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: 2.0
Date: 21.12.2025
Project Link: Errigo-Gholami-Ianosel (github.com)

Contents

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 possible path closures.

[G4] **Logged-in users can track their cycling activities over time**, monitoring their trips and reviewing their past performances.

[G5] **Logged-in users can contribute to the community knowledge** by creating new bike paths and by submitting or confirming reports about existing ones.

[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 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 total distance, duration, average speed, and other performance metrics. When available, **additional information** such as weather conditions, temperature, and wind speed can be **retrieved** from **external meteorological services** and included in the trip summary.

BBP can **detect when a user is biking** based on their speed and GPS movement. If the app is open and such activity is detected, the system may prompt the user to start recording a trip.

While following a path, the user can visualize their current position on the map. Additionally, the system may provide **pop-up alerts** about nearby obstacles or particular conditions along the path, that the user can confirm or dismiss.

BBP supports both manual reports and automatically detected anomaly reports during an active trip. **manual reporting** is always available: during a ride, the user may submit a report describing obstacles or issues, and BBP automatically retrieves the user's GPS position and associates it with the report. If the user enables **automatic mode** when starting the trip, BBP additionally analyzes sensor data (e.g., accelerometer, gyroscope) to detect irregular movements that may indicate potholes or other issues. Since this process may generate false positives, each detected anomaly is immediately shown to the user, who may confirm, edit, or reject it. Only validated anomalies become publishable reports. Activating automatic mode does not disable manual reporting, both mechanisms

remain available throughout the trip.

BBP also supports both manual and automatic creation of new bike paths. In **manual mode**, the user draws the desired path directly on the map or selects the map segments composing it. In **automatic mode**, the system reconstructs the path by acquiring GPS data while the user cycles along it. Regardless of the creation method, the new path may remain private or be marked as publishable for the community.

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 recent 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.

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 used by the system, such as meteorological services and map visualization providers.

[W2] The user has a device capable of measuring 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. World Controlled Shared Phenomena

[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 user starts or stops recording a bike trip.

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

[SP14] External sensors transmit their data to the system.

[SP15] External meteorological services provide weather conditions.

1.2.3. Machine Controlled Shared Phenomena

[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 provides pop-up alerts 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 external 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

- **Path:** A route defined either by data collected from users or by paths manually created within BBP. A Bike Path consists of one or more Path Segments.
- **Path Segment:** A portion of a 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*, *Requires Maintenance*, and *Closed*.
- **Path Score:** The final ranking value computed by the system, based on the path's aggregated status and effectiveness, used to recommend the best alternatives between an origin and a destination.
- **Trip:** A cycling activity tracked through the BBP application. If the user is logged-in, the trip is stored and becomes part of the user's trip history. 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 or Automatically Detected (sensor-detected).
- **Freshness:** A metric used when merging reports; newer reports carry more weight than older ones when determining Path Status.
- **Obstacles:** Elements on a path that negatively impact cycling conditions, such as potholes or flooding, as identified by users or automatic sensors.
- **Manual Creation Mode:** The creation mode in which a logged-in user defines a new path by drawing its geometry on the map or selecting the map segments

composing it.

- **Automatic Creation Mode:** The creation mode in which a logged-in user defines a new path by cycling along it, allowing the system to reconstruct the path using GPS data.
- **Manual Report:** The system functionality where the logged-in user manually makes a report, inserting the bike path's condition and obstacle, through the application interface.
- **Automatic Report:** The system functionality where, during an active trip with Automatic Mode enabled, the system analyzes sensor data to detect anomalies (e.g., potholes) and presents them to the user for confirmation before generating a report.

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 (30 November 2025);
- Version 2.0 (21 December 2025);

1.5. 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 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. **References**: lists all sources (including Software) and documents used in the preparation of this document.
6. **Effort Spent**: provides a breakdown of time spent on each section of the 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 can't 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 logged-in user starts to browse and create paths**:

   Irene accesses the BBP app to plan her next cycling trip. From the main page, the system automatically sets her current position as the starting point, while she

manually specifies her desired destination. BBP then displays on a map all available bike paths connecting these 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 the option to create a new path by first providing a title and a short description, and by choosing whether the path should remain private or be publishable for the community. The system also asks her to select the creation mode: manual or automatic. Irene chooses the manual mode. In this mode, she can define the path by drawing it directly on the map or by selecting the map segments that compose her desired path. Once satisfied with the resulting path, she confirms the creation 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 has chosen to make it publishable, 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 one of the suggested bike paths, he presses the start button, available only when the GPS position matches the path origin. At this point, BBP asks him whether he wants to enable the Automatic Mode. Cesare confirms by selecting Yes, and the trip begins. 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 trip history.

4. **A logged-in user browses 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 corresponding section, where BBP displays a list of his recorded trips over time. Each trip is accompanied by summary information such

as the date, distance traveled, duration, and destination. Luca can sort trips by date or distance and scrolling through the list, he recognizes some of his favorite routes. 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 accesses his trip and per trip statistics**:

   Pietro selects his latest trip from his trip list. BBP opens the path details page, displaying the entire path 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 external weather service, 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, and he had confirmed the presence of obstacles during the ride. 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. During her ride, she notices that a section of the path 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, Giulia selects the report button. Then, she selects the obstacle type, and chooses "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 path.

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

   While cycling through a familiar path, Marco receives a pop-up alert informing him 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 pop-up, he presses the confirmation button, and BBP registers his feedback as an additional confirmation for that path condition. Later, while continuing his ride, Marco receives another pop-up alert about a reported bump on the same road, but this time the road surface looks perfectly fine. He therefore indicates that the problem has been resolved or no longer exists, by clicking the corresponding button. 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.

8. **A logged-in user unintentionally leaves the planned path**:

   Stefania is cycling while following one of the suggested routes previously selected in BBP. During her trip, she gets distracted and forgets to turn right at a crucial intersection, gradually moving away from the planned path without realizing it. BBP continuously compares her real-time GPS position with the expected path. When the system detects that Stefania has deviated significantly from the planned path, it automatically stops the current trip. A pop-up appears on her screen informing her that the trip has been stopped due to a deviation from the selected path and that the recorded data has been safely saved. Stefania slows down and decides to resume her ride. Since her current position is now different from the original path, she opens the BBP app and starts a new trip: the system automatically sets her current position as the starting point, and she manually enters the destination again. BBP displays all available bike paths that connect her new origin to the chosen destination, ordering them according to their overall score and current condition. After reviewing the suggested options, Stefania selects the path that best fits her needs and starts her new trip from the updated location.

9. **A logged-in user forgets to start a trip**:

   Eugenia is cycling on a familiar road while the BBP app is open on her phone. She intends to track her trip to later review her statistics, but she gets distracted and forgets to press the start button before beginning her ride. As she continues pedaling, BBP analyzes her speed and GPS movement. After detecting that Eugenia is likely biking, the system shows a pop-up alert, asking if she wants to start recording her trip. Eugenia slows down for a moment and decides she indeed wants her cycling session to be tracked. She confirms the prompt, and BBP asks her to enter her desired destination. The system automatically sets her current GPS position as the origin. After entering the destination, BBP displays all available bike paths connecting her current location to the selected destination, sorted by their overall score and current condition. Eugenia selects the option that suits her best and presses the start button. She then chooses whether to activate the automatic report mode. The trip officially begins, and BBP starts recording GPS and sensor data as she resumes cycling. At the end of her ride, Eugenia will find the complete trip summary in her trip history, including distance, duration, speed, and weather information retrieved from the external meteorological service.

