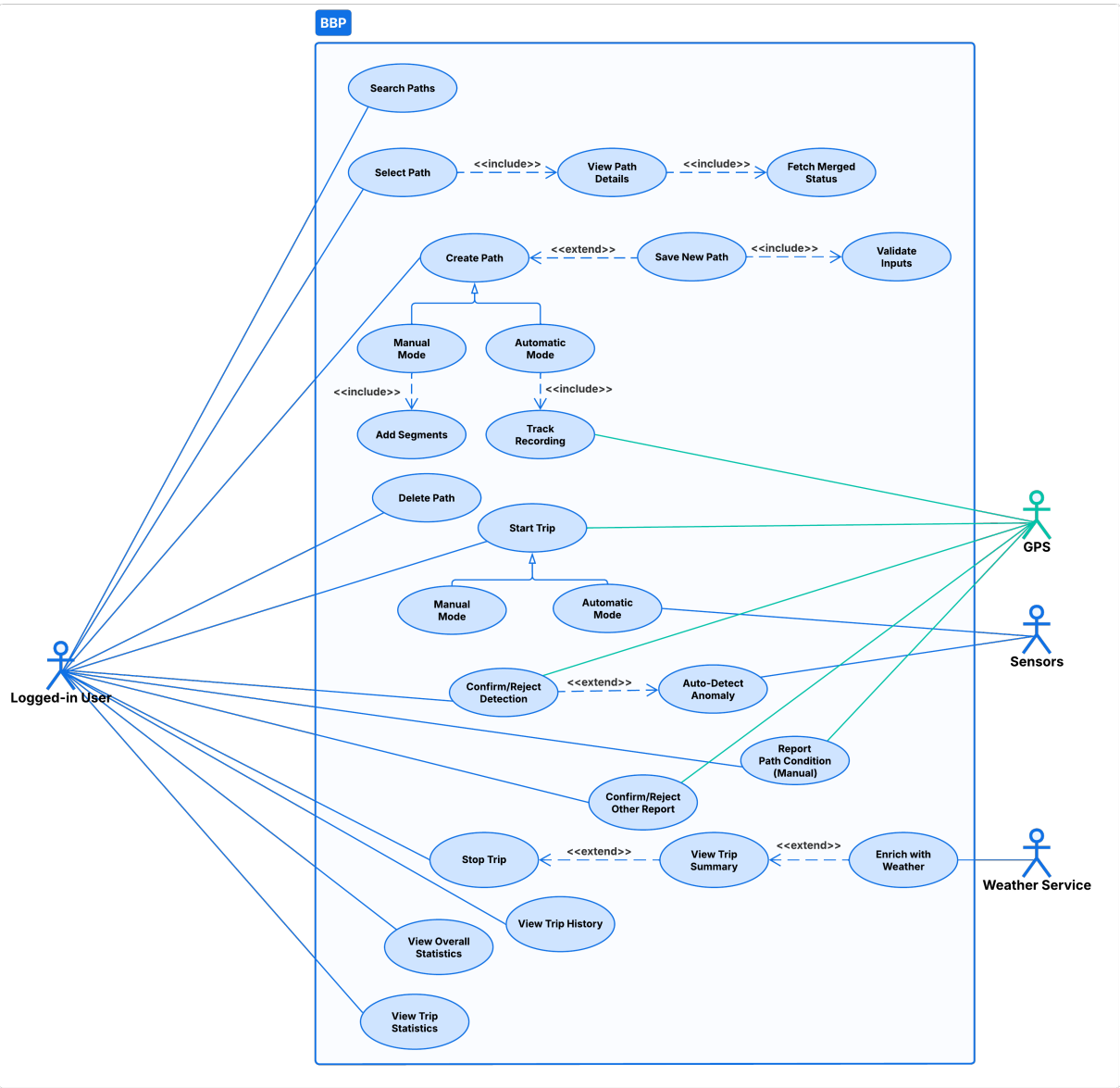


Figure 3.3: Logged-in User Use Case Diagram

3.2.3. Use Cases

The main system processes that define the core functionalities of BBP are described below. Each process is presented through a structured table that details the actors involved, the flow of events, and possible exceptions. A corresponding sequence diagram is provided for each process to illustrate the interaction between the actor and the system components during its execution.

[UC1] - User Registration

Actor(s)	Guest User
Entry Condition	The actor does not have an account, wants to register and is on the app's welcome page.
Event Flow	1. The actor clicks on the "Sign Up" button. 2. The system displays the registration form. 3. The actor fills in the required personal information (e.g., name, email, password). 4. The system validates the provided data. 5. If validation succeeds, the system creates a new account and redirects the actor to the welcome page.
Exit Condition	A new user account is created.
Exceptions	If the data are invalid or incomplete, the system shows an error and asks for correction. If the email is already logged-in, the system shows an error and invites the user to log in. If a temporary connection or BBP error occurs, the system displays an appropriate error message.
Notes	The entered password is stored in a hashed form for security purposes.

Table 3.1: User Registration Process Detail

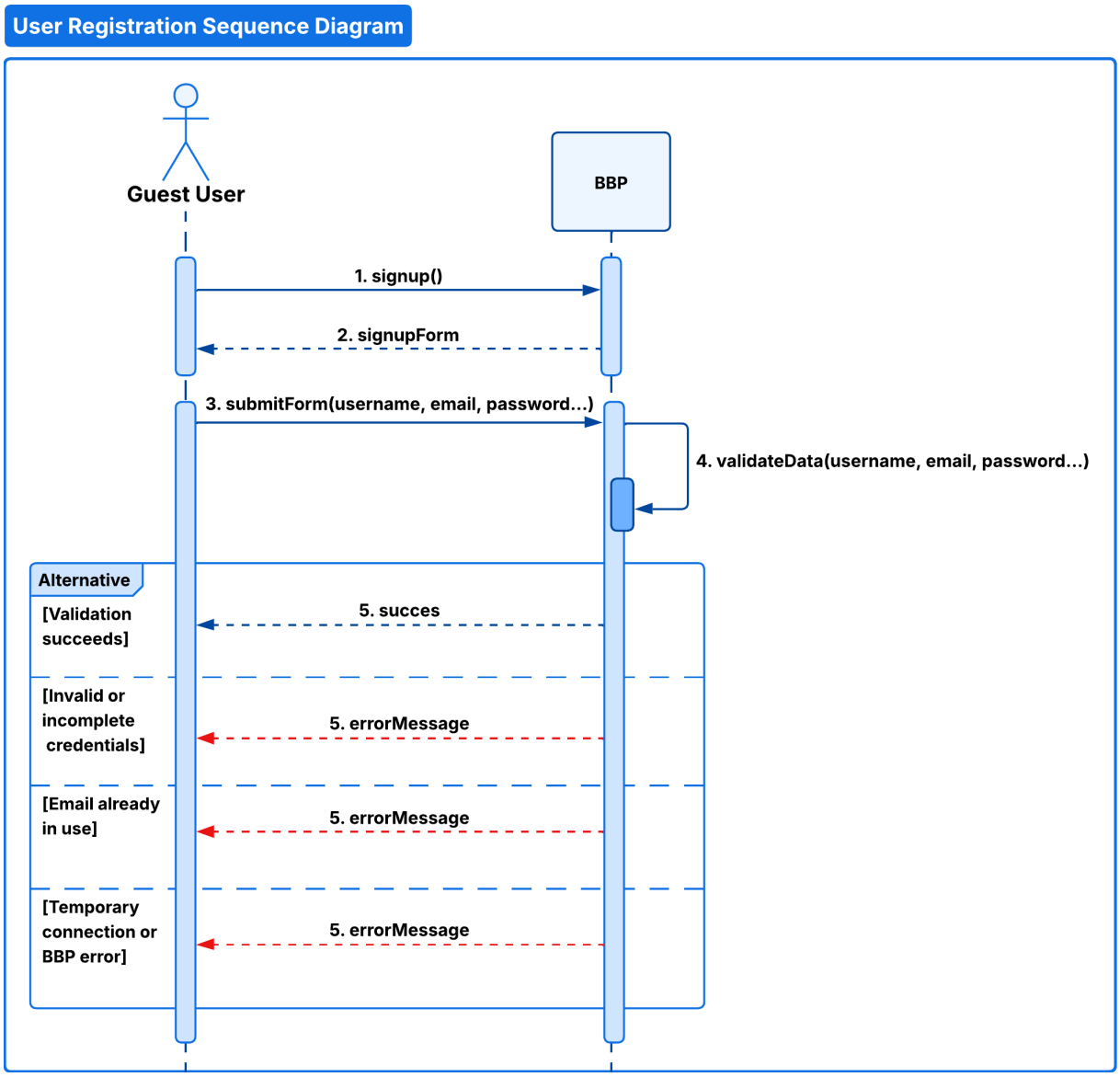


Figure 3.4: User Registration Sequence Diagram

[UC2] - User Log In

Actor(s)	Guest User
Entry Condition	The actor has an account, wants to access it and is on the application's welcome page.
Event Flow	1. The actor clicks the "Log In" button. 2. The system displays the login form. 3. The actor enters valid credentials (email and password). 4. The system validates the credentials against the stored data. 5. If validation succeeds, the system authenticates the user and redirects them to the home page.
Exit Condition	The actor is successfully authenticated.
Exceptions	If the credentials are invalid or incomplete, the system displays an error message and prevents access until corrected. If the credentials are unrecognized, the system displays an error message and allows another login attempt. If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	After login, a valid session is created; the session expires automatically after a period of inactivity.

Table 3.2: User Log In Process Detail

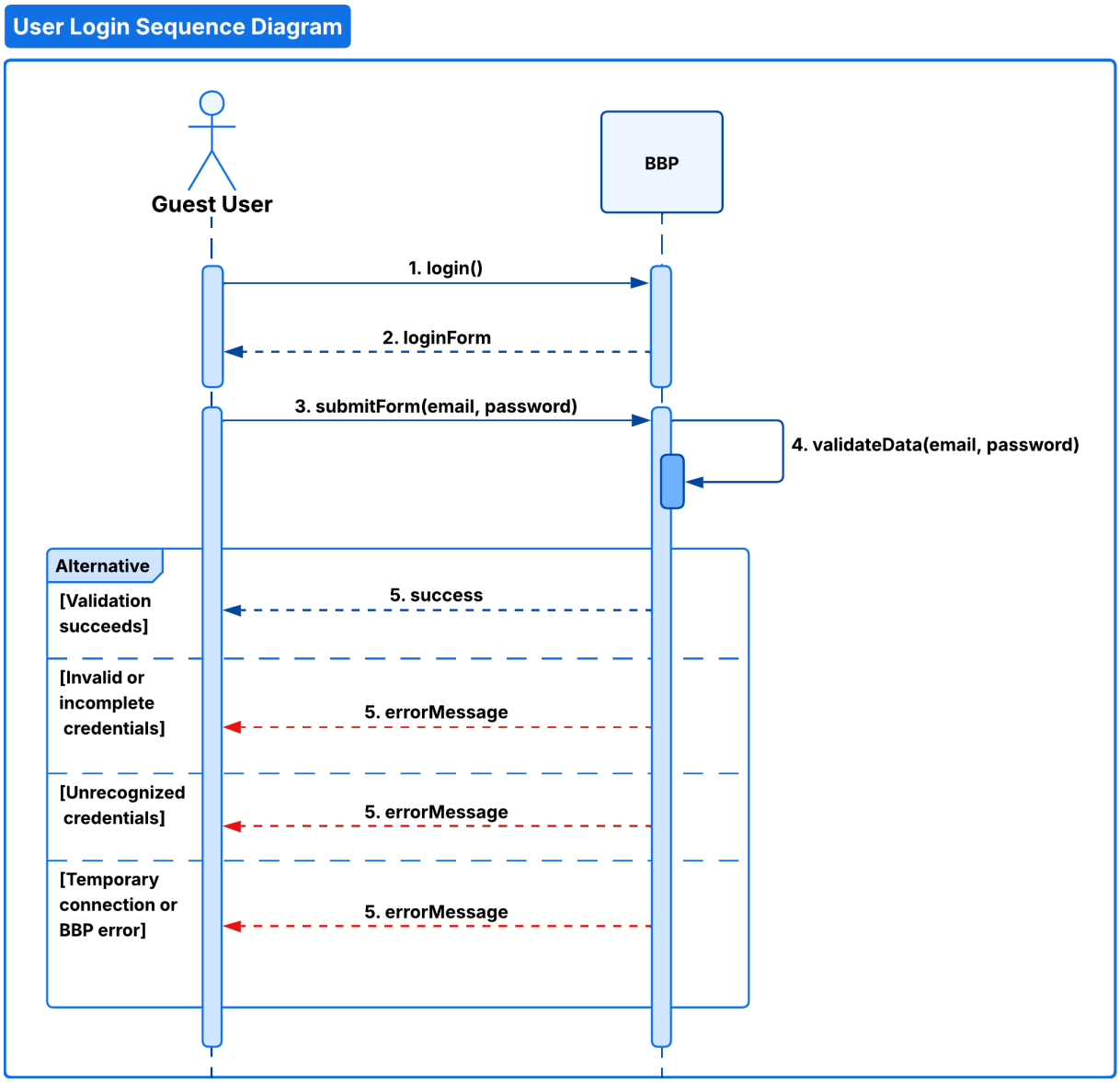


Figure 3.5: User Log In Sequence Diagram

[UC3] - User Log Out

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and is using the application.
Event Flow	1. The actor clicks on the "Log out" option. 2. The system disconnects the actor from the current session. 3. The system clears user-specific local data, and redirects the actor to the welcome page.
Exit Condition	The actor is successfully logged out.
Exceptions	If BBP cannot invalidate the session or the network is unavailable, the system displays an error message and prompts the actor to retry later.
Notes	Session is terminated; next access requires re-authentication.

Table 3.3: User Log Out Process Detail

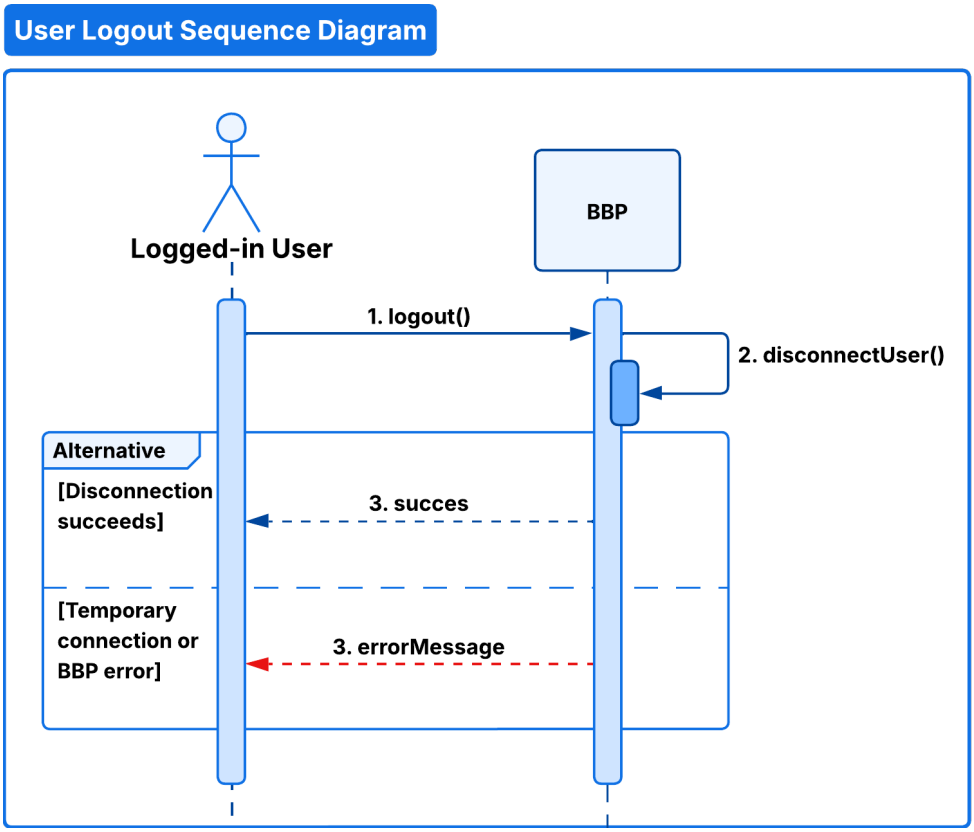


Figure 3.6: User Log Out Sequence Diagram

[UC4] - Search for a Path

Actor(s)	Guest User, Logged-in User
Entry Condition	The actor is on the home page and wants to find a bike path.
Event Flow	1. The actor selects the "Search" option. 2. The system displays the search panel. 3. The actor enters start and destination points. 4. The system validates the provided inputs. 5. The system displays the list of suggested routes to the actor.
Exit Condition	The suggested routes are displayed to the actor.
Exceptions	If the inputs are invalid or incomplete, the system displays an error message and prevents search until corrected. If no route is found, the system displays an error message and suggests adjusting inputs/filters. If there is a temporary connection or BBP error, the system displays an error message and allows retry.
Notes	The search results are ranked based on path quality and distance.