2.1.2. Domain Class Diagram

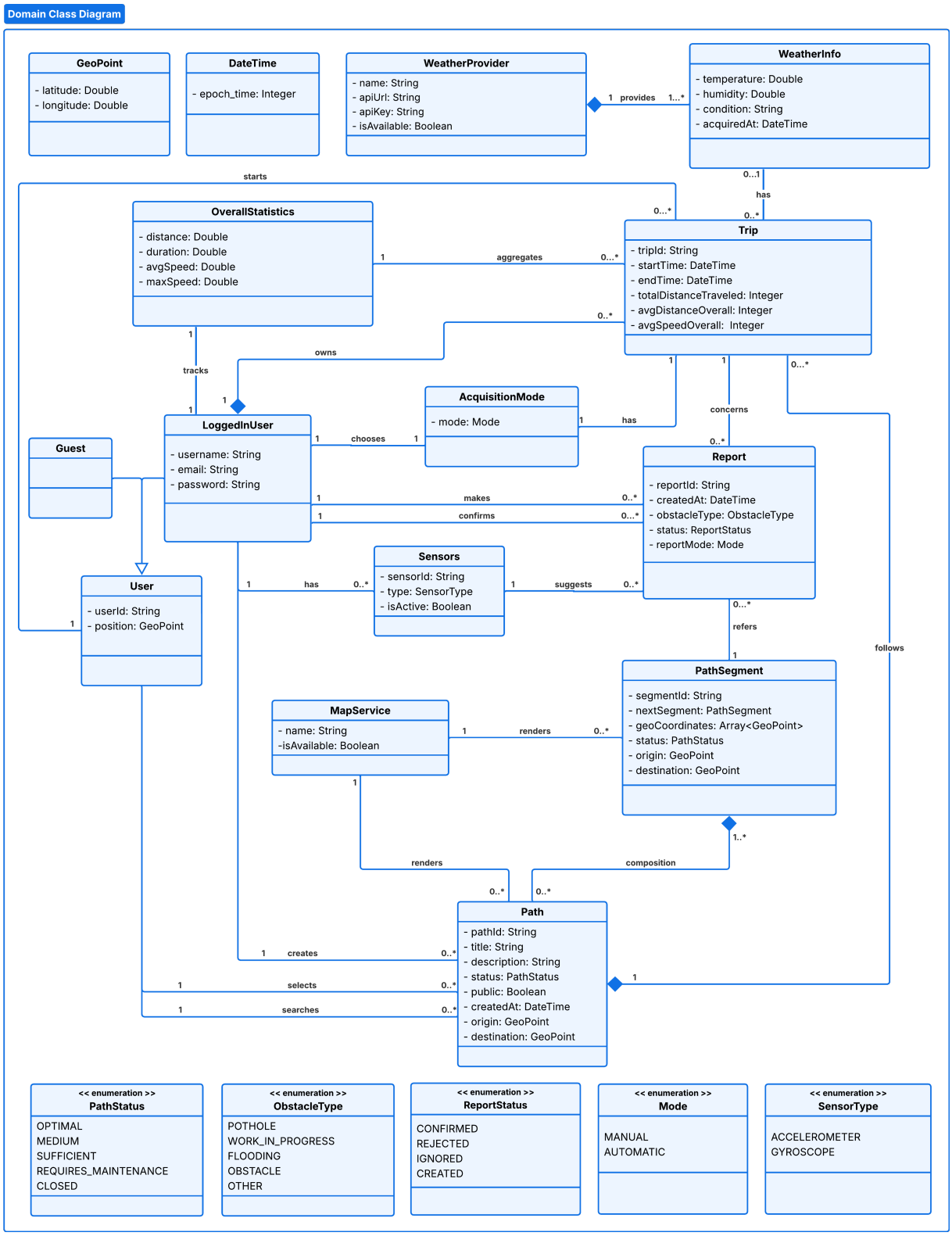

Figure 2.1: Domain Class Diagram

The Domain Class Diagram for the Best Bike Paths (BBP) system establishes a comprehensive framework for managing cycling infrastructure and user activity. At the foundation of the system is the User entity, which tracks a unique identifier and a current position via a GeoPoint. This entity branches into two distinct roles: the Guest and the LoggedInUser.
A Guest user has the flexibility to search for existing paths and initiate trips, while also having the option to register for a permanent account. Once registered, the LoggedInUser is identified by a username, email, and password, allowing them to own specific trip records and maintain a long-term profile within the system.
The structural core of the application is the Path, which includes its description, public visibility, and a status enumeration that tracks the path's condition. A Path is built through a composition of one or more PathSegment entities. These segments link the origin and destination GeoPoints; each segment contains an array of specific coordinates and maintains a recursive reference to the next segment in the sequence, creating a traversable chain. These geographical structures are visualized through a MapService.
Activity within the system is captured through the Trip entity, which records the temporal and physical details of a specific journey. A Trip stores attributes such as start and end times, total distance, and average speed.
It is important to note that while the Trip contains its own specific performance data, it is also aggregated into the OverallStatistics entity owned by a LoggedInUser. This distinction ensures that individual trip metrics are preserved while contributing to the user's cumulative performance history, such as their lifetime average speed and total distance.
Furthermore, trips may be contextualized by WeatherInfo provided by a WeatherProvider which logs environmental factors like temperature and humidity at the time of the ride.
Crowd-sourced data and safety are managed through a robust reporting system. LoggedInUsers can generate Report entities to identify various obstacles, such as potholes or flooding, categorized by an ObstacleType. These reports can be created manually or triggered automatically via an AcquisitionMode. To facilitate automatic detection, the system integrates with hardware Sensors, including accelerometers and gyroscopes, which suggest potential reports based on physical movement. Each report undergoes a validation process reflected in its status, moving from creation to confirmation or rejection. All temporal data throughout these processes is standardized using the DateTime entity as an epoch-time integer, ensuring consistency across all system logs.

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 browsing and selecting a path, creating paths, and reporting obstacles.

Selecting a Path

The following diagram illustrates the possible states involved when a user performs a search, browses through results and selects a path. This process begins when the user opens the app and decides to perform a new search 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.

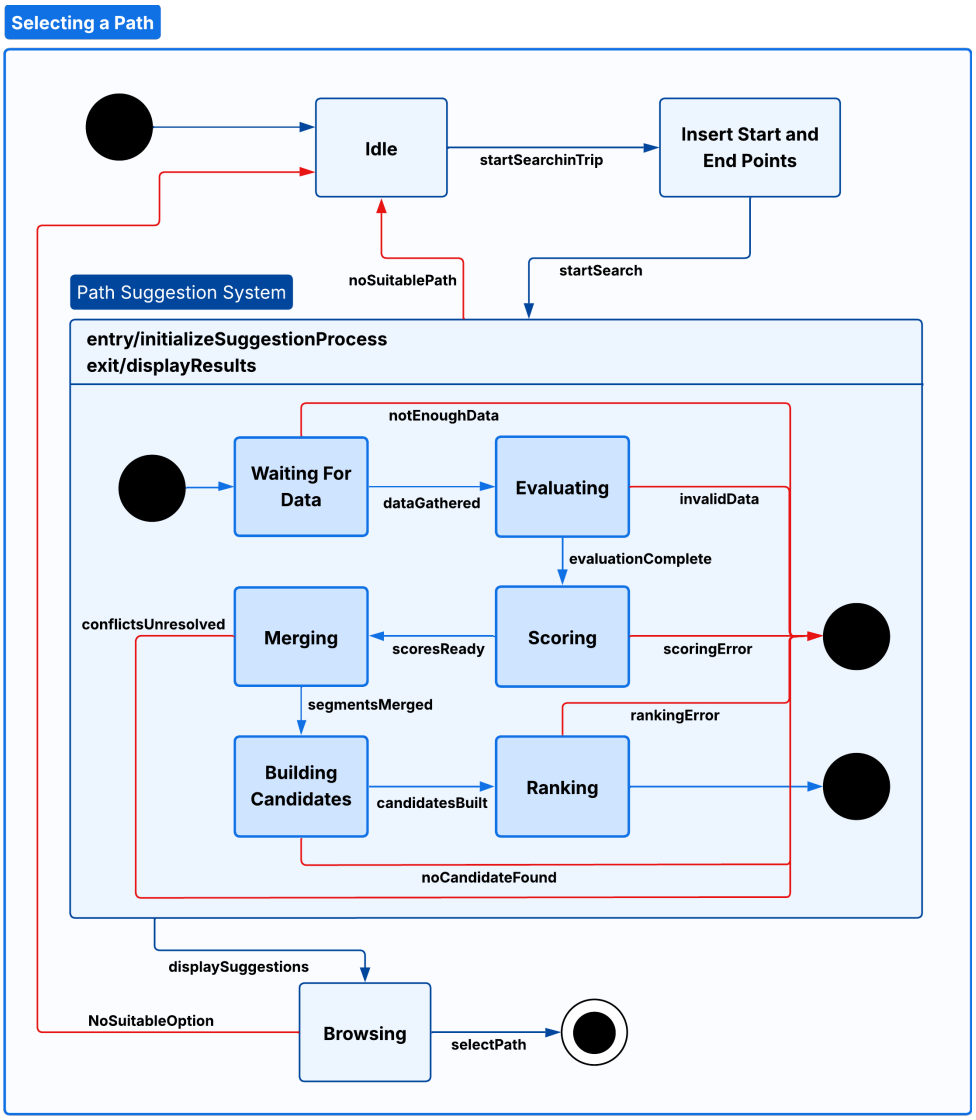

Figure 2.2: Selecting a Path State Diagram

Manual Path Creation

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 pressing the create path button. 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 creation mode, where he should select manual creation.

After the metadata are confirmed and saved, the process continues in the Adding Segments state. In this phase, the user manually draws segments on the map to define the complete path. 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 path. 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.