Table 3.4: Search for a Path Process Detail

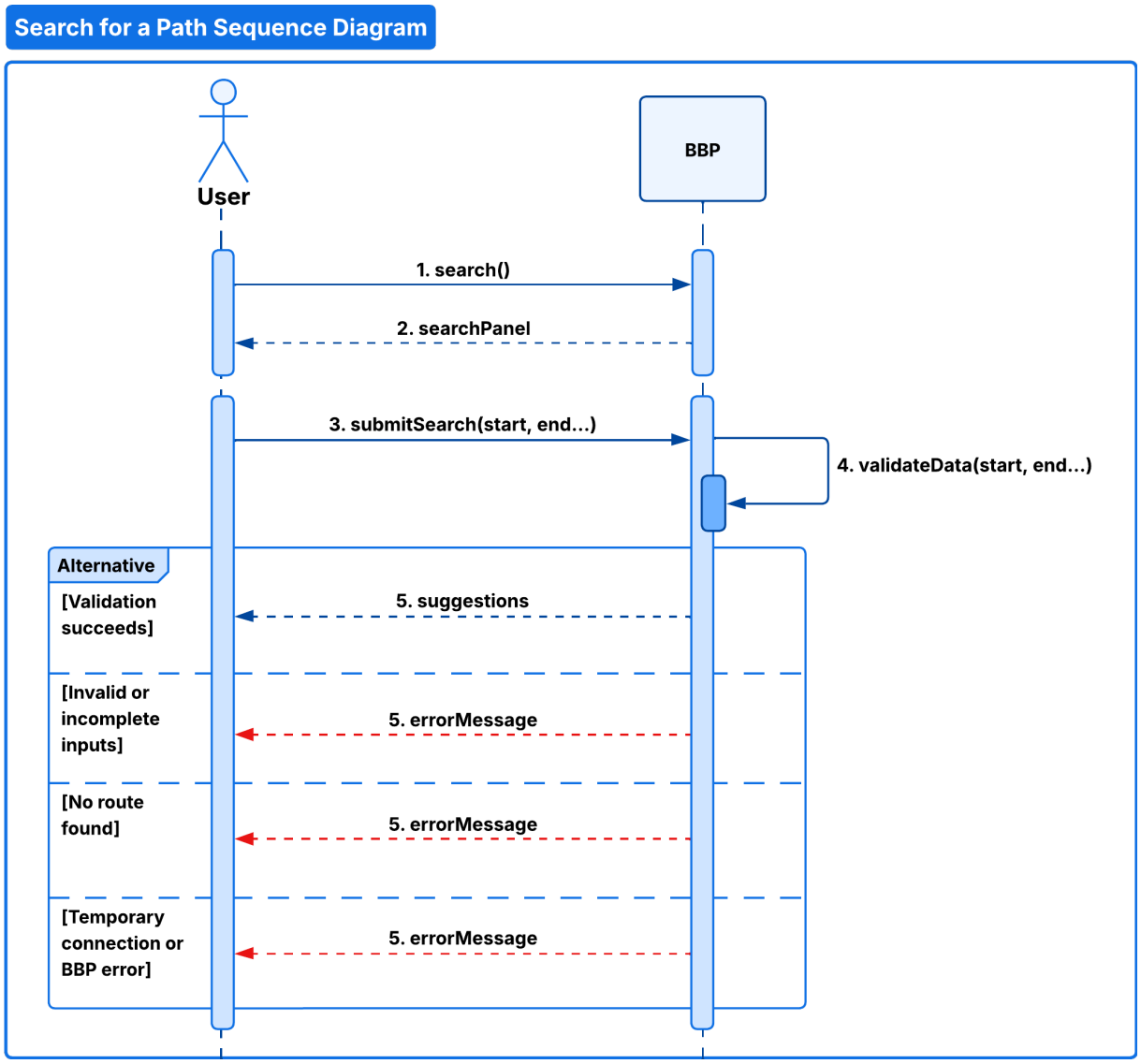


Figure 3.7: Search for a Path Sequence Diagram

[UC5] - Select a Path

Actor(s)	Guest User, Logged-in User
Entry Condition	The actor has just performed a search or is browsing the catalog and is viewing the list of suggested paths.
Event Flow	1. The actor selects one path from the results list. 2. The system displays the path details (overview, distance, ranking, reports)
Exit Condition	The path is selected. If the starting point is the current user's position, the system is ready to start the trip.
Exceptions	If the selected path is unavailable (removed/updated), the system displays an error message and returns to the results list. If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	Only Logged-in Users can store trip data; Guest Users can follow the path without recording.

Table 3.5: Select a Path Process Detail

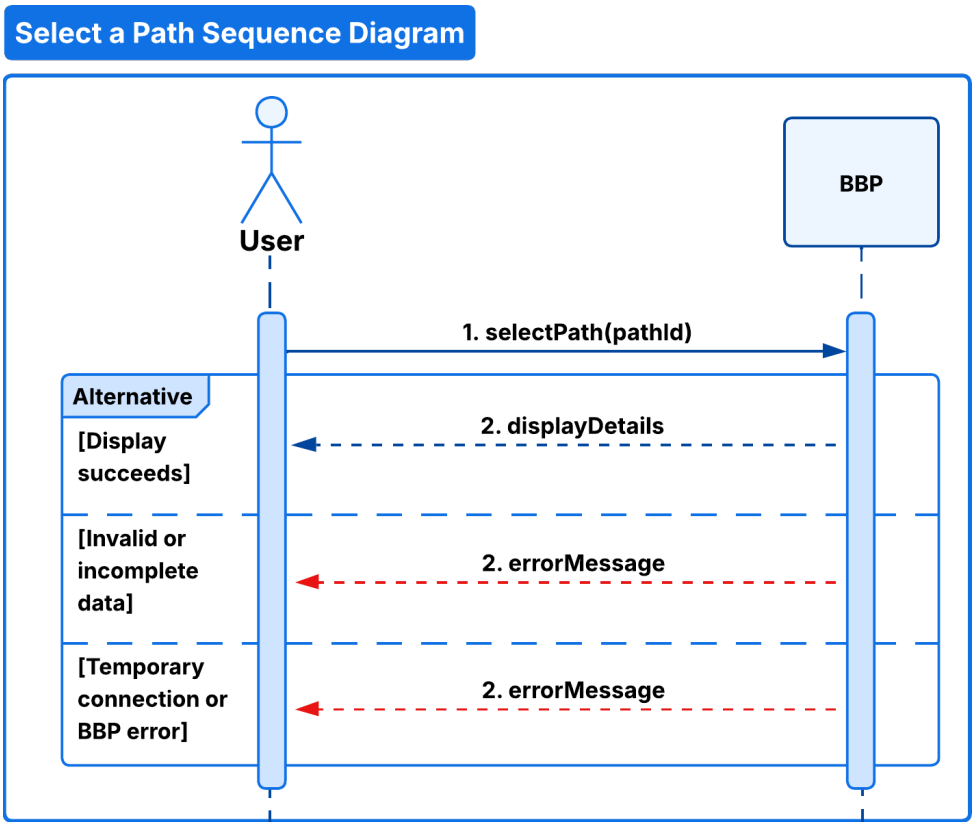


Figure 3.8: Select a Path Sequence Diagram

[UC6] - Create a Path in Manual Mode

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and wants to create a new path manually.
Event Flow	1. The actor selects 'Create Path'. 2. The system displays the available creation modes. 3. The actor chooses 'Manual'. 4. The system displays the input page for manual creation (metadata + map editor). 5. The actor fills in metadata (e.g., name, description, visibility) and draws/adds segments on the map. 6. The system validates the provided metadata and the geometry of the segments. 7. If validation succeeds, the system saves the path that will appear in the actor's path list.
Exit Condition	The new path is successfully created and associated with the actor's account.
Exceptions	If the data are invalid or incomplete, the system displays an error message and prevents saving until corrected. If there is a temporary connection or system error, the system displays an error message and allows retry.
Notes	The path visibility follows the selected setting.

Table 3.6: Create a Path in Manual Mode Process Detail

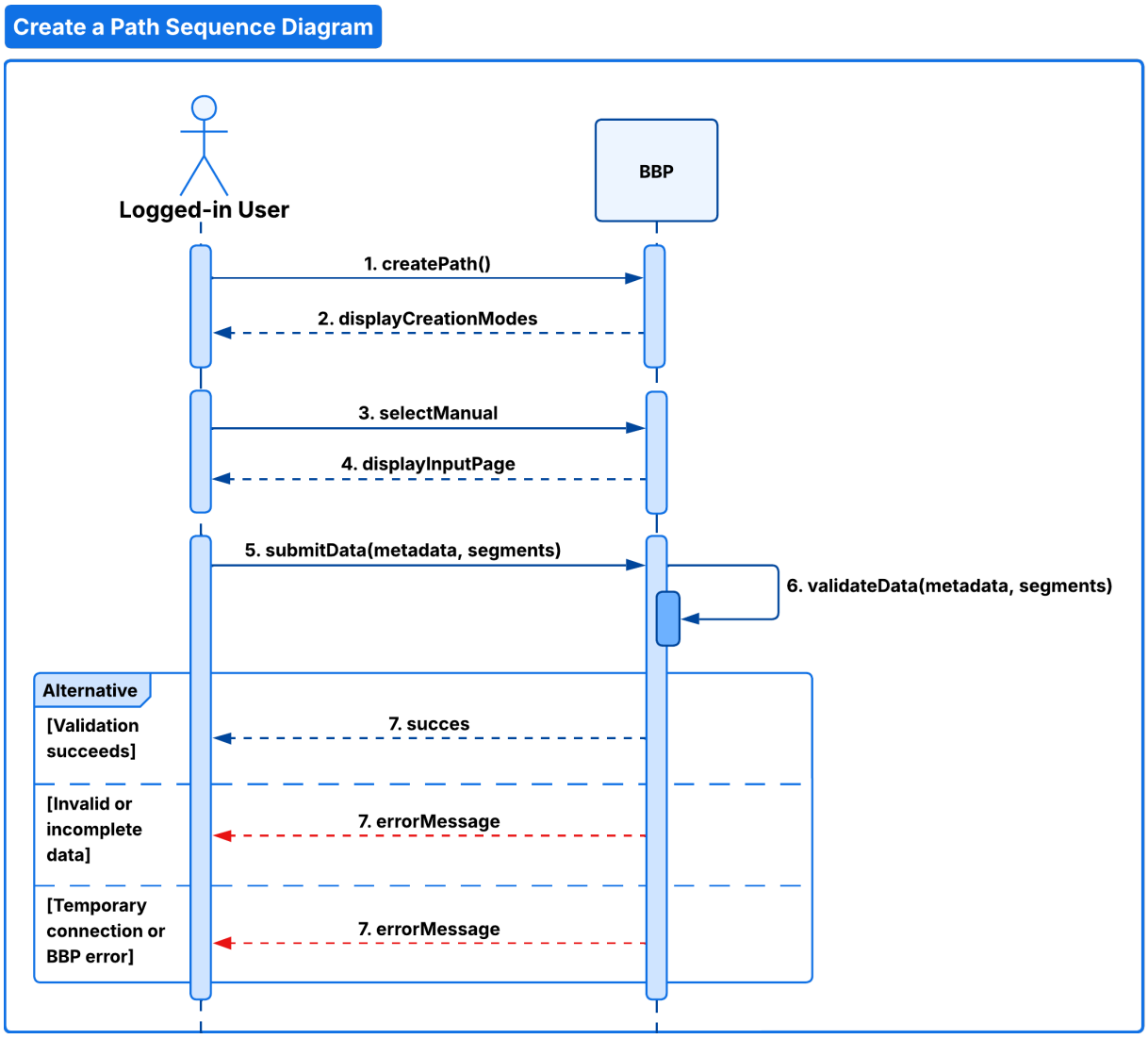


Figure 3.9: Create a Path in Manual Mode Sequence Diagram

[UC7] - Create a Path in Automatic Mode

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and wants to create a new path using the Automatic Mode.
Event Flow	1. The actor selects 'Create Path'. 2. The system displays the available creation modes. 3. The actor chooses 'Automatic'. 4. The system displays the input page for metadata configuration. 5. The actor fills in basic metadata (e.g., name, visibility, description). 6. The system validates the provided metadata. 7. If the validation succeeds, the system confirms readiness and starts GPS tracking. 8. The system continuously collects geo-coordinates from the GPS while the actor is moving. 9. Once tracking stops, the system validates the recorded GPS data. 10. If validation succeeds, the system saves the new path.
Exit Condition	The automatically recorded path is successfully validated and stored in the database, associated with the actor's account.
Exceptions	If the data are invalid or incomplete, the system displays an error message and prevents saving until corrected. If GPS signal is lost or unstable, the system displays an error message and prompts the actor to retry. If temporary connection or system error occurs, the system displays an error message and allows retry later.
Notes	The system continuously records and buffers GPS data locally during tracking. Only after successful validation is the path permanently stored.

Table 3.7: Create a Path in Automatic Mode Process Detail

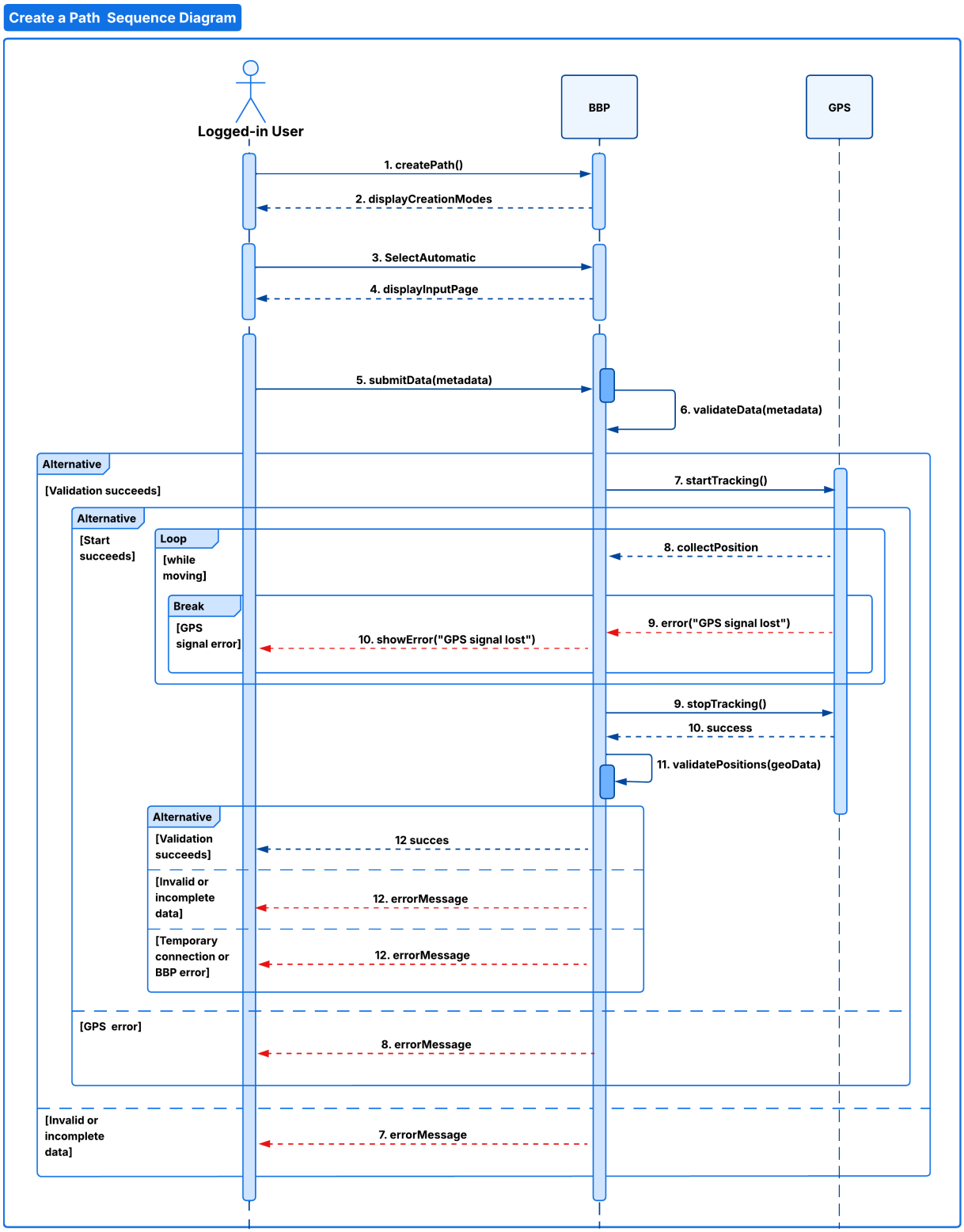


Figure 3.10: Create a Path in Automatic Mode Sequence Diagram

[UC8] - Delete a Path

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and wants to remove one of their previously created paths.
Event Flow	1. The actor accesses the "My Paths" section. 2. The system retrieves and displays the list of paths owned by the actor. 3. The actor selects a path and requests its deletion. 4. The system verifies ownership and deletes the corresponding path.
Exit Condition	The selected path is permanently deleted and no longer visible in the actor's list.
Exceptions	If the actor tries to delete a path they do not own, the system displays an ownership error message. If the selected path does not exist or was already deleted, the system notifies the actor that the path was not found. If a temporary connection or BBP error occurs, the system displays an error message and allows retry later.
Notes	Deletion is irreversible. Only the creator of the path can perform this operation.

Table 3.8: Delete a Path Process Detail

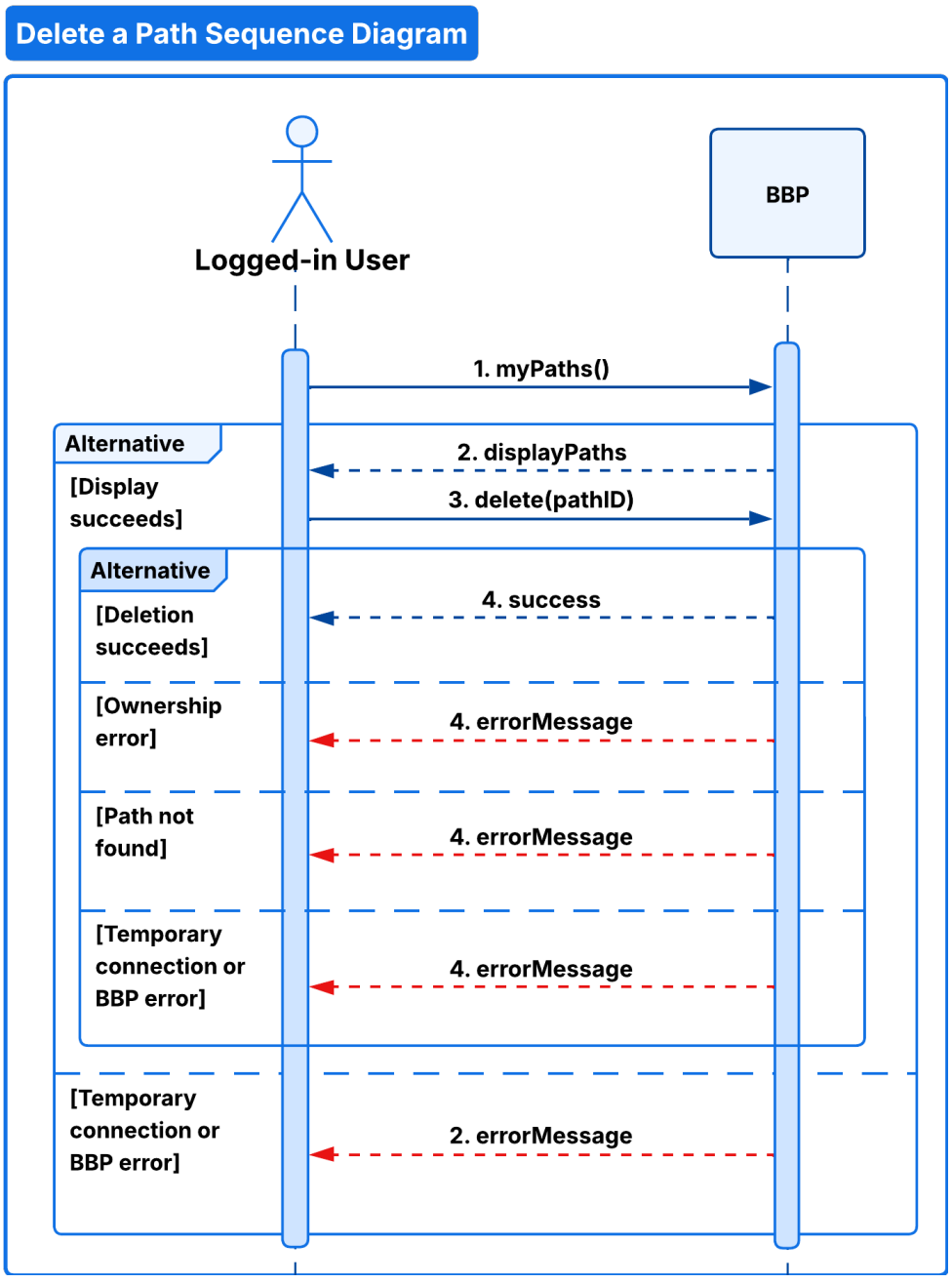


Figure 3.11: Delete a Path Sequence Diagram

[UC9] - Start a Trip as Guest User

Actor(s)	Guest User
Entry Condition	The actor has selected a path and wants to follow it without authentication.
Event Flow	1. The actor chooses to start the trip on the selected path. 2. The system activates GPS tracking to obtain the actor's real-time location. 3. The system continuously updates the actor's position on the map during the trip.
Exit Condition	The trip visualization ends when the actor stops the trip or leaves the page; no trip data are stored.
Exceptions	If the GPS signal is unavailable or the connection to BBP fails, the system displays an error message and allows retry.
Notes	Guest Users can only visualize their current location.