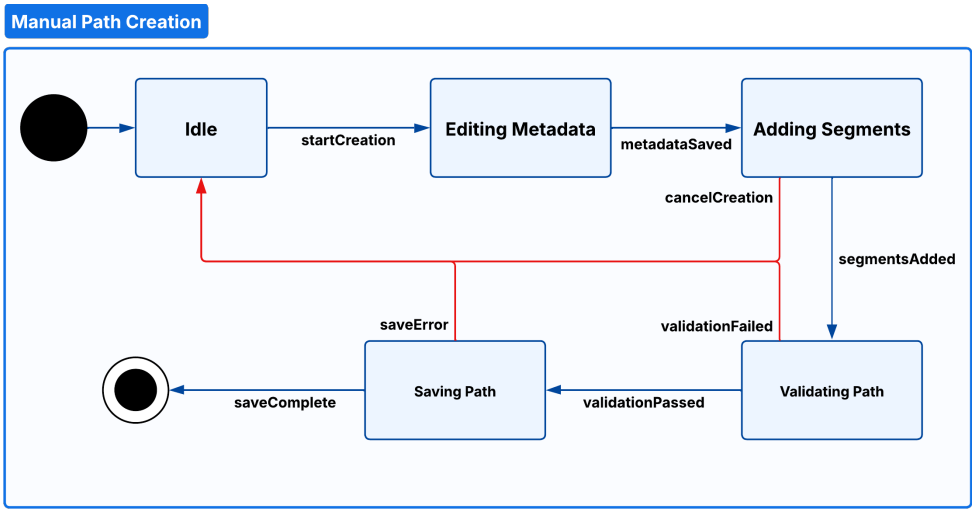

Figure 2.3: Manual Path Creation State Diagram

Automatic Path Creation

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 create path button. 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, and where he should select the automatic mode for path creation. 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 path 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.

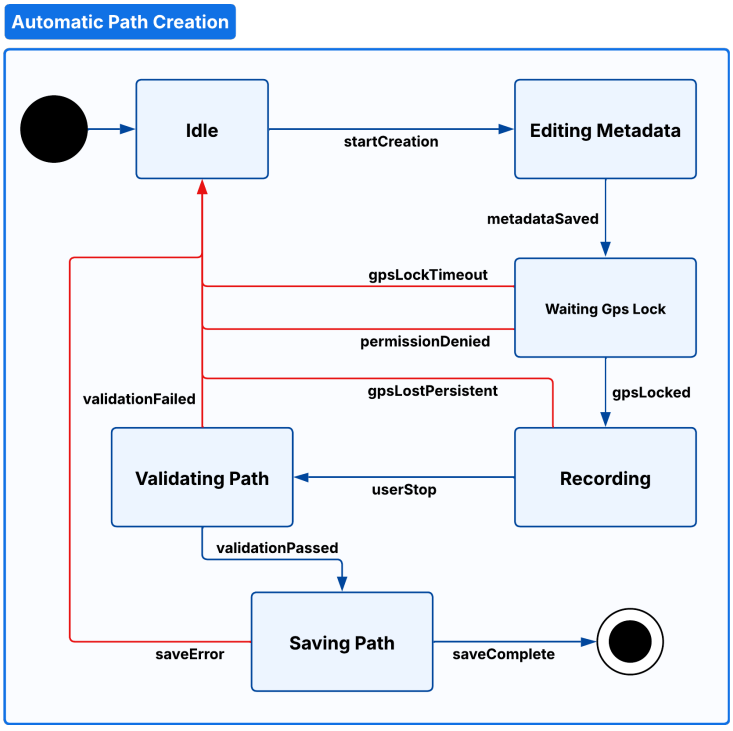

Figure 2.4: Automatic Path Creation State Diagram

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 path status 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.

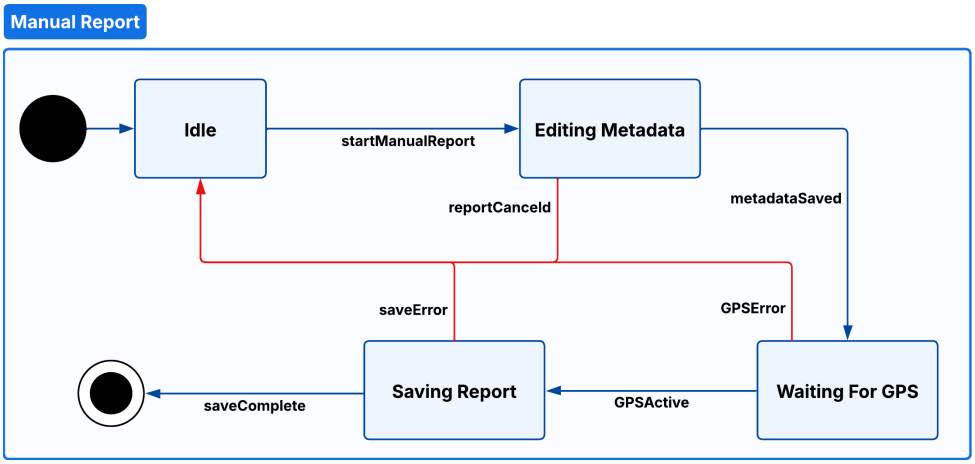

Figure 2.5: Manual Report State Diagram

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 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.

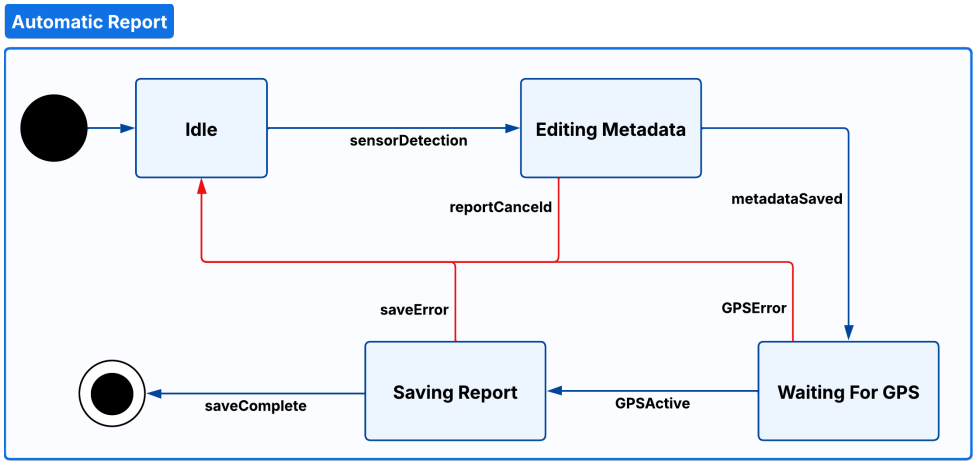

Figure 2.6: Automatic Report State Diagram

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 trip 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 path suggestions based on community rankings, but they can't store recorded trips, create new paths, or contribute by making 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, displayed as path status (e.g., optimal, medium, requires maintenance, closed). Users can explore the visualized paths, view their details, select the most suitable path for their needs and start a trip. However, only logged-in users can record a trip or contribute with 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 draws the path directly on the map or selects the segments they wish to include. This allows cyclists to define personalized paths that may not yet exist in the system or that better fit their preferences, without manually entering street names. In automatic mode, the system constructs the path by recording GPS data as the user physically cycles along it. While the user rides, BBP collects positional information and reconstructs the path in real time. Regardless of the chosen method, the newly created path may remain private or be marked as publishable for the entire BBP community.

- **Bike Path Visibility Management**:

  Logged-in users can modify the visibility of the bike paths they have created. Each user-created path can be either kept *private*, accessible only to its creator, or marked as *public*, making it available to the entire BBP community. Users can change this setting at any time from the path details page. When the visibility is updated, BBP reflects the new status immediately in future searches and suggestions. This functionality ensures user control over which paths are shared with the community.

- **Bike Path Deletion**:

  Logged-in users can delete any bike path they have personally created. From the path details page, the user may select the delete option, after which BBP removes the path and all its associated metadata from the system. Deleted paths are no longer available for future searches or for the community.

- **Trip Recording and Data Acquisition**:

  A logged-in user can start a trip by selecting a suggested bike path and reaching its origin, which must match the user's current GPS position. Before starting the trip, BBP asks whether the user wants to enable the *Automatic Mode*. GPS tracking is always active, while the use of accelerometer and gyroscope is enabled only when Automatic Mode is selected. In automatic mode, BBP uses the external sensors to identify potential irregularities in the road surface that may indicate obstacles

or potholes. Each detected event triggers a pop-up asking the user to confirm, edit, or reject the anomaly before it is stored as a report. When automatic mode is disabled, reporting is possible only through manual user input. During the trip, BBP continuously collects GPS data to track the user's path and provides navigation to the destination. The trip stops automatically upon arrival or manually by clicking the stop button, then the system processes the data and generates a summary which is stored in the user's trip history.

- **Automatic Ride Detection and Trip Start Suggestion**:

  BBP continuously analyzes the user's movement whenever the app is open. If the system detects, based on GPS speed and movement patterns, that the user is likely biking without having started a trip, it displays a pop-up suggesting to begin recording the activity. The pop-up prompts the user to start a new trip. If the user agrees, they should insert the destination, while the system automatically sets the current GPS location as the origin. Then, BBP displays the available suggested paths from the current position to the chosen destination. Once the user selects one, and chooses whether to enable the automatic mode, the trip starts normally, with full GPS tracking and (if enabled) sensor-based anomaly detection. If the user dismisses the pop-up, BBP returns to idle state without recording any data. This functionality ensures that users do not miss the opportunity to record their trip, maintain complete statistics, and benefit from trip tracking even when they forget to manually start a trip.