Table 3.9: Start a Trip as Guest User Process Detail

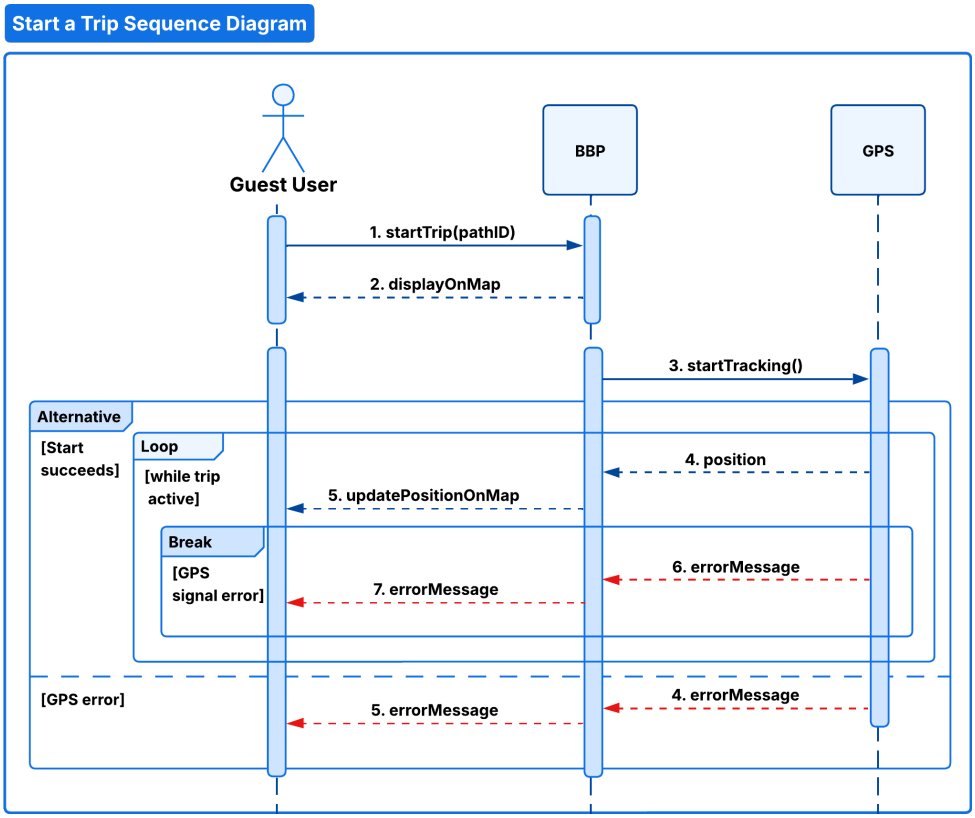


Figure 3.12: Start a Trip as Guest User Sequence Diagram

[UC10] - Start a Trip in Manual Mode as a Logged-in User

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated, has selected a path to follow, and wants to start the trip with manual mode.
Event Flow	1. The actor chooses to start the trip on the selected path. 2. The system displays the available start modes. 3. The actor selects Manual mode. 4. The system activates GPS tracking to obtain the actor's real-time location. 5. The system continuously updates the actor's position on the map during the trip.
Exit Condition	The trip is successfully started; live tracking and map updates are active.
Exceptions	If GPS permissions are denied or the GPS signal is unavailable/unstable, the system shows an error message. If a temporary connection or BBP error occurs, the system shows an error message and allows the actor to retry.
Notes	While the trip is active, the system may record trip data (e.g., time, distance, speed) for the actor's statistics.

Table 3.10: Start a Trip in Manual Mode as a Logged-in User Process Detail

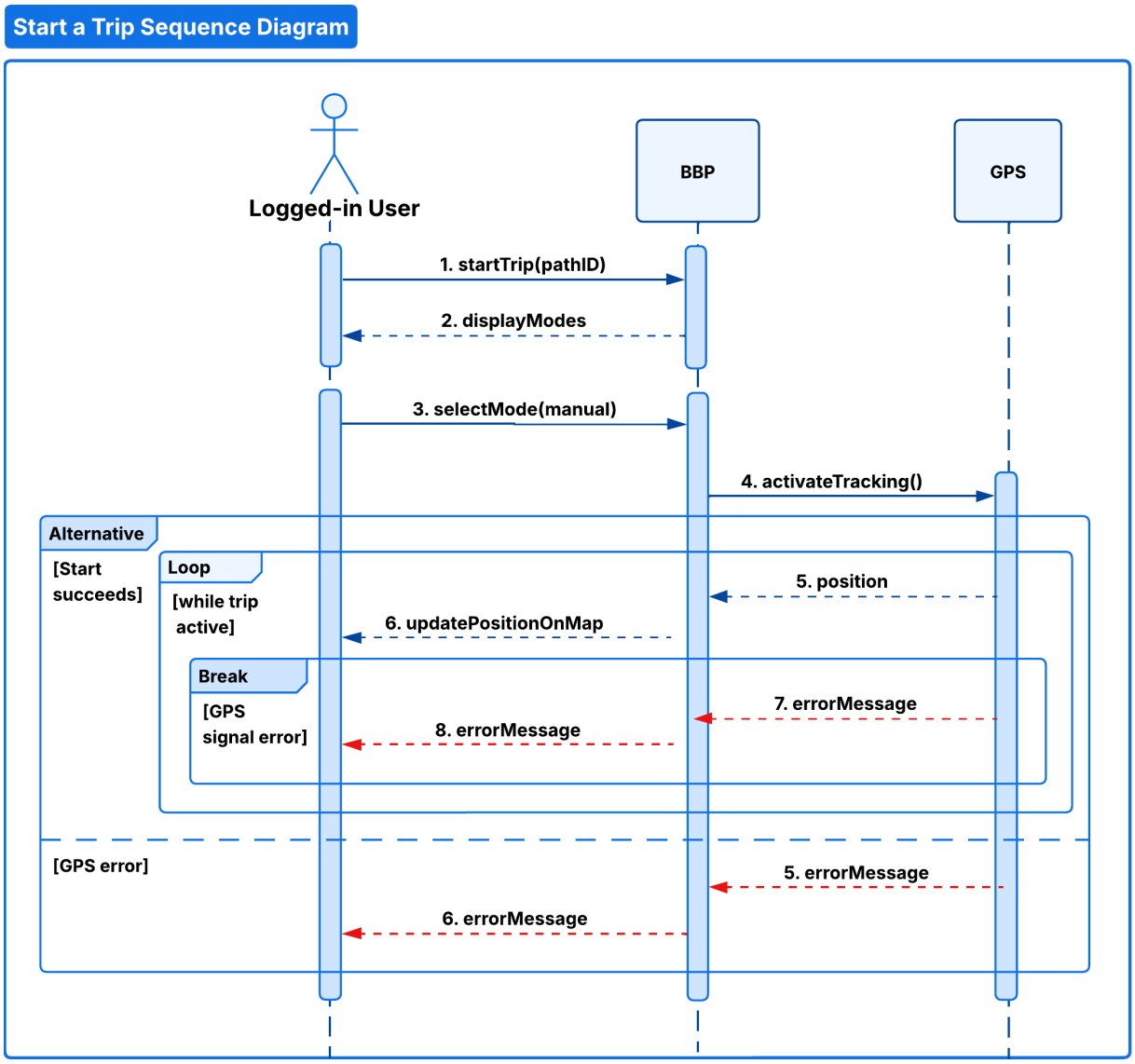


Figure 3.13: Start a Trip in Manual Mode as a Logged-in User Sequence Diagram

[UC11] - Start a Trip in Automatic Mode as a Logged-in User

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated, has selected a path to follow, and wants to start the trip in Automatic mode, using external devices.
Event Flow	1. The actor chooses to start the trip on the selected path. 2. The system displays the available start modes. 3. The actor selects Automatic mode. 4. The system attempts to connect to the configured external devices. 5. If the connection succeeds, the system confirms device readiness. 6. The system activates GPS tracking and enables trip data recording. 7. The system receives position updates from the GPS. 8. While the trip is active, the system updates the actor's position on the map and records trip samples.
Exit Condition	The trip is successfully started; live tracking, map updates, and external device streaming are active.
Exceptions	If external devices cannot be connected, the system shows an error message and does not proceed. If the GPS signal is unavailable/unstable or permissions are denied, the system shows an error message and does not start or interrupts tracking.
Notes	External devices are optional; when connected, their data are buffered locally together with GPS samples. Final validation/storage happens at trip stop. No path editing occurs in this flow.

Table 3.11: Start a Trip in Automatic Mode as a Logged-in User Process Detail

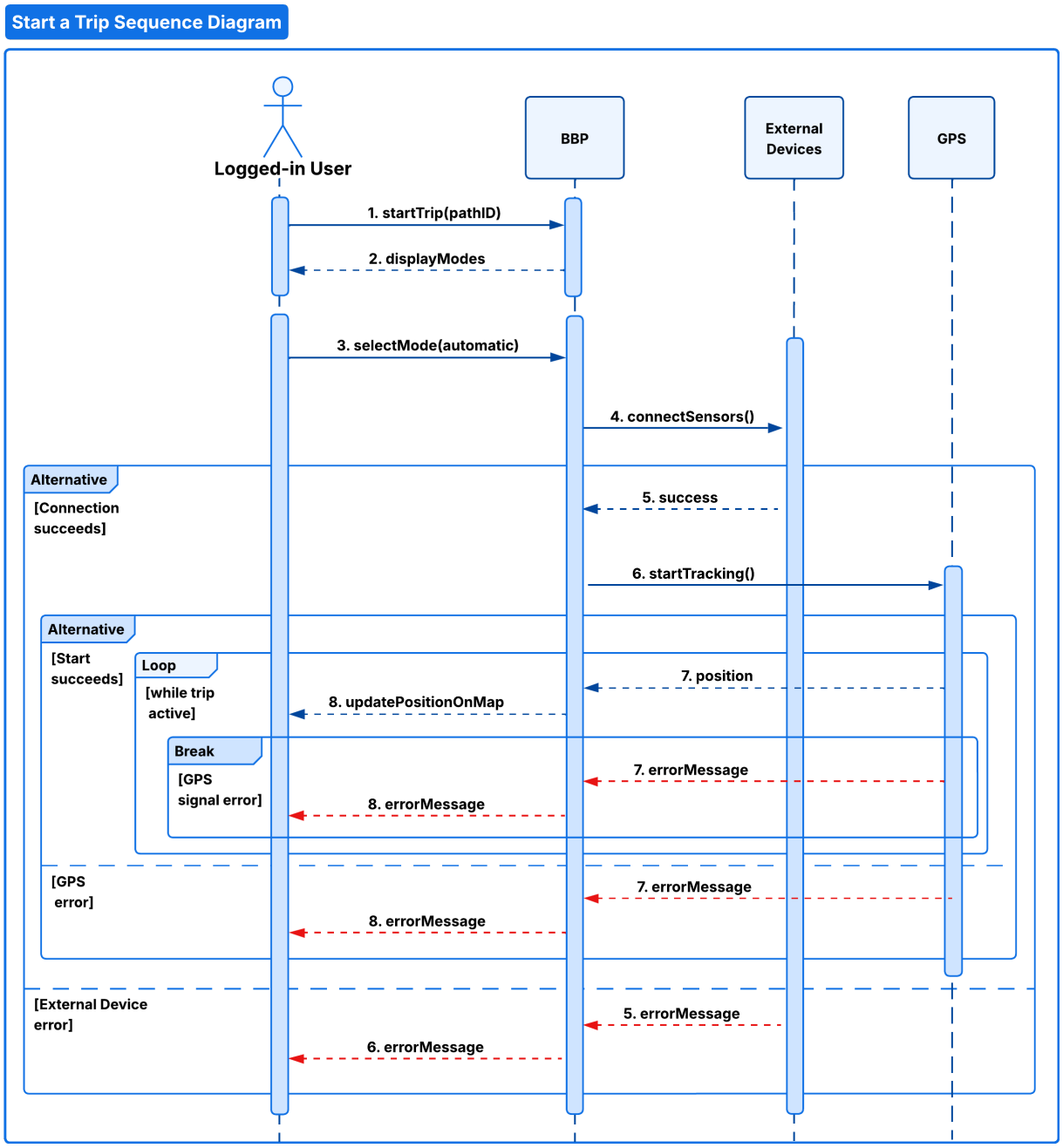


Figure 3.14: Start a Trip in Automatic Mode as a Logged-in User Sequence Diagram

[UC12] - Stop a Trip as Guest User

Actor(s)	Guest User
Entry Condition	The actor is currently on a trip, he is visualizing a map with real-time position.
Event Flow	1. The actor chooses to stop the trip or leaves the page. 2. The system stops GPS tracking and ends the map visualization.
Exit Condition	The trip visualization ends; no trip data are stored.
Exceptions	
Notes	Guest Users only visualize the trip; no recording, statistics, or reports are generated.

Table 3.12: Stop a Trip as a Guest User Process Detail

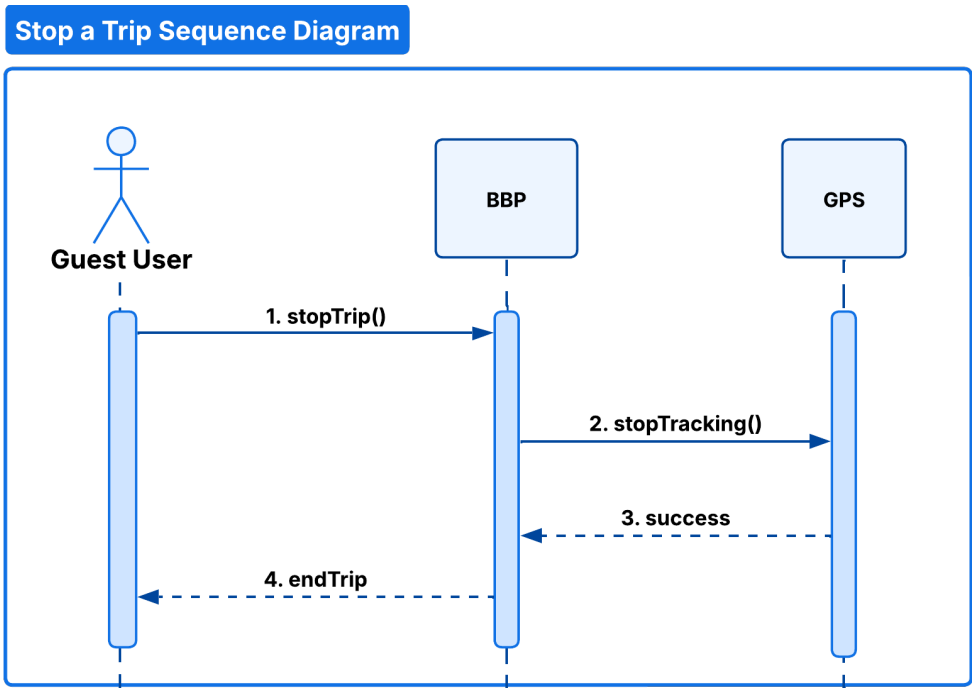


Figure 3.15: Stop a Trip as a Guest User Sequence Diagram

[UC13] - Stop a Trip as a Logged-in User

Actor(s)	Logged-in User
Entry Condition	The actor is currently performing a trip in Manual or Automatic mode.
Event Flow	1. The actor chooses to stop the current trip. 2. The system disconnects external devices (if any) and stops GPS tracking. 3. The system validates the collected data (e.g., duration, distance, reports). 5. If possible, the system enriches the trip data with meteorological information (e.g., weather, temperature), retrieved from an external service. 4. The system stores the trip data in the actor's account. 6. The system displays a confirmation and a summary view.
Exit Condition	The trip is stopped and the collected data are successfully saved.
Exceptions	If data validation fails, the system displays an error message and prevents completion until corrected. If enrichment with external data fails, the system proceeds without adding that information. If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	Once a trip is stopped, recording cannot be resumed; a new session is required to continue.

Table 3.13: Stop a Trip as a Logged-in User Process Detail

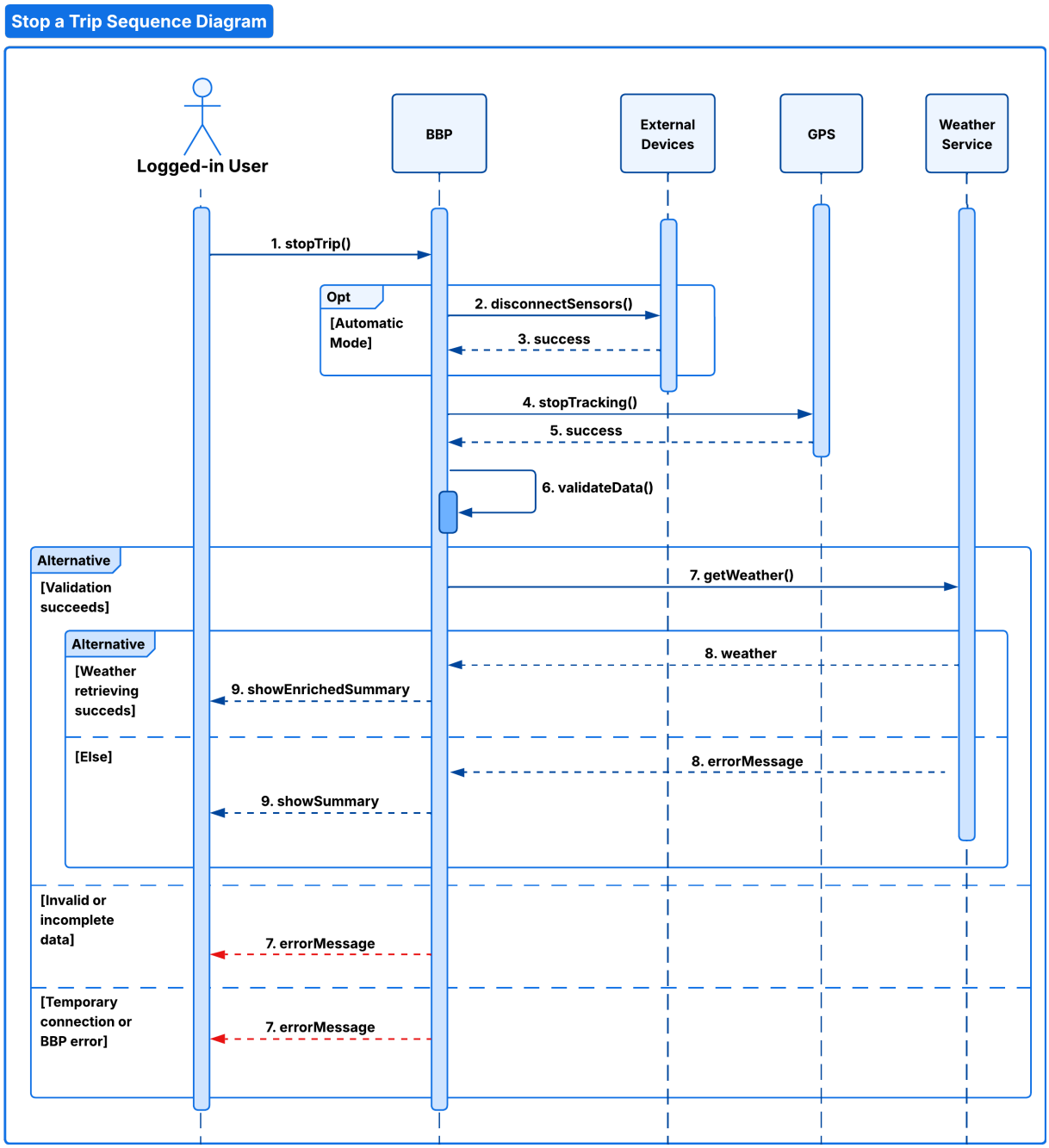


Figure 3.16: Stop a Trip as a Logged-in User Sequence Diagram

[UC14] - Make a Report in Manual Mode

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated, is on a trip, and wants to report a problem or an obstacle on the path.
Event Flow	1. The actor selects the “Report" option. 2. The system displays the reporting form. 3. The actor fills in the report data (e.g., type, description). 4. The system retrieves the actor’s current GPS position. 5. The system validates the provided data. 6. If validation succeeds, the system saves the report and links it to the position and the current trip. 7. The system confirms submission and returns to the path view.
Exit Condition	A new report is created and associated with the position and current trip.
Exceptions	If the data are invalid or incomplete, the system displays an error message and prevents submission until corrected. If GPS position cannot be retrieved, the system displays an error message and allows retry. If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	The report includes timestamp and geolocation. Reports will impact path ranking.

Table 3.14: Make a Report in Manual Mode Process Detail

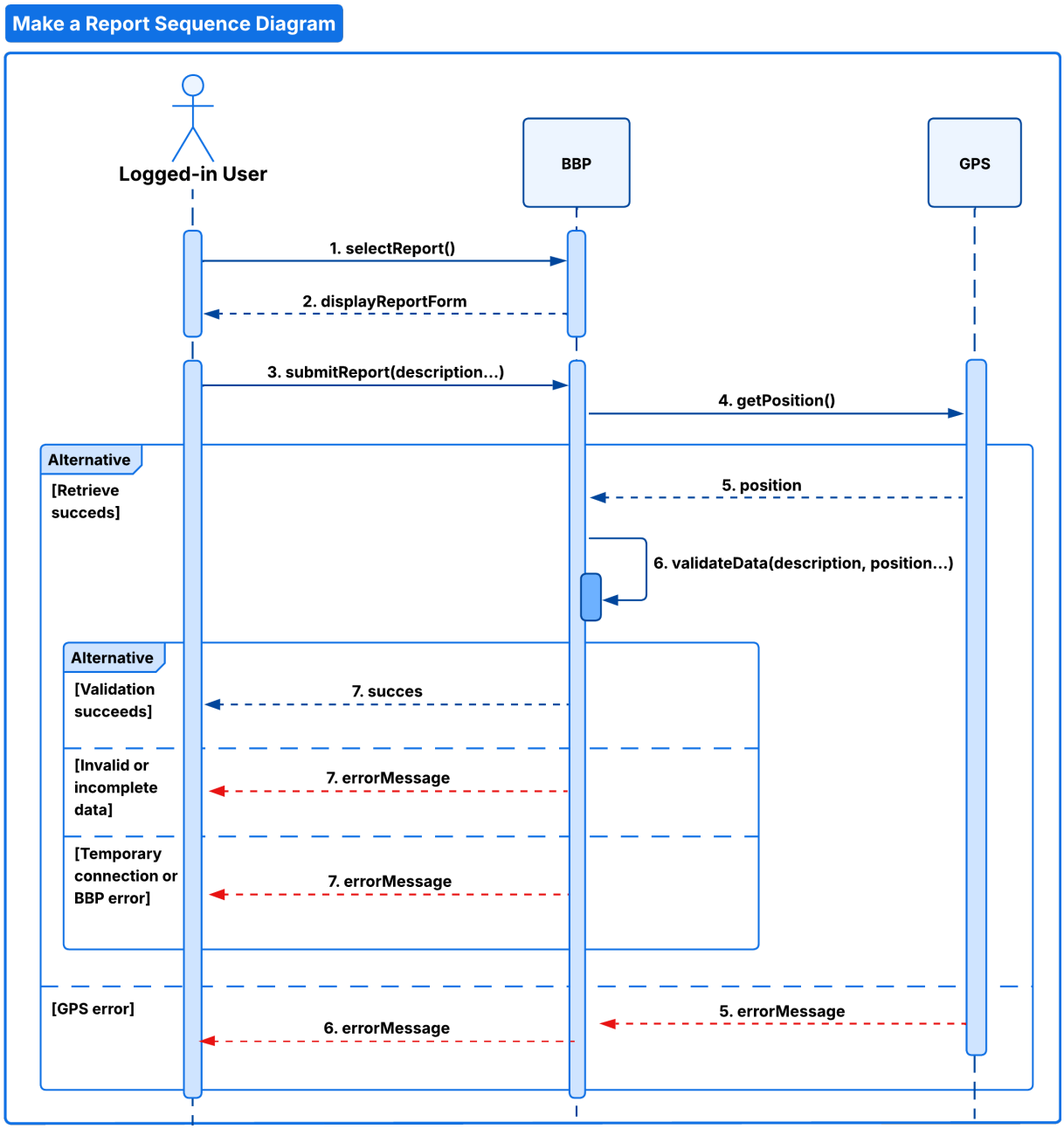


Figure 3.17: Make a Report in Manual Mode Sequence Diagram

[UC15] - Make a Report in Automatic Mode

Actor(s)	Logged-in User
Entry Condition	The actor is performing a trip in Automatic mode; external devices/sensors are active.
Event Flow	1. A connected sensor detects a potential issue on the path. 2. The system displays a form with a prefilled report (e.g., type, timestamp, location). 3. The actor chooses whether to review/edit the data and submit the report, or dismiss the form. 4. The system retrieves the actor's current GPS position. 5. The system validates the provided data. 6. If validation succeeds, the system saves the report and links it to the position and the current trip. 7. The system confirms submission and returns to the trip view.
Exit Condition	The report is created and associated with the selected position and current trip, or the form is dismissed and the trip continues.
Exceptions	If the data are invalid or incomplete, the system displays an error message and prevents submission until corrected. If the GPS position cannot be determined, the system displays an error message and allows the actor to manually enter a location. If a temporary connection or BBP error occurs, the system displays an error message and allows retry later.
Notes	If the actor dismisses the form, no report is created and the trip proceeds. Prefilled data come from sensors (e.g., gyroscope).

Table 3.15: Make a Report in Automatic Mode Process Detail

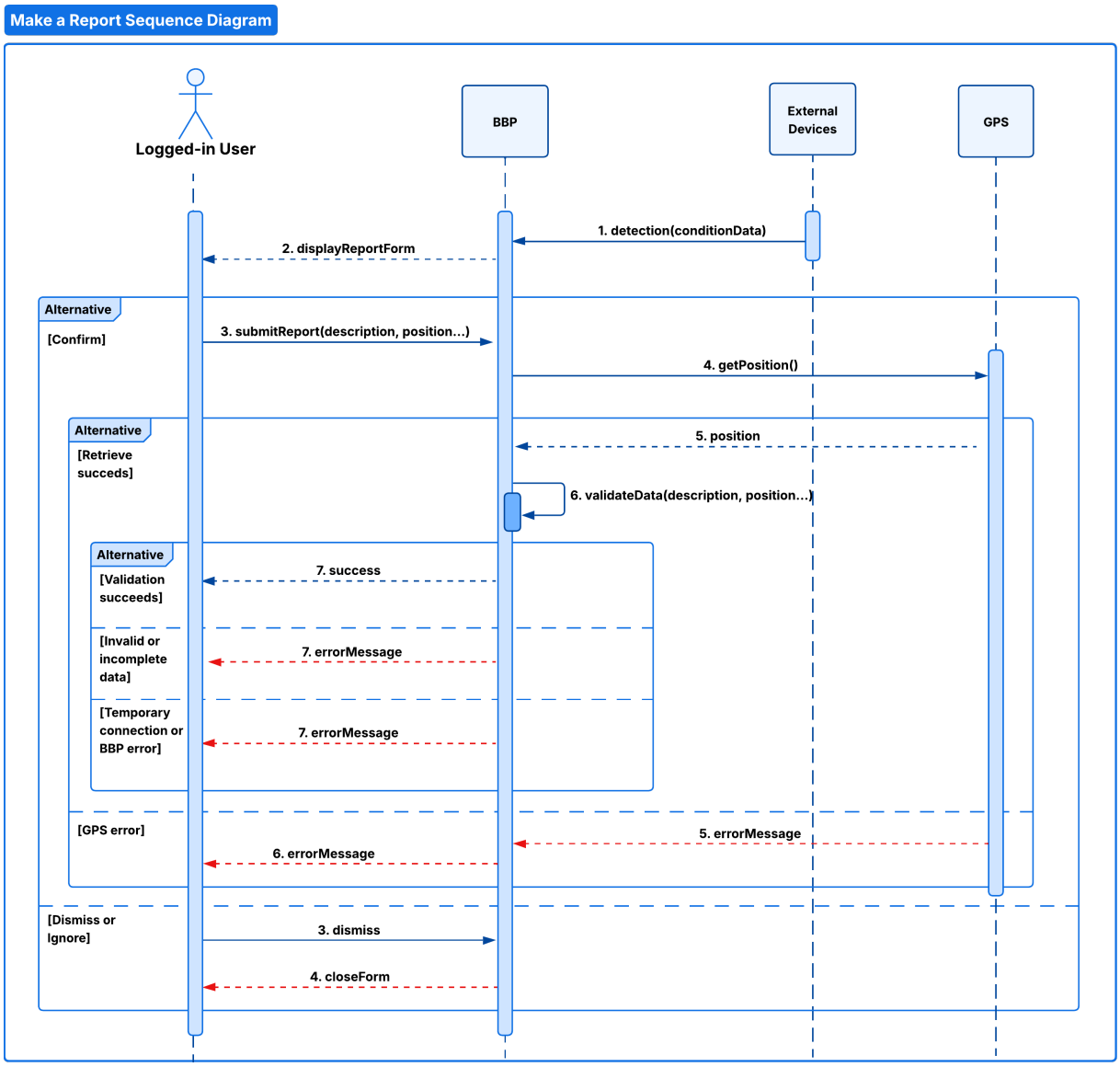


Figure 3.18: Make a Report in Automatic Mode Sequence Diagram

[UC16] - Confirm a Report

Actor(s)	Logged-in User
Entry Condition	The actor is currently performing a trip. The system already knows the actor's current position through GPS tracking and can detect proximity to path segments with existing reports.
Event Flow	1. While following the path, the system detects proximity to a segment with existing reports based on the actor's current position. 2. The system displays a popup summarizing the reported issue (e.g., "Users have reported a pothole here. Do you confirm it?"). 3. The actor chooses one of the available options: **Confirm:** the system registers the actor's confirmation, increasing the report's reliability. **Reject:** the system registers the actor's feedback as "not confirmed". **Ignore:** the popup is dismissed without any action. 4. The trip continues normally.
Exit Condition	The system updates the report's confirmation status (confirmed, rejected, or unchanged).
Exceptions	If the data are invalid or incomplete, the system displays an error message. If a temporary connection or BBP error occurs and the feedback cannot be saved, the system displays an error message.
Notes	The feature enables collaborative validation of reports. Multiple confirmations increase a report's reliability, while rejections decrease it. Guest Users are not prompted for confirmations.

Table 3.16: Confirm a Report Process Detail

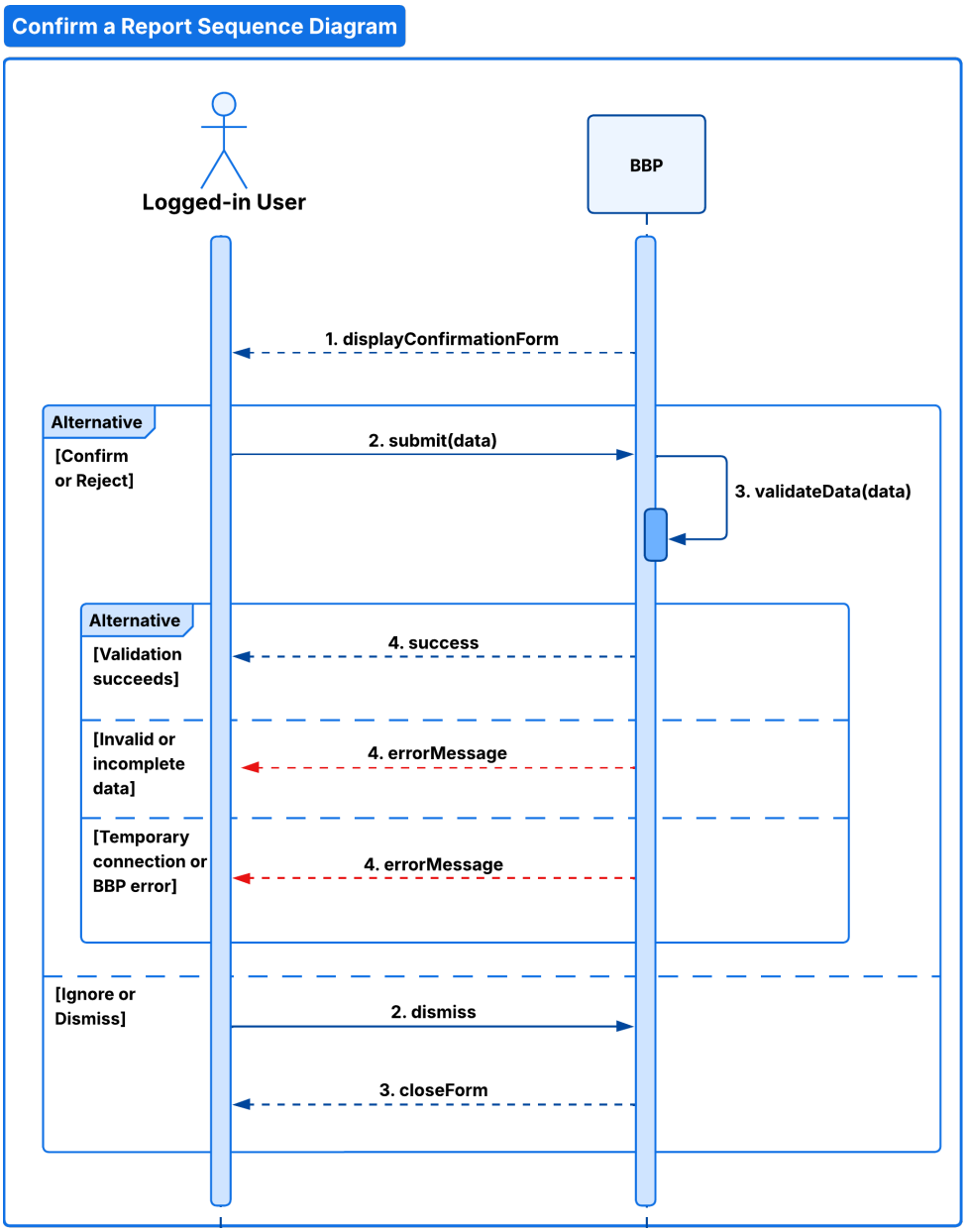


Figure 3.19: Confirm a Report Sequence Diagram

[UC17] - Manage Path Visibility

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and is viewing the details of one of their created paths, intending to change its visibility.
Event Flow	1. The actor selects "Edit visibility" for a specific path. 2. The system retrieves and displays the current visibility setting (e.g., Public, Private). 3. The actor selects a new visibility option and confirms. 4. The system validates the input and sends an update request. 5. If the operation succeeds, the system confirms the update and refreshes the path details.
Exit Condition	The path visibility is successfully updated according to the actor's selection.
Exceptions	If the path is not found, the system notifies the actor. If the actor is not the path owner, the system denies the operation and shows an appropriate error message. If a temporary connection or BBP error occurs, the system displays an error message and allows retry. If the input is invalid or incomplete, the system displays an error and asks for correction.
Notes	This operation modifies only a metadata field ("visibility") and does not affect the path geometry or statistics. Public paths are visible to other users according to platform rules, while private paths are accessible only to the owner.