- **Automatic Trip Interruption on Path Deviation**:

  BBP continuously compares the user's real-time GPS position with the selected path. If the system detects a significant deviation from the planned route, it automatically stops the trip and displays a pop-up informing the user of the interruption. If the user is logged-in, the recorded data up to that point are safely stored in their trip history. The user can then start a new trip from the updated position by selecting a new destination. This mechanism ensures accurate tracking and prevents invalid trip recordings when the user leaves the chosen path.

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

  BBP allows logged-in users to manually report the condition of a bike path only while they are on an active trip. This feature helps to keep path information accurate, especially when users encounter issues such as obstacles or partial closures. By selecting the report button during the ride, the user can choose the type of issue

from a predefined list (e.g., obstacle, debris, surface damage). In addition, the user may assign an overall status to the affected path segment, such as "optimal," "medium," "requires maintenance," or "closed." The system automatically retrieves the user's current GPS position and associates the report with the corresponding path segment, without requiring the user to manually indicate the affected location.

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

  During a ride, logged-in users may receive pop-up alerts about nearby issues reported by others. As they approach the location, users can confirm that the issue persists or reject it, if the problem has been resolved or was inaccurately reported. Their feedback is instantly recorded by BBP and influences the path's overall condition status. This real-time confirmation mechanism helps maintain accurate and up-to-date information about bike path conditions, benefiting the entire BBP community.

- **Confirmation of Automatically Detected Anomalies**:

  When a logged-in user is riding in automatic mode, BBP uses the device's sensors to detect potential anomalies, such as bumps or sudden irregular movements, that may indicate potholes or obstacles. To avoid false positives, detected anomalies are not published immediately. As soon as an anomaly is detected, BBP shows a confirmation prompt allowing the user to validate, edit or reject it in real time. Only confirmed detections become publishable. Users can still submit manual reports at any time during the trip, regardless of whether the automatic mode is enabled.

- **Overall Statistics**:

  Logged-in users can access aggregate statistics summarizing the users' overall performance over time. The aggregate statistics include metrics such as number of trips, number of paths created, kilometers traveled and average speed. Users can select a specific time range (e.g., last week, last month, last year) to filter the statistics accordingly. The overall statistics are 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 access their personal cycling history, that displays a list of all recorded trips. Users can select any trip from their trip list to access detailed trip specific statistics. Users can also sort the list by date or distance to easily find specific activities. For each individual trip, BBP displays a map showing the complete path followed by the user and provides statistics such as total distance, duration and average speed. There is also information about meteorological conditions during

the ride, such as average temperature, humidity levels, and wind speed or direction. Additionally, regardless of whether the trip was recorded in automatic or manual mode, the map highlights both the points where the accelerometer detected irregular movements that the user confirmed during the ride and the points where the user submitted manual reports.

- **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.

- **User Profile Management and Account Settings**:

  Logged-in users can access a dedicated section to view and update their personal information, such as username, email address, and password. Logged-in users can also manage their account settings, including privacy options. Profile changes are validated and stored by BBP, ensuring that future trips, statistics, and community contributions are always associated with the updated account data.

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 and rankings, and consult public reports about obstacles or road conditions. Guest users are also allowed to start a trip. However, they can't enable the Automatic Mode, can't submit obstacle reports, can't confirm reports submitted by others, and do not receive pop-ups related to anomaly detection or confirmation. At the end of the trip, no data is stored in the system: guest users have no trip history and do not have access to overall or per-trip statistics. They are not allowed to create or delete any paths, nor to modify any

existing data in the system.

- **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 drawing on a map interface. They can choose whether to keep their created paths private or share them with the community. They can also delete any paths they have created. During trips, logged-in users can enable the Automatic Mode to allow BBP to detect potential obstacles using device sensors. They can also submit obstacle reports, and validate or confirm existing ones shared by other users. Furthermore, logged-in users can store their trips and view their personal trip history. They can access both overall statistics summarizing their cycling activities over time, and detailed statistics for each individual trip. Logged-in users can also manage their profile information and account settings.

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 services. Access to motion sensors is required only for sensor-based features.

[D4] When Automatic Mode is enabled, 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] GPS position updates are received with sufficient frequency (e.g., at least 1 update per second) to detect movement, deviations, and trip progress.

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

[D9] The map service returns valid, connected, and navigable path segments for any origin/destination pair inside its coverage.

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 guest users to start a cycling trip.

   [R7] The system shall allow guest users to stop a currently active trip, but shall not store any trip data after the trip ends.

   [R8] The system shall allow the user to start a trip only when their GPS position matches the path origin.

   [R9] The system shall display a pop-up suggesting to start a trip when cycling is detected while no trip is active and the app is open.

   [R10] The system shall set the current GPS position as trip origin when starting from

auto-detection.

[R11] The system shall automatically stop the active trip when the user's GPS position deviates from the selected path within a certain threshold.

[R12] The system shall allow logged-in users to start a cycling trip in manual or automatic mode.

[R13] The system shall allow logged-in users to stop a currently active trip and save the recorded data.

[R14] The system shall collect GPS data during trip recording.

[R15] The system shall collect motion sensor data (accelerometer, gyroscope) during trip recording only when Automatic Mode is enabled.

[R16] The system shall allow logged-in users to view the list of their recorded trips.

[R17] The system shall allow logged-in users to view a summary of their overall cycling statistics (total distance, total time, average speed, etc.).

[R18] The system shall allow logged-in users to view statistics for each trip (distance, speed, duration, etc.).

[R19] The system shall display the route and reported obstacles associated with a recorded trip.

[R20] The system shall allow logged-in users to delete a recorded trip.

3. **Trip Data Enrichment**

[R21] 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**

[R22] The system shall detect when a user is cycling based on speed and acceleration patterns.

[R23] The system shall detect irregular movements from sensor data that may suggest potholes or surface defects when Automatic Mode is enabled.

[R24] The system shall present automatically detected path and obstacle data to the logged-in user for manual confirmation before publishing.

5. **Path Information Management**

[R25] The system shall allow logged-in users to manually create a new bike path by drawing segments.

[R26] The system shall allow logged-in users to manually report obstacles or problems on a bike path while performing an active trip.

[R27] The system shall allow logged-in users to manually confirm or reject the presence of obstacles reported by other users.

[R28] The system shall allow logged-in users to create a new bike path in automatic mode using GPS tracking.

[R29] The system shall allow logged-in users to delete their previously created paths.

[R30] 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**

[R31] The system shall aggregate multiple user reports referring to the same path segment.

[R32] The system shall evaluate the reliability of each path segment based on the number of confirmations and report freshness.

[R33] The system shall determine the current status of a path (optimal, medium, sufficient, requires maintenance, closed).

[R34] 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**

[R35] The system shall allow any user (guest or logged-in) to browse available public bike paths on a map.

[R36] The system shall allow any user to search for bike paths connecting two locations.

[R37] The system shall compute suggested routes based on path quality and distance.

[R38] The system shall rank suggested routes according to their safety and quality.

8. **Navigation**

[R39] The system shall display the user's current GPS position during navigation along a selected path.

9. **Alerts**

[R40] The system shall send pop-ups to warn users about nearby obstacles or closed path segments during an active trip.

10. **External Services Integration**

[R41] The system shall interface with map and geocoding services to translate addresses into coordinates and render paths.

[R42] 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 manage their profile settings.

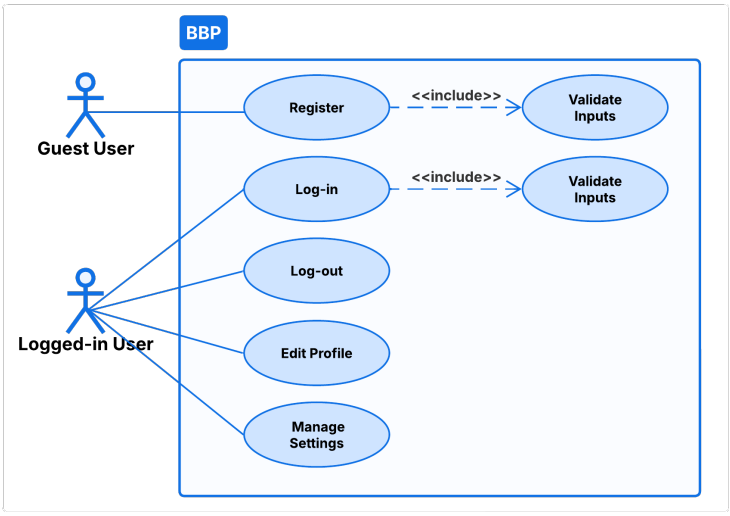

Figure 3.1: Authentication and Profile Management Use Case Diagram

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.