Table 3.17: Manage Path Visibility Process Detail

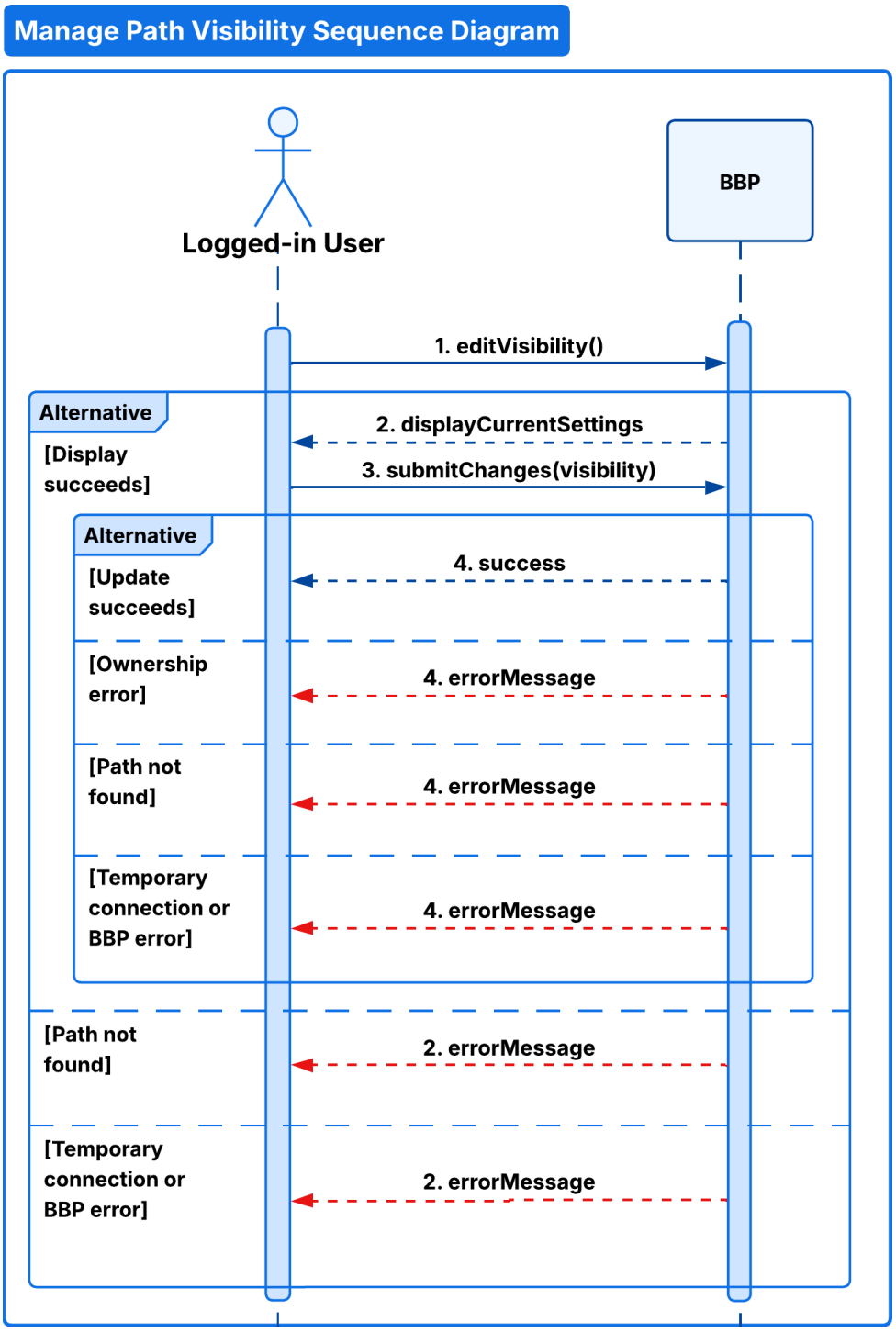


Figure 3.20: Manage Path Visibility Sequence Diagram

[UC18] - View Trip History and Trip Details

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and has completed at least one trip.
Event Flow	1. The actor accesses the "Trip History" section from the profile or main menu. 3. The system displays the trips associated with the actor chronologically. 4. The actor can selects a specific trip to view detailed data (map, statistics, reports). 5. The system retrieves and displays the detailed information for the selected trip.
Exit Condition	The actor can visualize the details of one or more past trips.
Exceptions	If no trips are found, the system shows "No trips available". If no trip details are available, the system displays an error message. If connection to BBP fails, the system shows an error and allows retry.
Notes	The system may allow sorting or filtering by date, distance, or ranking.

Table 3.18: View Trip History Process Detail

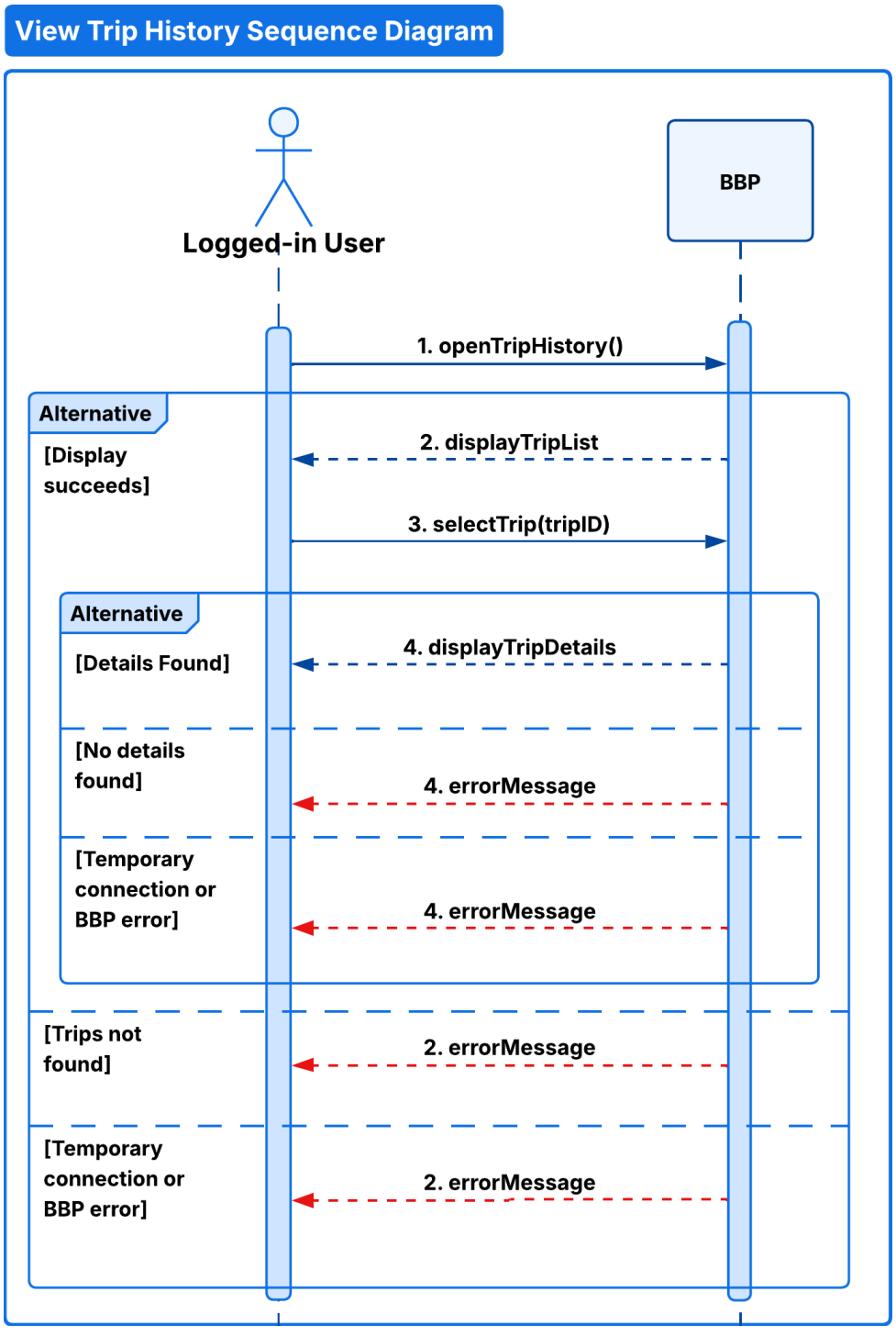


Figure 3.21: View Trip History and Trip Details Sequence Diagram

[UC19] - View Overall Statistics

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and has completed one or more trips.
Event Flow	1. The actor opens the "Statistics" section from the profile or dashboard. 2. The system aggregates data from all completed trips (e.g., total distance, average speed, total duration). 3. The system displays overall metrics and summaries.
Exit Condition	The actor visualizes overall statistics about their activity.
Exceptions	If no trip data are available, the system displays "No statistics available". If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	These statistics are private.

Table 3.19: View Overall Statistics Process Detail

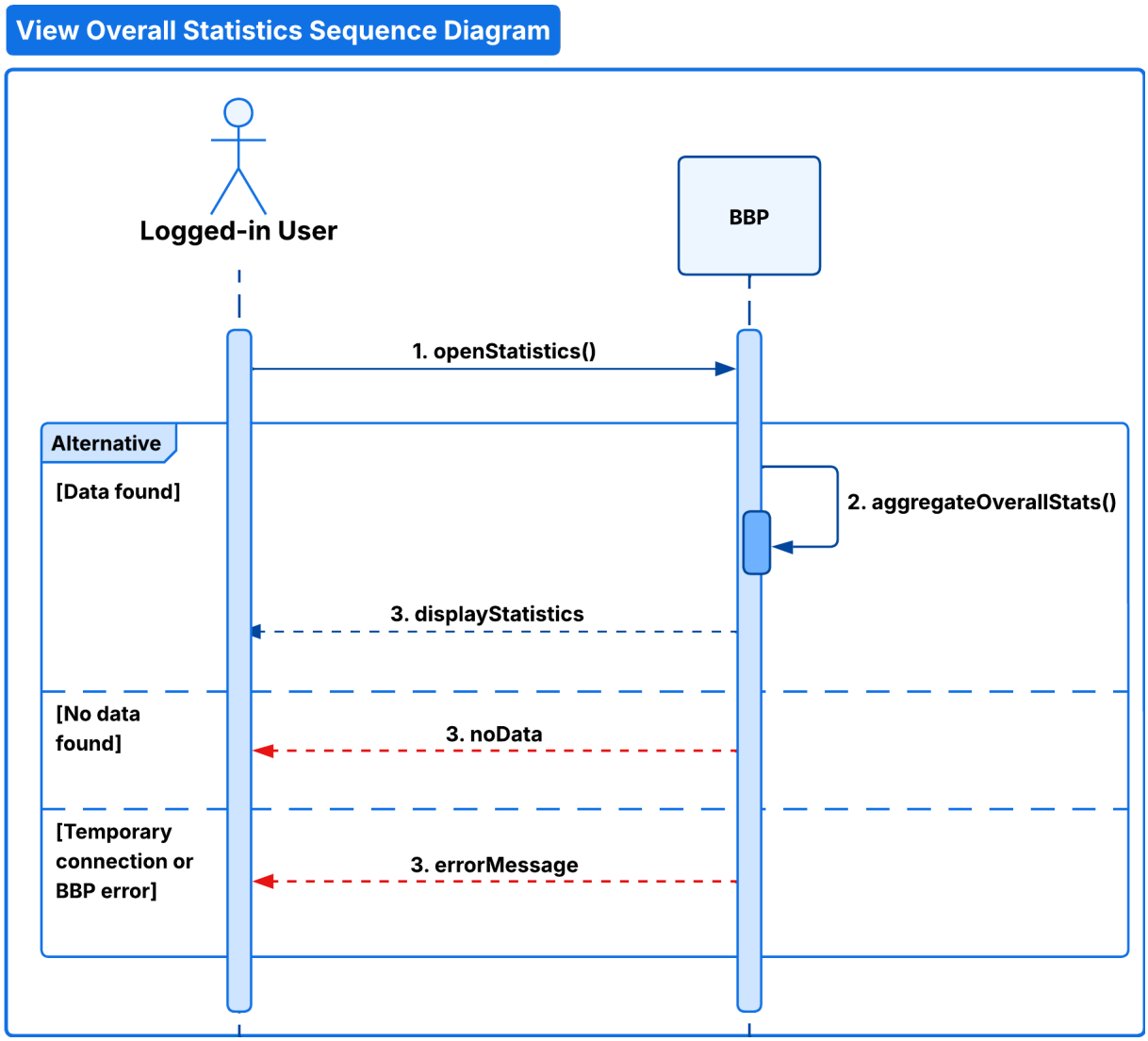


Figure 3.22: View Overall Statistics Sequence Diagram

[UC20] - View Trip Statistics

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and has selected a trip from the trip history.
Event Flow	1. The actor selects a trip to view its detailed statistics. 2. The system retrieves all data recorded during that trip (e.g., duration, speed graph, distance, elevation, number of reports). 3. The system displays the trip's detailed statistics.
Exit Condition	The actor views the trip's performance metrics and related information.
Exceptions	IIf no trip data are available, the system shows an error message. If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	This information is visible only to the trip owner.

Table 3.20: View Trip Statistics Process Detail

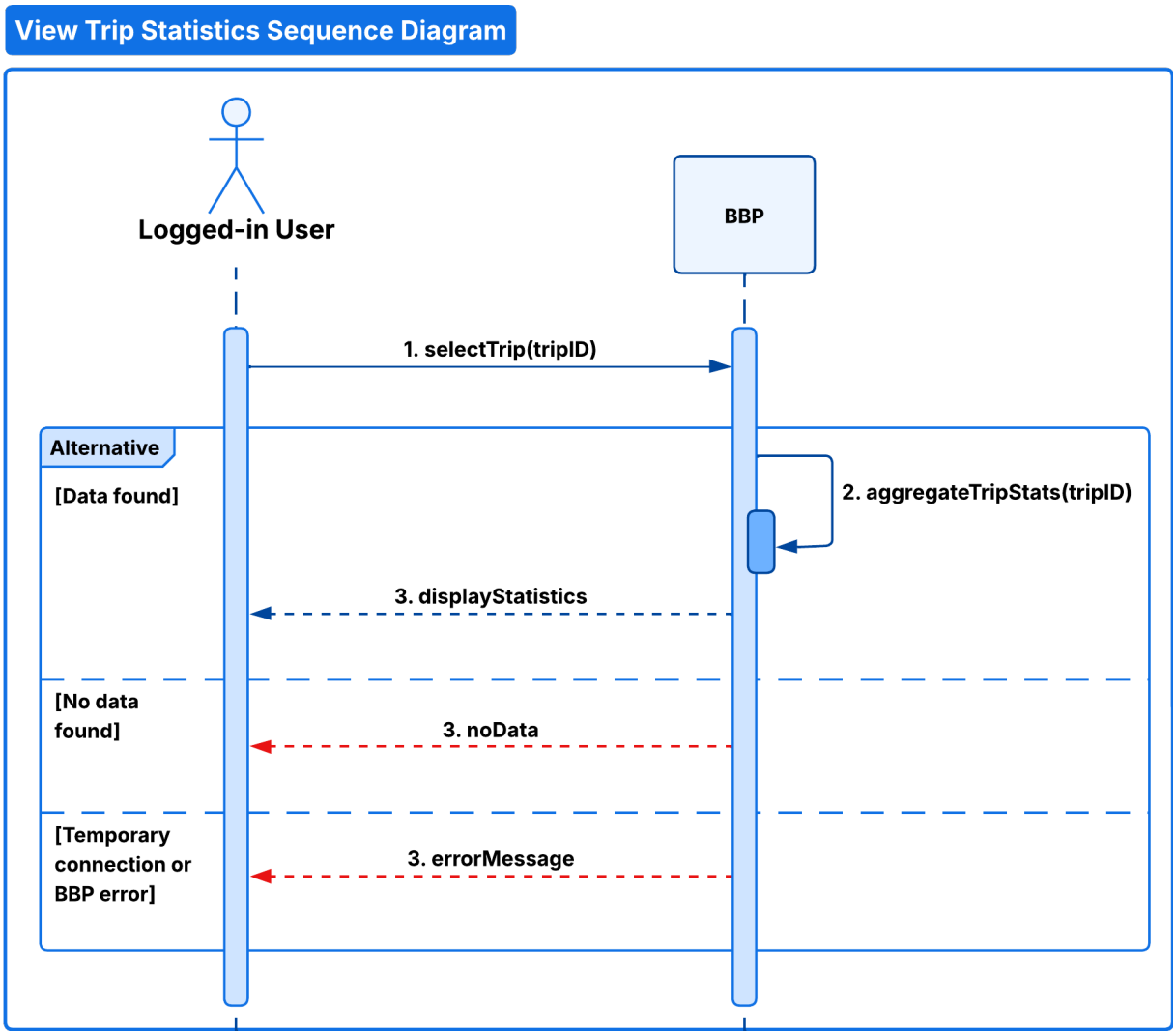


Figure 3.23: View Trip Statistics Sequence Diagram

[UC21] - Edit Personal Profile

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and is on the personal profile page.
Event Flow	1. The actor selects the "Edit Profile" action. 2. The system displays the editable profile form (e.g., username). 3. The actor modifies one or more fields and submits the changes. 4. The system validates the provided data (e.g., required fields). 5. If validation succeeds, the system saves the changes and updates the profile view.
Exit Condition	The actor's profile information is updated successfully.
Exceptions	If data are invalid or incomplete, the system displays an error message and prevents saving until corrected. If a temporary connection or BBP error occurs, the system displays an error message and allows retry.
Notes	Changing sensitive data (e.g., email or password) may require re-authentication and/or email verification. Avatar uploads must respect size and format limits.

Table 3.21: Edit Personal Profile Process Detail

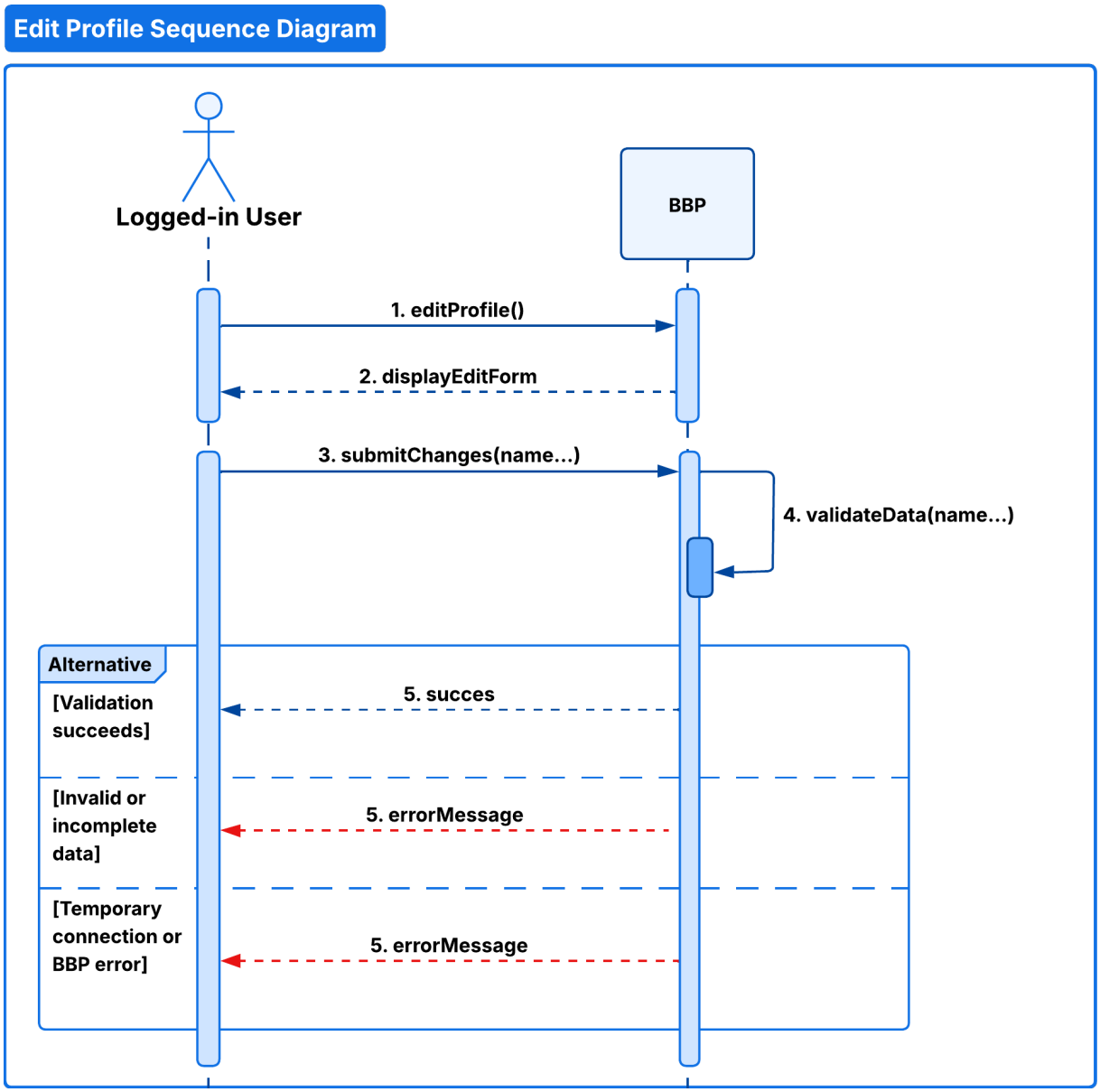


Figure 3.24: Edit Personal Profile Sequence Diagram

3.2.4. Requirement Mapping

Goal	Requirements	Domain Assumptions
G1	R5, R25, R26, R27, R28, R29, R30, R31, R32, R35, R36	D1, D5, D6, D8
G2	R5, R25, R26, R27, R28, R29, R30, R31, R32, R35, R36	D1, D5, D6, D8
G3	R5, R8, R9, R25, R28, R29, R30, R33, R34, R35, R36	D1, D3, D4, D6, D7, D8
G4	R1, R2, R3, R4, R6, R7, R8, R9, R10, R13, R14, R15, R16, R35, R36	D1, D2, D3, D4, D6
G5	R1, R2, R6, R8, R9, R16, R17, R18, R19, R20, R21, R22, R23, R24, R35, R36	D1. D3, D4, D5, D6, D7, D8
G6	R1, R2, R3, R4, R6, R7, R8, R9, R10, R11, R12, R15, R16, R35, R36	D1. D2, D3, D4, D6

Table 3.22: Mapping between goals, requirements, and domain assumptions

3.3. Performance Requirements

3.4. Design Constraints

The design constraints represent external factors and limitations that influence how the BBP system can be developed and operated.

3.4.1. Standards Compliance

The system must comply with current privacy and data protection regulations, in particular with the GDPR (General Data Protection Regulation, EU 2016/679). All collected data, such as location and trip information, must be processed only for the purposes explicitly accepted by the user. The system must also follow accessibility and usability guidelines defined by the major mobile platforms.

3.4.2. Hardware Limitations

The application requires a smartphone equipped with GPS and a stable Internet connection to ensure correct functionality. Accuracy of certain features, such as automatic path

recording or nearby alerts, may depend on the quality of the device's sensors and network coverage.

3.5. Software System Attributes

This section describes the main quality attributes that the BBP system must satisfy to ensure a good performance and overall user experience.

3.5.1. Reliability

The system must ensure reliable operation over time, minimizing crashes and data loss during trip recording or report submission. All critical data, such as user profiles and path information, must be stored and synchronized safely with the server. Any detected malfunction or bug that affects reliability must be fixed as soon as possible through updates distributed via the app stores.

3.5.2. Availability

The BBP system must be available at all times, especially since many of its functions (such as navigation and path recording) may be used during trips. The backend services should guarantee an uptime of at least 99%, with maintenance scheduled during off-peak hours.

3.5.3. Security

The system manages personal and location data, which must be protected from unauthorized access. All communications between the app and the server must occur over HTTPS. User passwords must be securely stored on the server using strong hashing algorithms. Access to user data and sensitive operations must be restricted through authentication tokens and permission control.

3.5.4. Maintainability

The system must be designed with a modular and service-oriented architecture to simplify maintenance and future extensions. Code should be clearly documented and follow standard conventions for both client and server components. This will allow independent updates of single modules without compromising the overall functionality.

3.5.5. Portability

The application must be compatible with both Android and iOS platforms. It should adapt to different screen sizes and hardware capabilities while maintaining consistent behavior and appearance. The backend must remain platform-independent, ensuring that future client versions (e.g., a desktop or web app) can be integrated without major redesigns.

4 | Formal analysis using alloy

The main objective of the Alloy model presented here is to structure and describe the domain and the behavior of a user selecting an origin and a destination, choosing a suggested path, starting a trip, moving through path segments, and eventually stopping it.

Users are divided into two categories: Guest and LoggedInUser. Through the specification of appropriate predicates, it is possible to highlight the differences between the processes carried out by these two user types. In particular, LoggedInUsers have access to additional functionality, such as viewing the history of completed trips.

We did not include features related to data acquisition modes (manual or automatic) or statistics, as the focus of the model is on the path selection process and the management of the trip itself.

In this model, a location represents an area within which multiple path segments may be defined, and in which a user may be considered present. This notion of location does not represent the user's exact GPS position but rather their presence within a broader area. This simplification avoids the need to model precise user coordinates, which are not relevant to the purpose of the model.

Finally, a set of assertions has been defined to verify the correctness of the model and its fundamental properties, without manually exploring all possible system states.

Below are the main components and constraints of the developed Alloy model.

```
enum Boolean { True, False }

abstract sig User {
  var has_selected: lone Path,
  var currentLocation: one Location,
  var current_trip: lone Trip,
  var trip_origin: lone Location,
  var trip_destination: lone Location,
  var suggested_paths: set Path
}
```

```
sig Guest extends User {}

sig LoggedInUser extends User {
  var completed_trips: set Trip,
}

var sig Trip {
  var bike_path: lone Path,
  var has_started: one Boolean,
  var has_finished: one Boolean,
  var current_segment: lone PathSegment
}

var sig ActiveTrip, CompletedTrip in Trip {}

sig Path {
   segments: some PathSegment,
   origin: one Location,
   destination: one Location
}

sig Location {}

sig PathSegment {
  segmentLocation: one Location,
  nextSegment: lone PathSegment
}

fact init {
  #LoggedInUser = 1
  #Guest = 1
  #Path = 2

  no has_selected
  no completed_trips
  no trip_origin
  no trip_destination
  no suggested_paths
  no current_trip
  no Trip
  no ActiveTrip
  no CompletedTrip
}

fact segHasPath {
   all s: PathSegment | some p: Path | s in p.segments
}

fact pathStructure {
  all p: Path | {
    p.origin ≠ p.destination

    some s: p.segments | s.segmentLocation = p.origin and
      (no s2: p.segments | s2.nextSegment = s)

    some s: p.segments | s.segmentLocation = p.destination and
```

```
      no s.nextSegment

      all s: p.segments | (some s.nextSegment) implies (s.nextSegment in p.segments)

    all s: p.segments | s not in s.^nextSegment

    let first = {s: p.segments | s.segmentLocation = p.origin and
                  no s2: p.segments | s2.nextSegment = s} |
    let reachable = first.*nextSegment |
    p.segments = reachable
  }
}

fact liveTripConsistency {
   always all t: Trip | (t in ActiveTrip + CompletedTrip) iff some bike_path[t]
}

fact currentTripConsistency {
  always (
    all t: ActiveTrip | one u: User | u.current_trip = t
    and all u: User | lone u.current_trip
    and all u: User | some u.current_trip implies (
      u.current_trip in ActiveTrip
      and has_selected[u] = bike_path[u.current_trip]
    )
  )
}

fact completedTripsDisjoint {
  always no (ActiveTrip & CompletedTrip)
}

fact tripBooleanConsistency {
  always all t: Trip | one has_started[t] and one has_finished[t]
}
```

To make the model as realistic as possible, some mutable signatures have been defined, such as ActiveTrip and CompletedTrip. These signatures enable the representation of the dynamic evolution of entities within the system, allowing the model to more accurately capture the system's behavior over time.

Furthermore, a set of facts has been introduced to ensure the consistency and integrity of the model. For example, the pathStructure fact guarantees that all paths are correctly formed, free of cycles, and fully connected between their origin and destination. Other facts, such as liveTripConsistency and currentTripConsistency, ensure that the state of Trip and User entities remains coherent with the operations that can be performed in the system.

For simplicity, paths are assumed to be unidirectional, since the possibility for a user to move backwards along a path is not considered in this model.

An init fact is also defined to initialize the system in its starting state. In this simplified initial configuration, the model begins with one Guest user, one LoggedInUser, and two predefined paths already present in the system.

We now show the main actions a user can perform within the system, modeled using predicates in Alloy. Note that each predicate contains frame conditions, which ensure that all relationships not involved in the action remain unchanged.

```
pred selectOriginDestination[u: User, dest: Location] {
  // PRECONDITIONS
  no u.trip_origin
  no u.trip_destination
  no u.current_trip
  u.currentLocation ≠ dest

  // POSTCONDITIONS
  trip_origin' = trip_origin + (u -> u.currentLocation)
  trip_destination' = trip_destination + (u -> dest)

  // FRAME CONDITIONS
  Trip' = Trip
  ActiveTrip' = ActiveTrip
  CompletedTrip' = CompletedTrip
  has_selected' = has_selected
  currentLocation' = currentLocation
  current_trip' = current_trip
  completed_trips' = completed_trips
  bike_path' = bike_path
  has_started' = has_started
  has_finished' = has_finished
  current_segment' = current_segment
  suggested_paths' = suggested_paths
}
```

A user begins the route selection process by specifying an origin and a destination. The destination must be different from the origin. Furthermore, for simplicity, it is assumed that the user's current location is always the origin of the selected route. This is because, in reality, the user could navigate between multiple locations, but would not be able to begin the journey if they were at a location other than the origin of the chosen cycling route.

```
pred suggestPaths[u: User] {
  // PRECONDITIONS
  some u.trip_origin
  some u.trip_destination
  no u.suggested_paths
```

```
  no u.has_selected

  // POSTCONDITIONS
  let matchingPaths = {p: Path | p.origin = u.trip_origin and p.destination = u.
      ↪ trip_destination} |
  some matchingPaths and
  suggested_paths' = suggested_paths + (u -> matchingPaths)

  // FRAME CONDITIONS
  Trip' = Trip
  ActiveTrip' = ActiveTrip
  CompletedTrip' = CompletedTrip
  has_selected' = has_selected
  currentLocation' = currentLocation
  current_trip' = current_trip
  completed_trips' = completed_trips
  bike_path' = bike_path
  has_started' = has_started
  has_finished' = has_finished
  current_segment' = current_segment
  trip_origin' = trip_origin
  trip_destination' = trip_destination
}
```

This model assumes that a route is suggested simply, without considering factors such as ranking or the route's status as rated by users.

```
pred userSelectsPath[u: User, p: Path] {
  // PRECONDITIONS
  p in u.suggested_paths
  no u.has_selected
  no u.current_trip

  // POSTCONDITIONS
  has_selected' = has_selected + (u -> p)

  // FRAME CONDITIONS
  Trip' = Trip
  ActiveTrip' = ActiveTrip
  CompletedTrip' = CompletedTrip
  currentLocation' = currentLocation
  current_trip' = current_trip
  completed_trips' = completed_trips
  bike_path' = bike_path
  has_started' = has_started
  has_finished' = has_finished
  trip_origin' = trip_origin
  trip_destination' = trip_destination
  suggested_paths' = suggested_paths
}

// STEP 4: Start the trip
pred startTrip[u: User] {
  // PRECONDITIONS
  some u.has_selected
```

```
  no u.current_trip
  u.currentLocation = u.trip_origin

  let firstSeg = {s: u.has_selected.segments |
    s.segmentLocation = u.trip_origin and
    no s2: u.has_selected.segments | s2.nextSegment = s} |

  // POSTCONDITIONS
  some t: Trip' - Trip |
    Trip' = Trip + t
    and ActiveTrip' = ActiveTrip + t
    and bike_path' = bike_path + (t -> u.has_selected)
    and has_started' = has_started + (t -> True)
    and has_finished' = has_finished + (t -> False)
    and current_trip' = current_trip + (u -> t)
    and current_segment' = current_segment + (t -> firstSeg)

  // FRAME CONDITIONS
  has_selected' = has_selected
  currentLocation' = currentLocation
  completed_trips' = completed_trips
  CompletedTrip' = CompletedTrip
  trip_origin' = trip_origin
  trip_destination' = trip_destination
  suggested_paths' = suggested_paths
}

pred moveAlongPath[u: User] {
  // PRECONDITIONS
  some u.current_trip
  u.current_trip in ActiveTrip
  has_started[u.current_trip] = True
  has_finished[u.current_trip] = False
  some current_segment[u.current_trip]
  some current_segment[u.current_trip].nextSegment

  let currSeg = current_segment[u.current_trip] |
  let nextSeg = currSeg.nextSegment |

  // POSTCONDITIONS
  current_segment' = current_segment - (u.current_trip -> PathSegment) + (u.current_trip
      ↪ -> nextSeg)
  and currentLocation' = currentLocation - (u -> Location) + (u -> nextSeg.
      ↪ segmentLocation)

  // FRAME CONDITIONS
  Trip' = Trip
  ActiveTrip' = ActiveTrip
  CompletedTrip' = CompletedTrip
  has_selected' = has_selected
  current_trip' = current_trip
  completed_trips' = completed_trips
  bike_path' = bike_path
  has_started' = has_started
  has_finished' = has_finished
  trip_origin' = trip_origin
```

```
  trip_destination' = trip_destination
  suggested_paths' = suggested_paths
}
```

It is assumed that if two path segments are in the same location, the user always stays in that location but if the next path segment is in a different location, the user moves accordingly.

```
pred stopTrip[u: User] {
  // PRECONDITIONS
  some u.current_trip
  u.current_trip in ActiveTrip
  has_started[u.current_trip] = True
  u.currentLocation = u.trip_destination
  no current_segment[u.current_trip].nextSegment

  // POSTCONDITIONS
  has_finished' = has_finished - (u.current_trip -> Boolean) + (u.current_trip -> True)

  // FRAME CONDITIONS
  Trip' = Trip
  ActiveTrip' = ActiveTrip
  CompletedTrip' = CompletedTrip
  has_selected' = has_selected
  currentLocation' = currentLocation
  current_trip' = current_trip
  completed_trips' = completed_trips
  bike_path' = bike_path
  has_started' = has_started
  trip_origin' = trip_origin
  trip_destination' = trip_destination
  suggested_paths' = suggested_paths
  current_segment' = current_segment
}
```

Since the model does not account for the possibility of a user abandoning a trip midway, it is assumed that the user can only stop the trip upon reaching the destination and after traversing all path segments.

```
pred finalizeTrip[u: User] {
  // PRECONDITIONS
  some u.current_trip
  u.current_trip in ActiveTrip
  has_finished[u.current_trip] = True

  let t = u.current_trip |

  // POSTCONDITIONS
  ActiveTrip' = ActiveTrip - t
  and current_trip' = current_trip - (u -> Trip)
  and has_selected' = has_selected - (u -> Path)
  and trip_origin' = trip_origin - (u -> Location)
```

```
  and trip_destination' = trip_destination - (u -> Location)
  and suggested_paths' = suggested_paths - (u -> Path)
  and current_segment' = current_segment - (t -> PathSegment)

  and ((u in LoggedInUser) implies
        (Trip' = Trip
         and CompletedTrip' = CompletedTrip + t
         and bike_path' = bike_path
         and has_started' = has_started
         and has_finished' = has_finished
         and completed_trips' = completed_trips + (u -> t)))

  and ((u not in LoggedInUser) implies
        (Trip' = Trip - t
         and CompletedTrip' = CompletedTrip
         and bike_path' = bike_path - (t -> Path)
         and has_started' = has_started - (t -> Boolean)
         and has_finished' = has_finished - (t -> Boolean)
         and completed_trips' = completed_trips))

  // FRAME CONDITIONS
  currentLocation' = currentLocation
}
```

The finalizeTrip predicate handles the completion of a trip differently based on the user's type. LoggedInUsers retain a record of their completed trips, while Guest users have their trip data removed from the system upon finalization. Furthermore this predicates ensures to have an extra step where is possibile to see the transtition from ActiveTrip to CompletedTrip.

```
pred do_something_else {
  // POSTCONDITIONS = FRAME CONDITIONS
  Trip' = Trip
  ActiveTrip' = ActiveTrip
  CompletedTrip' = CompletedTrip
  bike_path' = bike_path
  has_started' = has_started
  has_finished' = has_finished
  has_selected' = has_selected
  currentLocation' = currentLocation
  current_trip' = current_trip
  completed_trips' = completed_trips
  trip_origin' = trip_origin
  trip_destination' = trip_destination
  suggested_paths' = suggested_paths
  current_segment' = current_segment
}
```

The do_something_else predicate represents a stuttering step in which the system performs no significant action with respect to the modeled entities and all mutable relations remain unchanged. This allows the model to capture periods of inactivity or transi-

tions without observable effects, making the execution traces more realistic. In addition, do_something_else ensures that the trans fact can always be satisfied: when none of the main events is enabled (because their preconditions fail), the system can still progress by taking a do_something_else step instead of "hanging" in a deadlocked state.

```
fact trans {
  always (
    (some u: User, d: Location | selectOriginDestination[u, d]) or
    (some u: User | suggestPaths[u]) or
    (some u: User, p: Path | userSelectsPath[u, p]) or
    (some u: User | startTrip[u]) or
    (some u: User | moveAlongPath[u]) or
    (some u: User | stopTrip[u]) or
    (some u: User | finalizeTrip[u]) or
    do_something_else
  )
}
```

The predicate trans defines the possible transitions that can occur in the system at any given time. It ensures that at each step, one of the defined actions can be executed, or the system can remain in a state of inactivity through the do_something_else predicate. This approach allows the model to capture the dynamic behavior of the system while ensuring that it can always progress or maintain its state without getting stuck.

Finally, we introduce the predicates used to execute the model and the assertions that verify its correctness. We intentionally do not define a single predicate combining both user types, since running their scenarios separately makes the resulting traces clearer and avoids confusion.