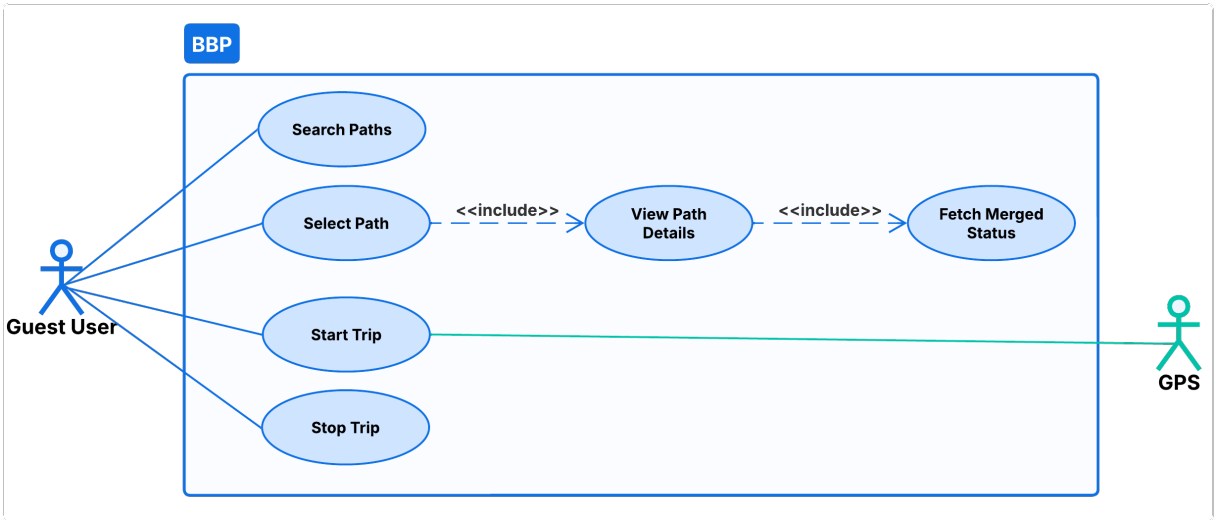

Figure 3.2: Guest User Use Case Diagram

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. Logged-in 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 logged-in user to verify their validity. Logged-in 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, logged-in users can view their activity history and per-trip or overall statistics.

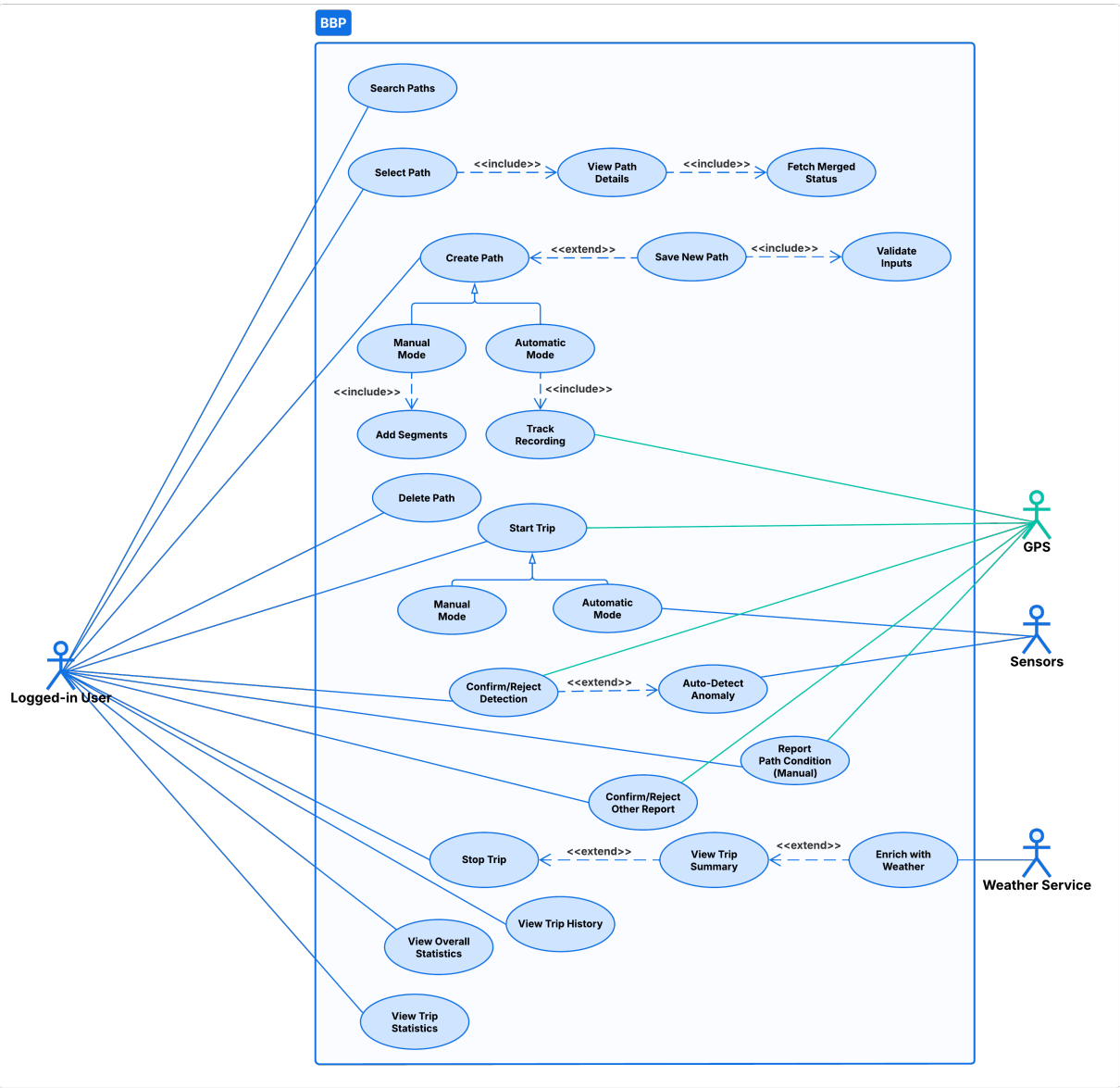

Figure 3.3: Logged-in User Use Case Diagram

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 home 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 in use, 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