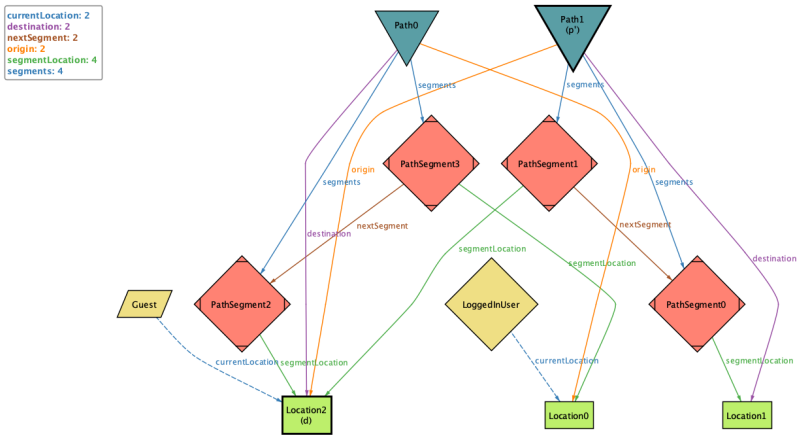
```
pred guestCompleteCycle {
  eventually (some d: Location | selectOriginDestination[Guest, d])
  and eventually suggestPaths[Guest]
  and eventually (some p: Path | userSelectsPath[Guest, p])
  and eventually startTrip[Guest]
  and eventually moveAlongPath[Guest]
  and eventually stopTrip[Guest]
  and eventually finalizeTrip[Guest]
}

pred loggedCompleteCycle {
  eventually (some d: Location | selectOriginDestination[LoggedInUser, d])
  and eventually suggestPaths[LoggedInUser]
  and eventually (some p: Path | userSelectsPath[LoggedInUser, p])
  and eventually startTrip[LoggedInUser]
  and eventually moveAlongPath[LoggedInUser]
  and eventually stopTrip[LoggedInUser]
  and eventually finalizeTrip[LoggedInUser]
  and eventually #LoggedInUser.completed_trips = 2
}
```

```
run guestCompleteCycle for 4 but exactly 3 Location, exactly 4 PathSegment, 15 steps
run loggedCompleteCycle for 4 but exactly 3 Location, exactly 4 PathSegment, 15 steps


// ASSERTS

assert activeAndCompletedDisjoint {
  always (no t: Trip | t in ActiveTrip and t in CompletedTrip)
}

assert oneActiveTripPerUser {
  always all u: User |
    lone u.current_trip
    and
    (some u.current_trip implies
      (u.current_trip in ActiveTrip and
       u.has_selected = bike_path[u.current_trip]))
}

assert tripCompletionAtDestination {
  always all u: User |
    (some u.current_trip and u.current_trip in ActiveTrip and
     has_finished[u.current_trip] = True) implies u.currentLocation = u.trip_destination
}

assert completedTripsNeverLost {
  always all u: LoggedInUser, t: Trip | (t in u.completed_trips) implies
    (always t in u.completed_trips)
}

assert pathConnectivity {
  all p: Path |
    let firstSeg = {s: p.segments | s.segmentLocation = p.origin and
                     no s2: p.segments | s2.nextSegment = s} |
    let lastSeg = {s: p.segments | s.segmentLocation = p.destination and
                    no s.nextSegment} |
    some firstSeg and some lastSeg and
    lastSeg in firstSeg.*nextSegment
}

check activeAndCompletedDisjoint for 4 but 16 steps
check oneActiveTripPerUser for 4 but 16 steps
check tripCompletionAtDestination for 4 but 16 steps
check completedTripsNeverLost for 4 but 16 steps
check pathConnectivity for 4 but 16 steps
```

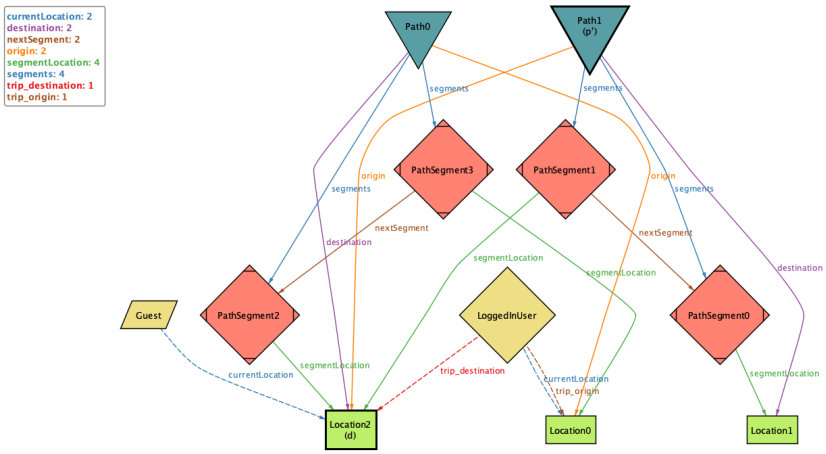
To illustrate the dynamic behavior of the Alloy specification, we present a complete execution trace generated by the loggedCompleteCycle predicate. The scenario shows how a LoggedInUser interacts with the system by selecting a route, initiating a trip, progressing through its segments, completing it, and finally repeating the process for a second trip until the goal of two completed trips is satisfied.

The initial state corresponds to the configuration defined in the init fact: one Guest, one LoggedInUser, and two predefined Path instances. The LoggedInUser begins in Location0 with no selected route and no active or completed trip.

In the first steps, the user selects the origin and destination of the trip. The origin is automatically assigned to the user's current location, while the destination must be distinct from it.



(a) Initial state



(b) Origin and destination selection

Figure 4.1: Initialization and beginning of the route selection process

After selecting the origin and destination, the system suggests one of the available paths that satisfy the specified endpoints.

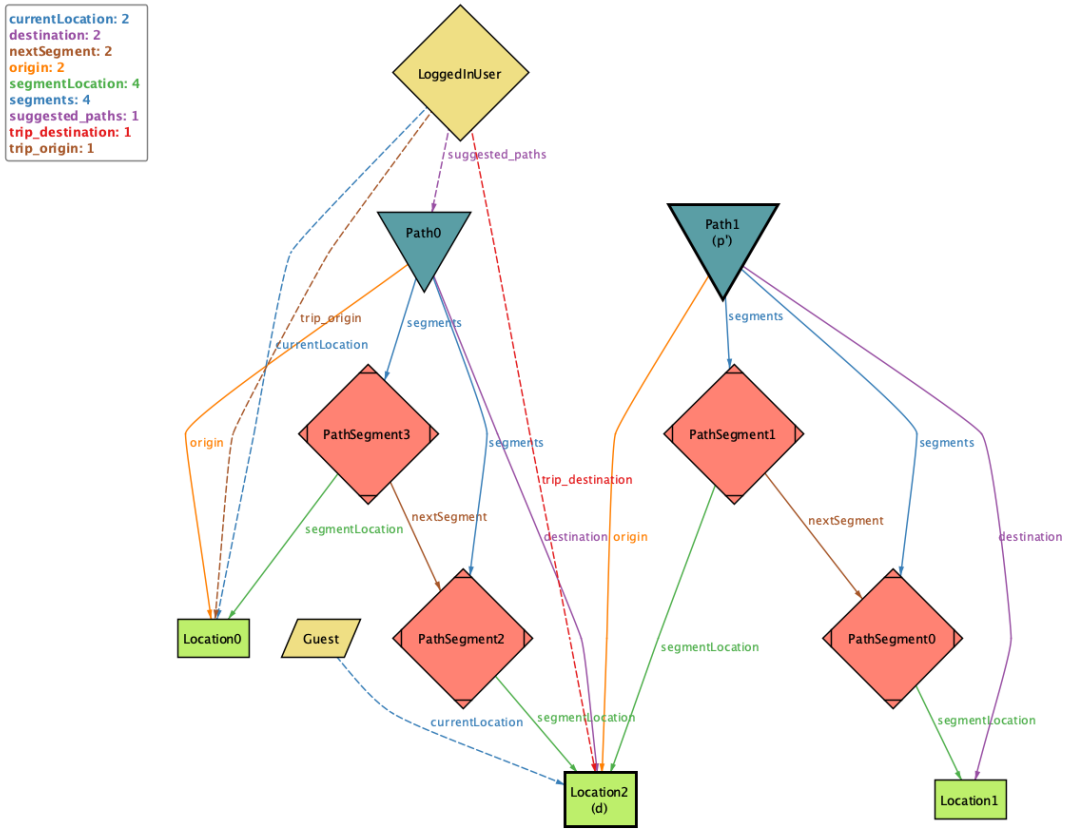
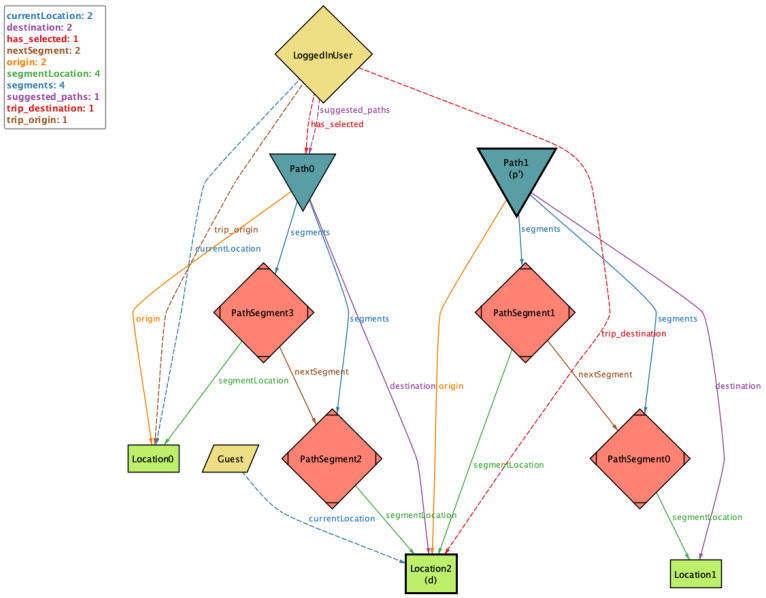
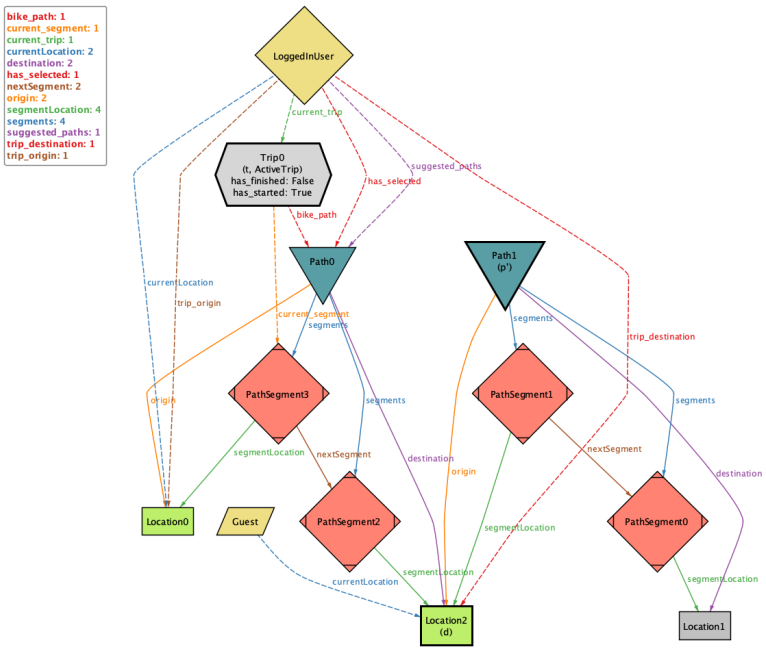


Figure 4.2: System suggesting available paths

The LoggedInUser chooses the suggested path and initiates the trip. Starting the trip creates a new ActiveTrip instance, sets its initial segment, and marks has_started = True.



(a) User selects a suggested path



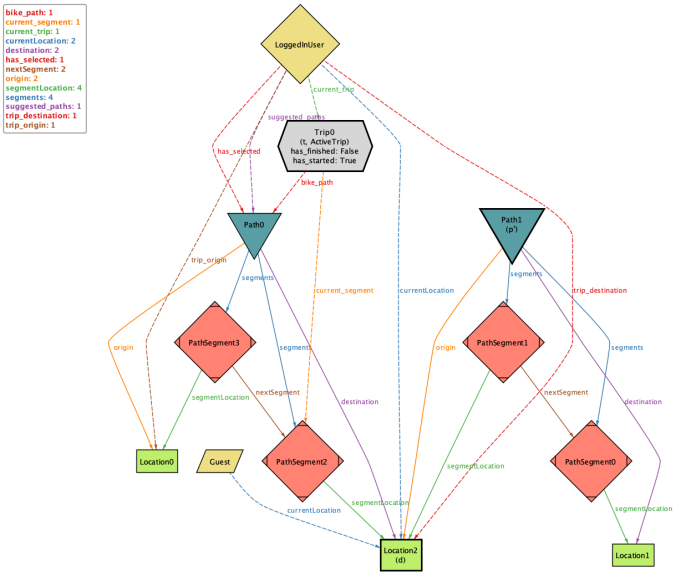
(b) User starting the trip

Figure 4.3: User selection and trip initialization

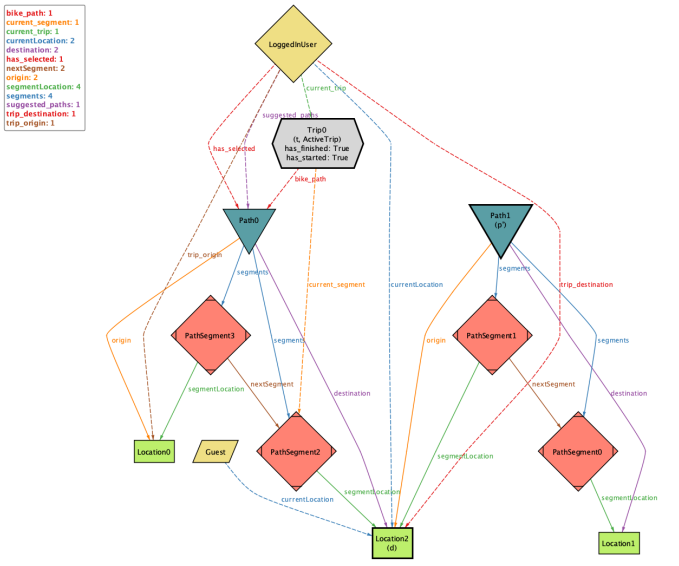
As shown in Figure 4.4a, the LoggedInUser changes position from `Location0` to `Location2`, moving from PathSegment3 to PathSegment2, which is the next segment after PathSegment3. As can be seen in the figure, the current_segment has also changed accordingly.

As the trip progresses, the user moves from one segment to the next.

Once the last segment is traversed and the destination is reached, the trip is marked as finished (has_finished = True).



(a) User moving to the next segment



(b) User arrives at destination

Figure 4.4: Completion of the first trip

Then there is another extra step to show the transition of the trip from ActiveTrip to CompletedTrip, as depicted in Figure 4.5. This state is important to show that when the trip is finalized, it is no longer active and is recorded in the user's completed_trips relation, that is his history of trips.

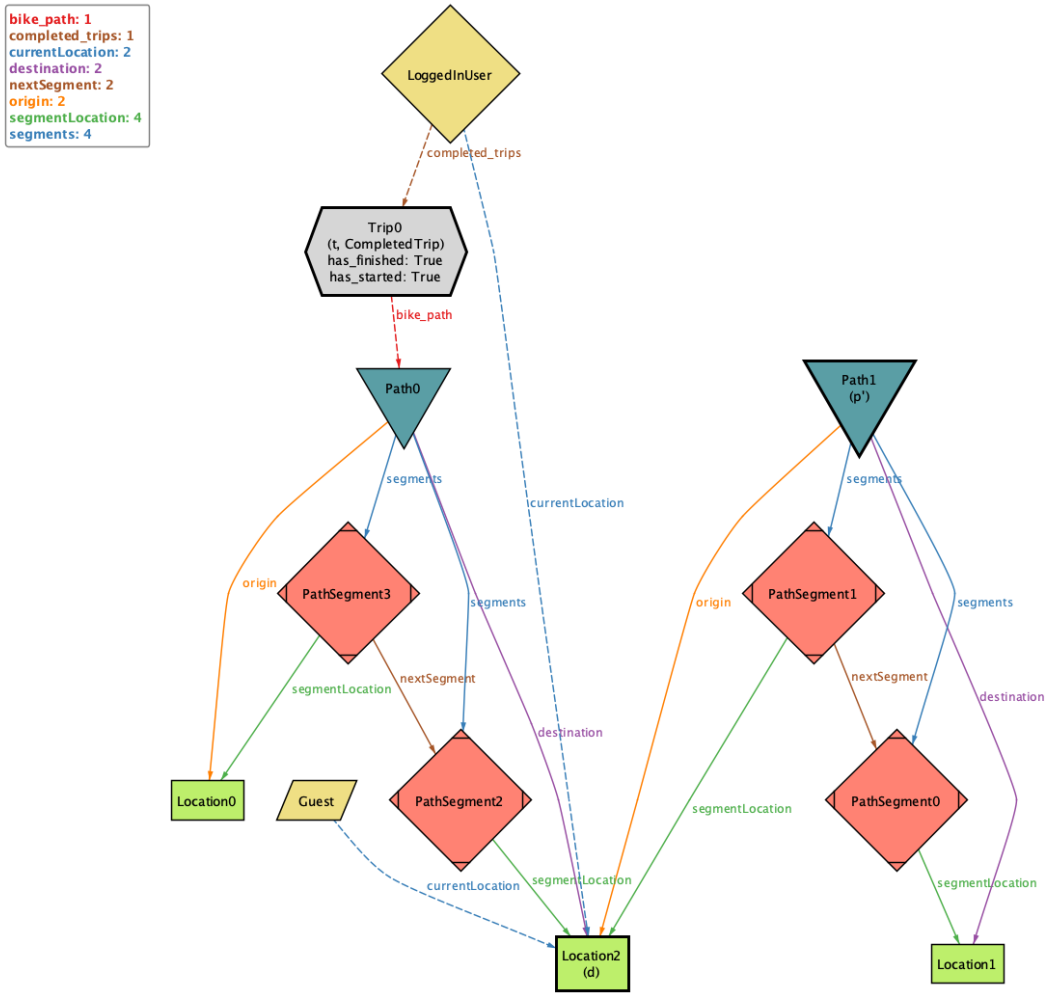
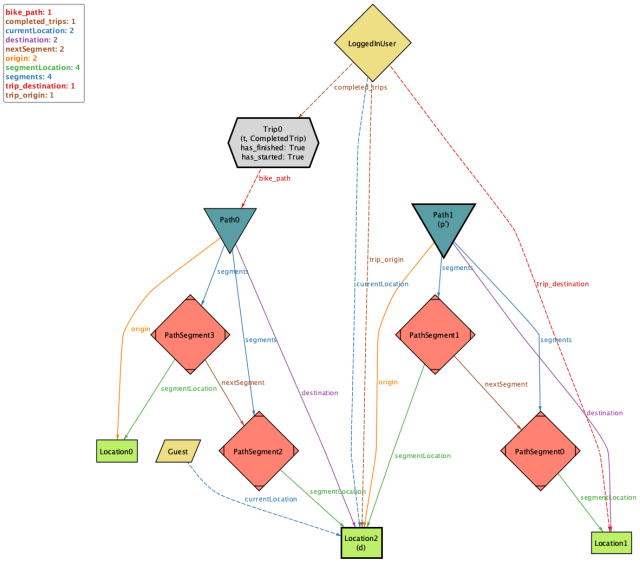
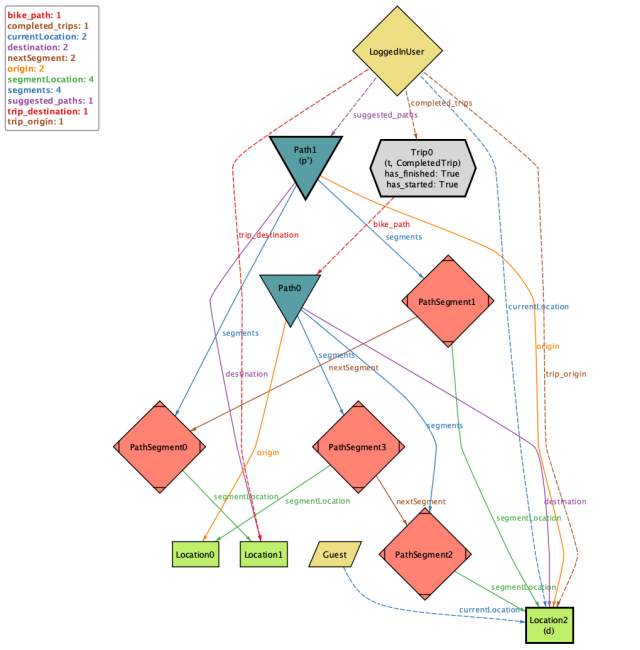


Figure 4.5: Transition from ActiveTrip to CompletedTrip

The model then restarts the cycle: the user selects a new origin and destination, a new path is suggested and all the steps are repeated (they're not shown because they are similar to the previous ones) until the user has exactly two completed trips as shown in figure 4.7.