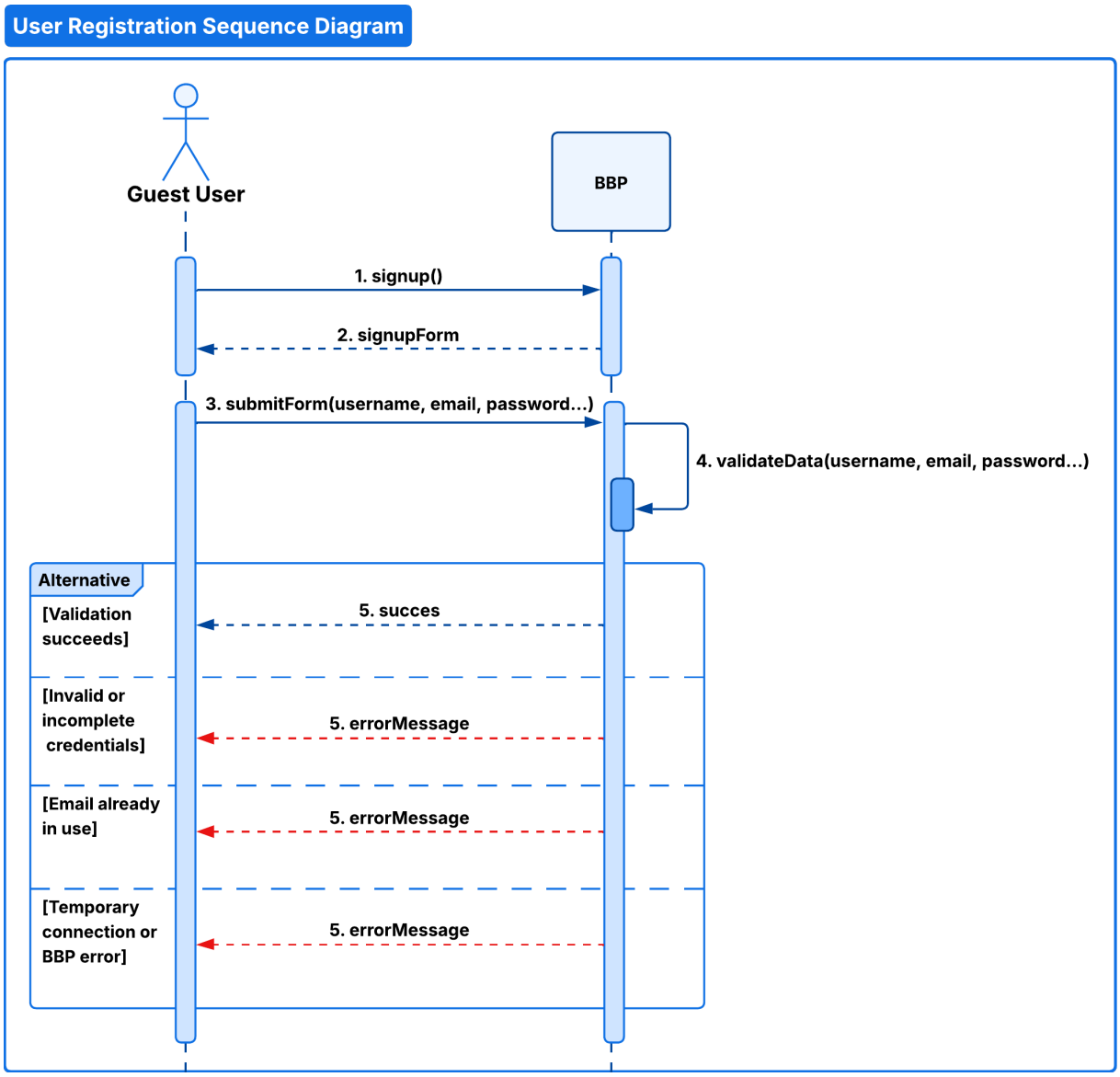

Figure 3.4: User Registration Sequence Diagram

[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

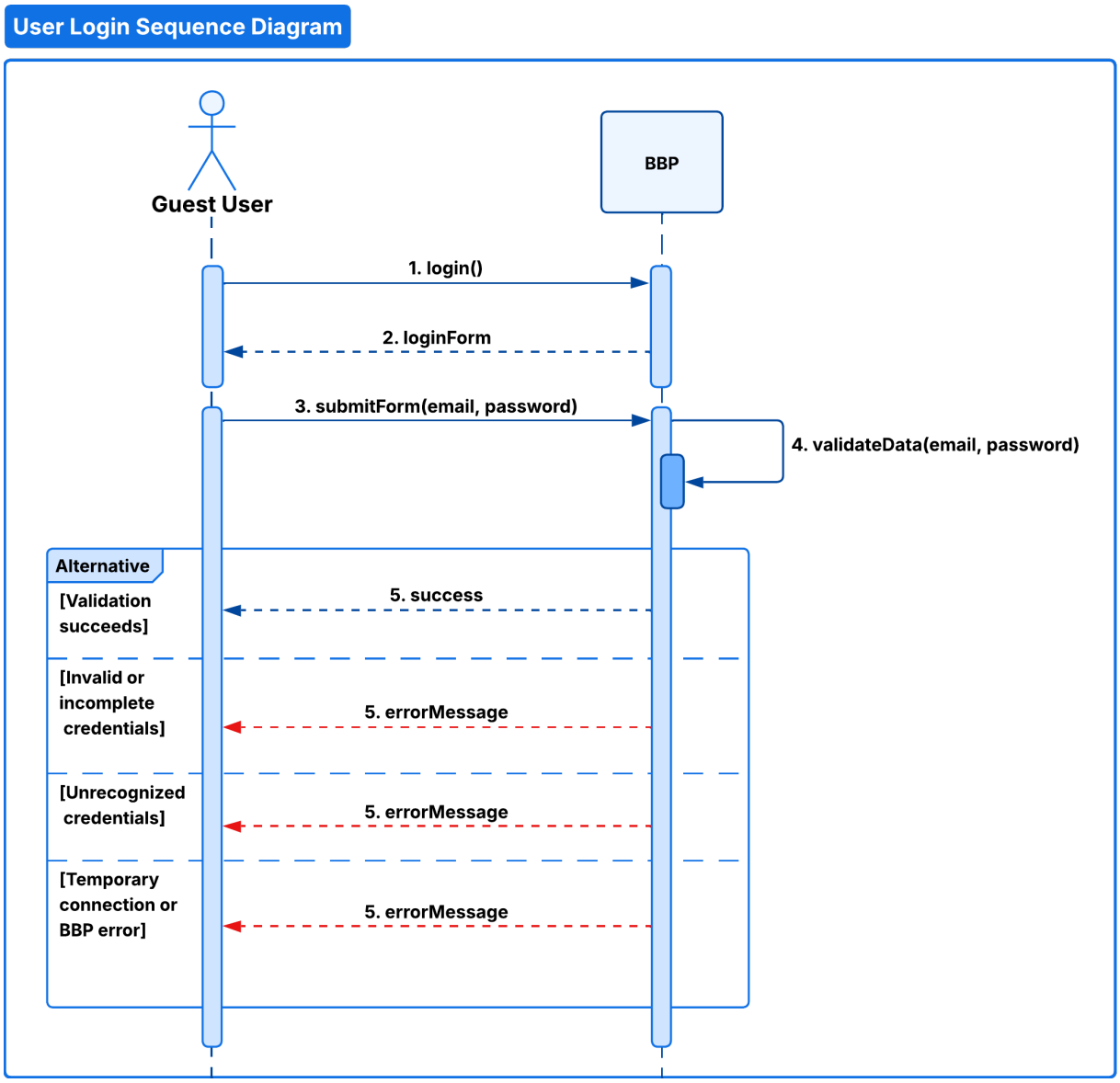

Figure 3.5: User Log In Sequence Diagram

[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

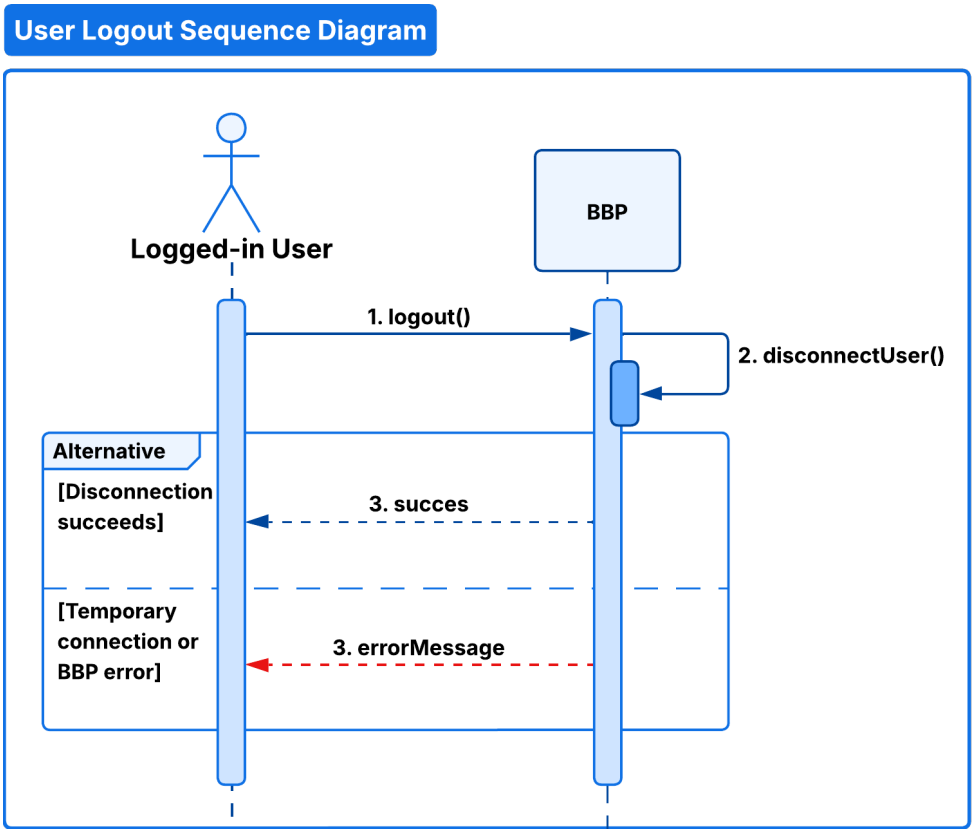

Figure 3.6: User Log Out Sequence Diagram

[UC4] - 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, email). 3. The actor modifies one or more fields and submits the changes. 4. The system validates the provided data. 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 3.4: Edit Personal Profile Process Detail

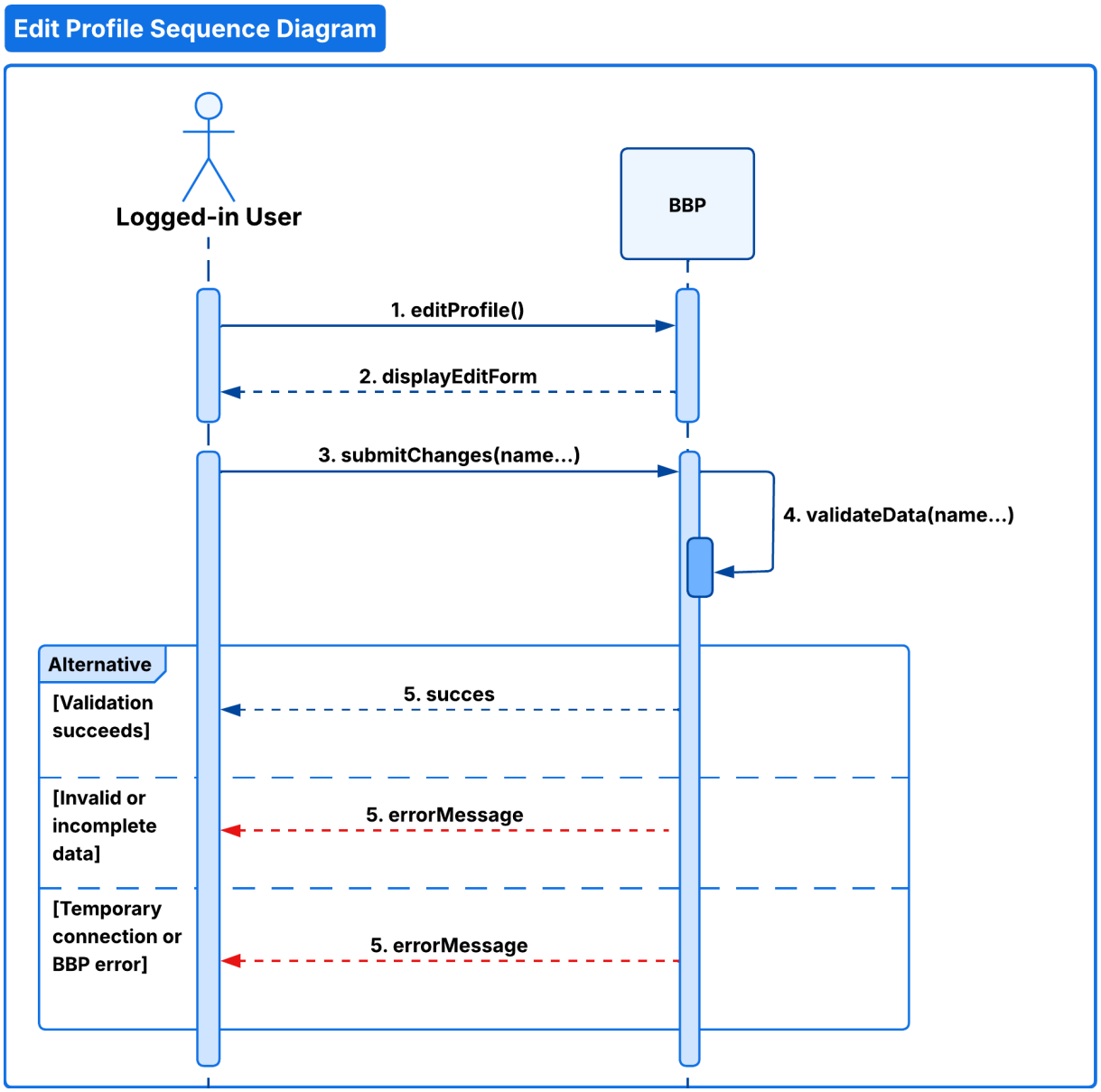

Figure 3.7: Edit Personal Profile Sequence Diagram

[UC5] - 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 enters start and destination points. 2. The system validates the provided inputs. 3. 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 3.5: Search for a Path Process Detail

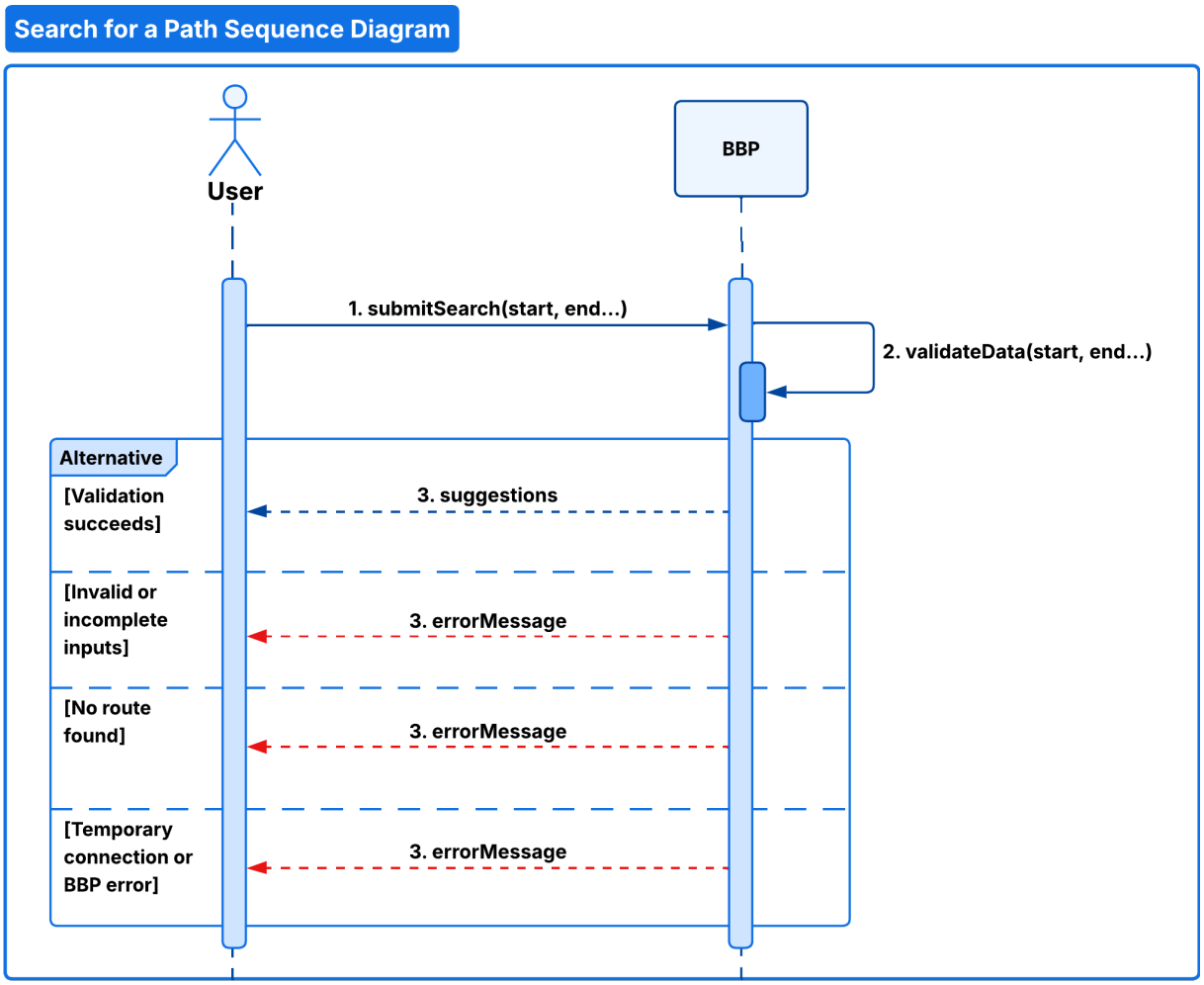

Figure 3.8: Search for a Path Sequence Diagram

[UC6] - 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 3.6: Select a Path Process Detail

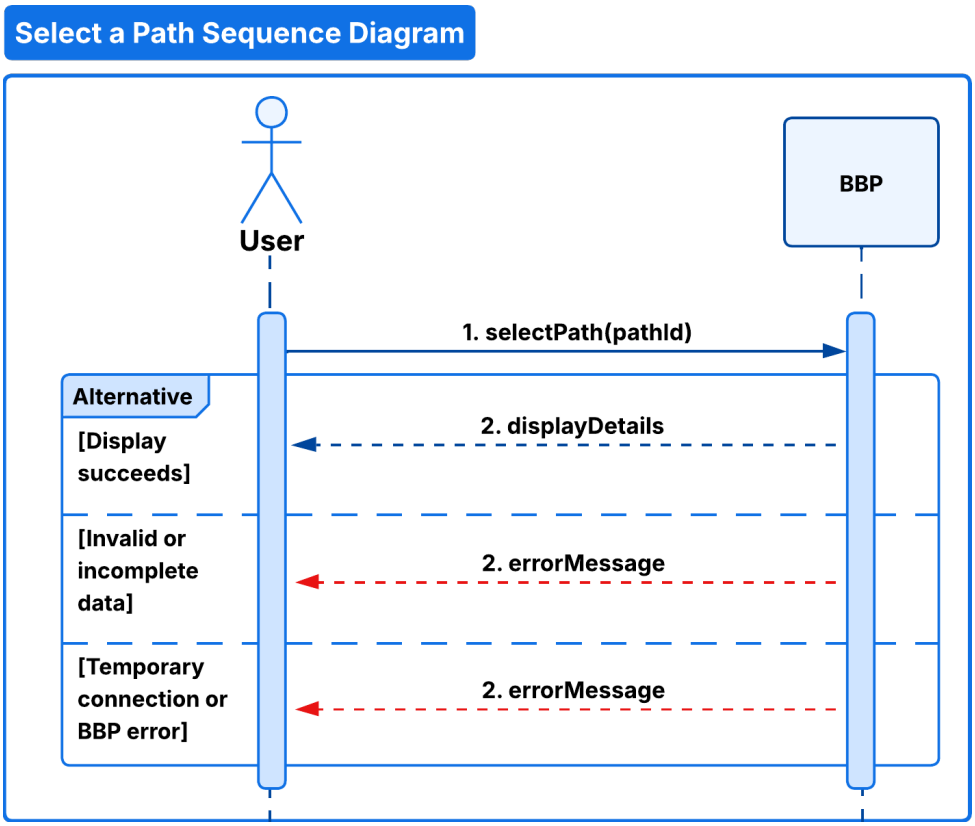

Figure 3.9: Select a Path Sequence Diagram

[UC7] - 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 the creation button. 2. The system displays an input page (metadata + mode selection). 3. The actor fills in metadata (e.g., name, description, visibility) and selects the manual mode. 4. The system displays a map editor. 5. The actor 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 3.7: Create a Path in Manual Mode Process Detail