(a) Second origin/destination selection



(b) Second path suggestion

Figure 4.6: Beginning of the second trip cycle

The second trip concludes with a new entry in the user's completed_trips relation.

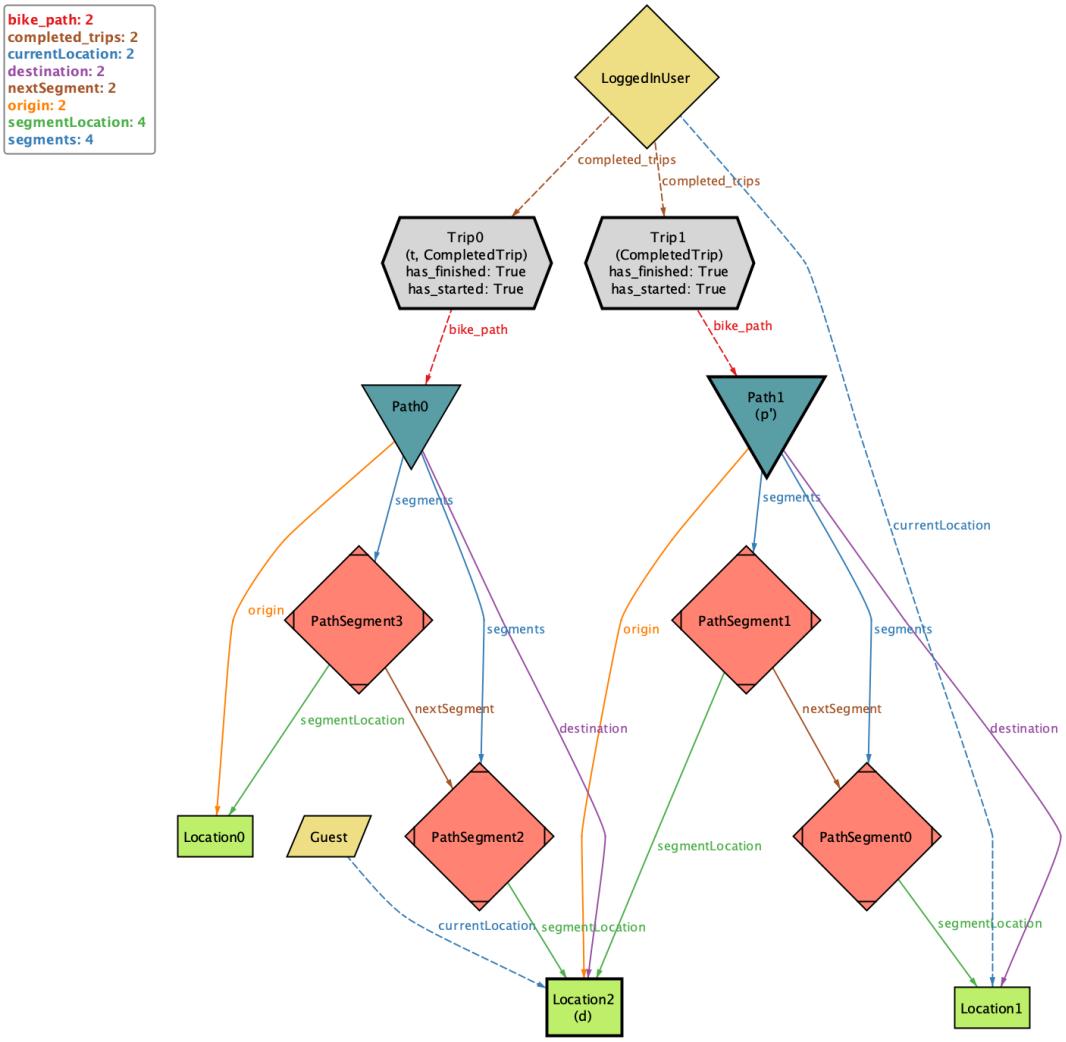


Figure 4.7: Finalization of the second trip

Once the user has exactly two completed trips, the goal of the loggedCompleteCycle predicate is satisfied. At this point, Alloy stops generating additional transitions, resulting in a trace of 15 steps (0–14).

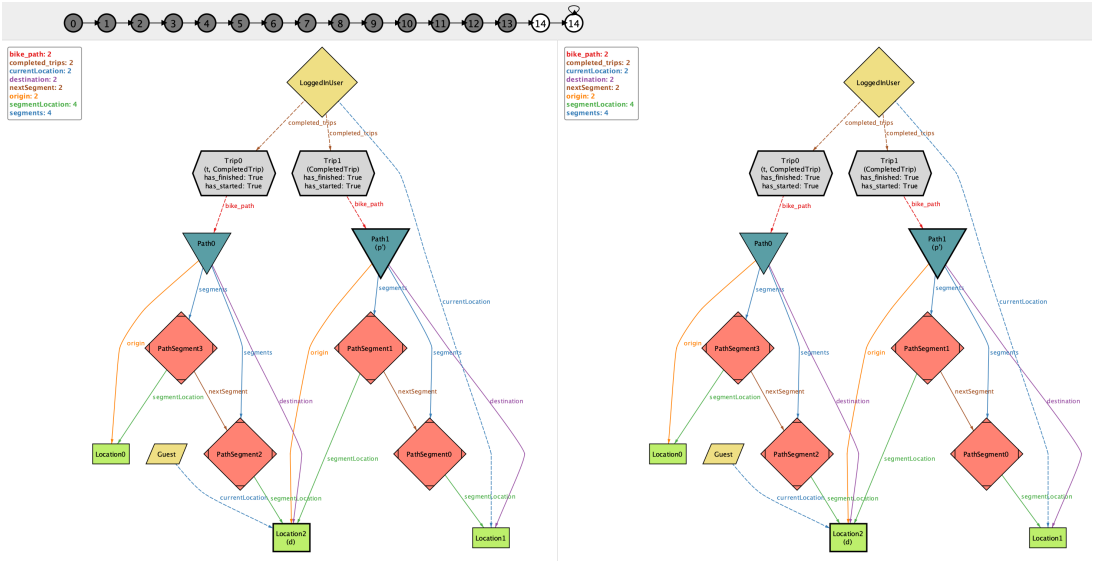
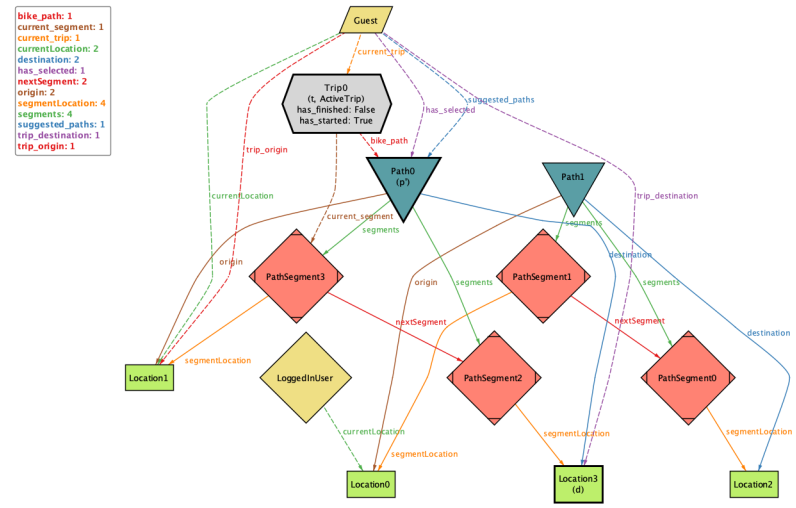


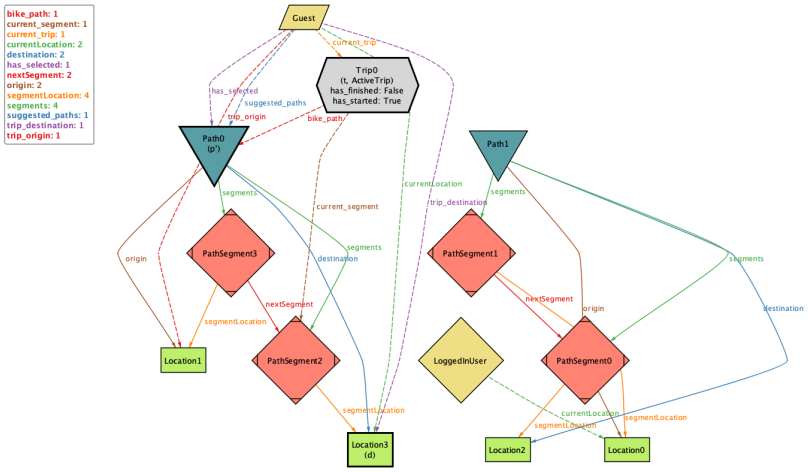
Figure 4.8: Total number of steps produced by the model

Regarding the Guest user, all the steps preceding trip finalization are analogous to those already shown for the LoggedInUser. The only semantic difference emerges at the end of the cycle: as illustrated in Figure 4.10b, the Guest user does not retain any record of completed trips, since Guest instances do not own the `completed_trips` relation.

For brevity, we do not repeat the screenshots related to origin and destination selection, path suggestion, and path selection for the Guest, as they are identical to those presented for the LoggedInUser. Instead, we focus on the part of the execution that starts when the Guest initiates a trip and ends when that trip is finalized.



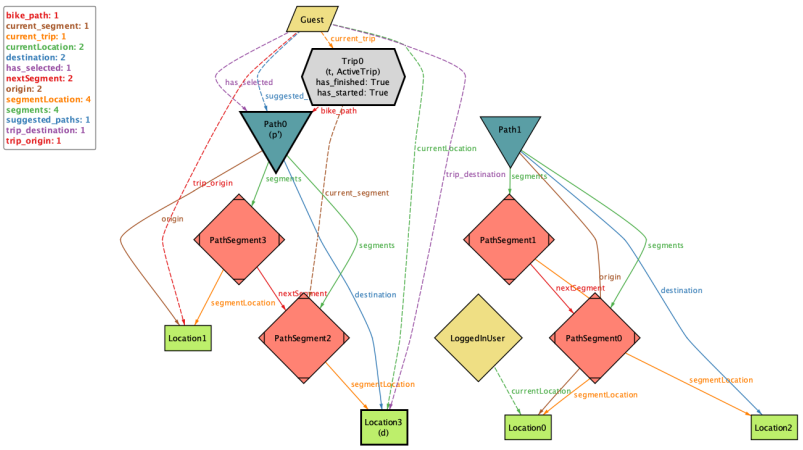
(a) User starting the trip



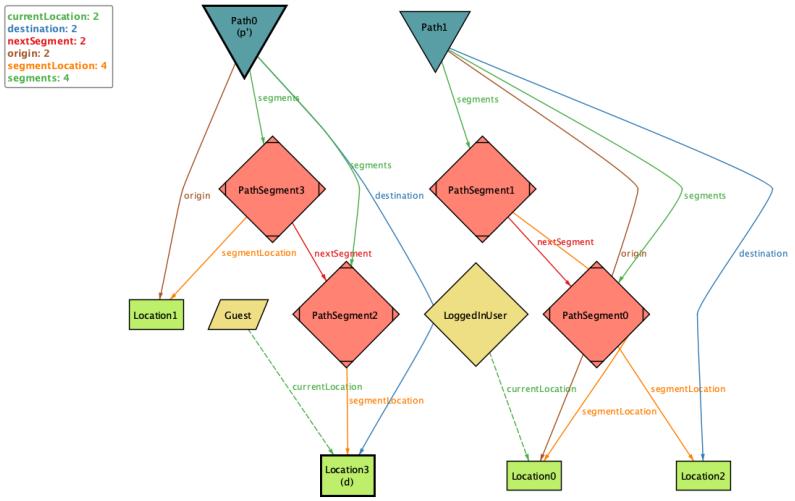
(b) User moving to the next segment

Figure 4.9: Guest user starting the trip and moving along the path

Figure 4.10a shows how the user position and the current segment evolve during the trip: the Guest moves from `Location1` to `Location3` and from `PathSegment3` to `PathSegment2`.



(a) User arrives at destination



(b) State after trip finalization

Figure 4.10: Completion and finalization of the Guest user's trip

In the final part of the trace (Figure 4.11), the Guest user does not start a new trip. At this point, the predicate used for this scenario is already satisfied, and the remaining steps are realized through stuttering transitions (represented by the `do_something_else` predicate), so no further path is selected. Moreover, in this particular instance there is no additional path available that would allow the Guest to start another trip.

To obtain an execution in which the Guest repeatedly selects and completes trips in a loop, it would be necessary to define a dedicated predicate that explicitly enforces such repeated behavior.

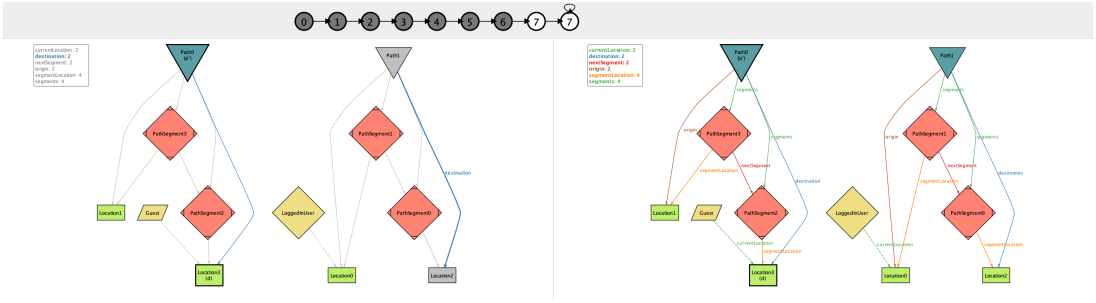


Figure 4.11: Total number of steps produced by the Guest scenario

All assertions defined in the model were checked using scopes consistent with the scenario. No counterexamples were generated (figure 4.12), confirming the internal consistency of the specification and the correctness of key properties such as:

- disjointness of ActiveTrip and CompletedTrip
- uniqueness of active trips per user
- completion only at destination
- persistence of completed trips for LoggedInUser
- path connectivity

Figure 4.12: Verification of assertions

5 | 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	Total Hours
Introduction	4 hours	4 hours	5 hours	13 hours
Overall Description	11 hours	7 hours	10 hours	28 hours
Specific Requirements	19 hours	8 hours	12 hours	39 hours
Formal Analysis	7 hours	21 hours	11 hours	39 hours
Final Review & Editing	3 hours	3 hours	3 hours	9 hours
Total Hours	**44 hours**	**43 hours**	**41 hours**	**128 hours**

Table 5.1: Time spent on document preparation

Bibliography

[1] ISO/IEC/IEEE. Systems and software engineering — life cycle processes — requirements engineering, 2018.

[2] M. Jackson and P. Zave. Deriving specifications from requirements: An example. In *Proceedings of the 17th International Conference on Software Engineering (ICSE'95)*, Seattle, WA, USA, 1995. ACM. ISBN 0-89791-708-1. doi: 10.1109/ICSE.1995.499.

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

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

List of Figures

2.1 Domain Class Diagram 13
2.2 Starting Trip State Diagram 16
2.3 Manual Path Creation State Diagram 17
2.4 Automatic Path Creation State Diagram 18
2.5 Manual Report State Diagram 19
2.6 Automatic Report State Diagram 20

3.1 Authentication and Profile Management Use Case Diagram 31
3.2 Guest User Use Case Diagram 32
3.3 Logged-in User Use Case Diagram 33
3.4 User Registration Sequence Diagram 35
3.5 User Log In Sequence Diagram 37
3.6 User Log Out Sequence Diagram 38
3.7 Search for a Path Sequence Diagram 40
3.8 Select a Path Sequence Diagram 41
3.9 Create a Path in Manual Mode Sequence Diagram 43
3.10 Create a Path in Automatic Mode Sequence Diagram 45
3.11 Delete a Path Sequence Diagram 47
3.12 Start a Trip as Guest User Sequence Diagram 48
3.13 Start a Trip in Manual Mode as a Logged-in User Sequence Diagram . . . 50
3.14 Start a Trip in Automatic Mode as a Logged-in User Sequence Diagram . . 52
3.15 Stop a Trip as a Guest User Sequence Diagram 53
3.16 Stop a Trip as a Logged-in User Sequence Diagram 55
3.17 Make a Report in Manual Mode Sequence Diagram 57
3.18 Make a Report in Automatic Mode Sequence Diagram 59
3.19 Confirm a Report Sequence Diagram 61
3.20 Manage Path Visibility Sequence Diagram 63
3.21 View Trip History and Trip Details Sequence Diagram 65
3.22 View Overall Statistics Sequence Diagram 67
3.23 View Trip Statistics Sequence Diagram 69

3.24 Edit Personal Profile Sequence Diagram 71

4.1 Initialization and beginning of the route selection process 85
4.2 System suggesting available paths 86
4.3 User selection and trip initialization 87
4.4 Completion of the first trip 88
4.5 Transition from ActiveTrip to CompletedTrip 89
4.6 Beginning of the second trip cycle 90
4.7 Finalization of the second trip 91
4.8 Total number of steps produced by the model 92
4.9 Guest user starting the trip and moving along the path 93
4.10 Completion and finalization of the Guest user's trip 94
4.11 Total number of steps produced by the Guest scenario 95
4.12 Verification of assertions 96

List of Tables

3.1 User Registration Process Detail 34
3.2 User Log In Process Detail 36
3.3 User Log Out Process Detail 38
3.4 Search for a Path Process Detail 39
3.5 Select a Path Process Detail 41
3.6 Create a Path in Manual Mode Process Detail 42
3.7 Create a Path in Automatic Mode Process Detail 44
3.8 Delete a Path Process Detail 46
3.9 Start a Trip as Guest User Process Detail 48
3.10 Start a Trip in Manual Mode as a Logged-in User Process Detail 49
3.11 Start a Trip in Automatic Mode as a Logged-in User Process Detail 51
3.12 Stop a Trip as a Guest User Process Detail 53
3.13 Stop a Trip as a Logged-in User Process Detail 54
3.14 Make a Report in Manual Mode Process Detail 56
3.15 Make a Report in Automatic Mode Process Detail 58
3.16 Confirm a Report Process Detail 60
3.17 Manage Path Visibility Process Detail 62
3.18 View Trip History Process Detail 64
3.19 View Overall Statistics Process Detail 66
3.20 View Trip Statistics Process Detail 68
3.21 Edit Personal Profile Process Detail 70
3.22 Mapping between goals, requirements, and domain assumptions 72

5.1 Time spent on document preparation 97