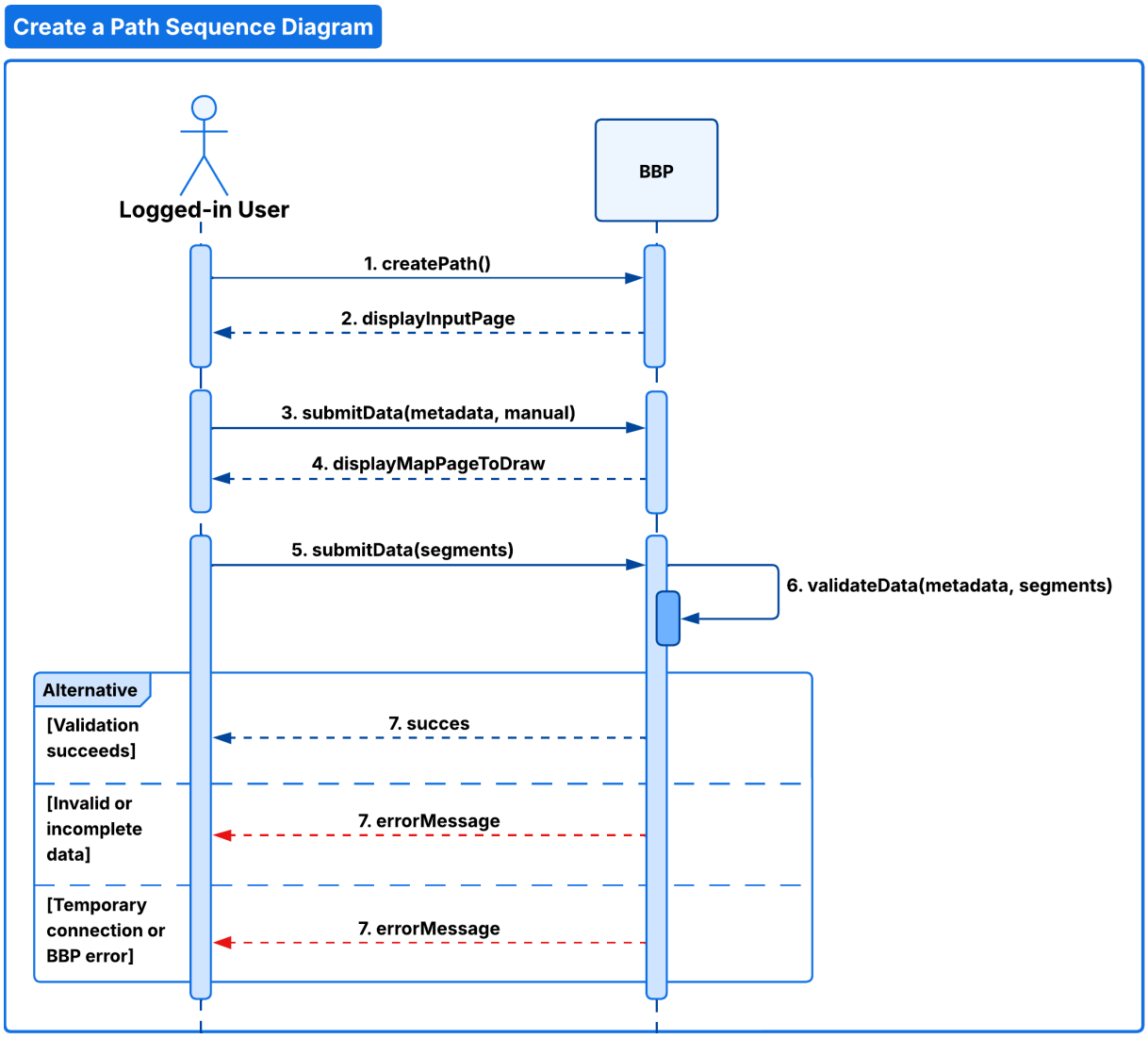

Figure 3.10: Create a Path in Manual Mode Sequence Diagram

[UC8] - 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 the creation button. 2. The system displays an input page (metadata + mode selection) 3. The actor fills in metadata (e.g., name, description, visibility) and selects the automatic mode. 4. The system validates the provided metadata. 5. If the validation succeeds, the system confirms readiness and starts GPS tracking. 6. The system continuously collects geo-coordinates from the GPS while the actor is moving. 7. Once tracking stops, the system validates the recorded GPS data. 8. 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 3.8: Create a Path in Automatic Mode Process Detail

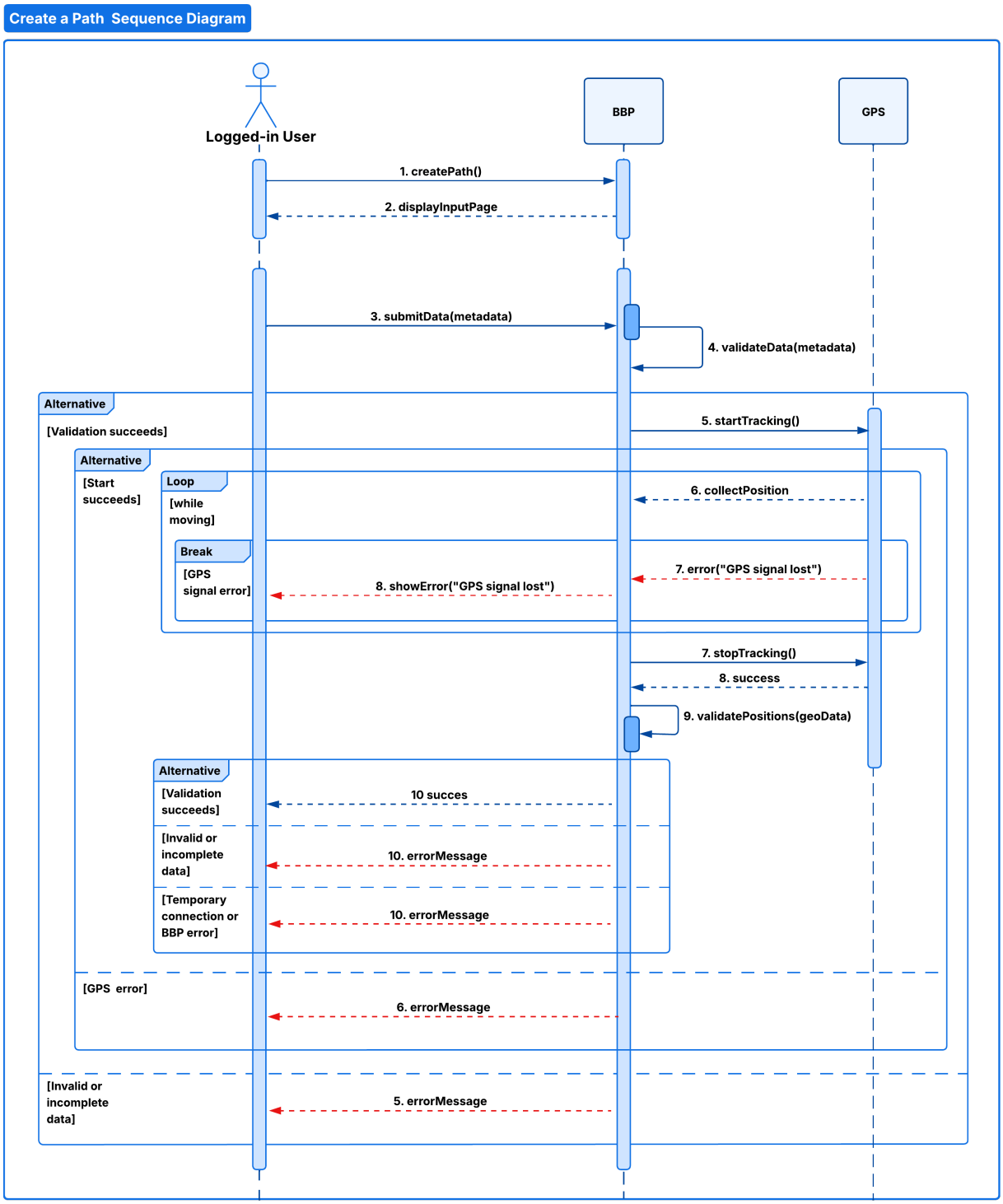

Figure 3.11: Create a Path in Automatic Mode Sequence Diagram

[UC9] - View Path Library

[UC10] - 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 taps on the visibility indicator of a specific path. 2. The system displays a confirmation dialog to switch the path visibility (e.g., from Private to Public or vice versa). 3. The actor confirms the action and submits the changes. 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 3.9: Manage Path Visibility Process Detail

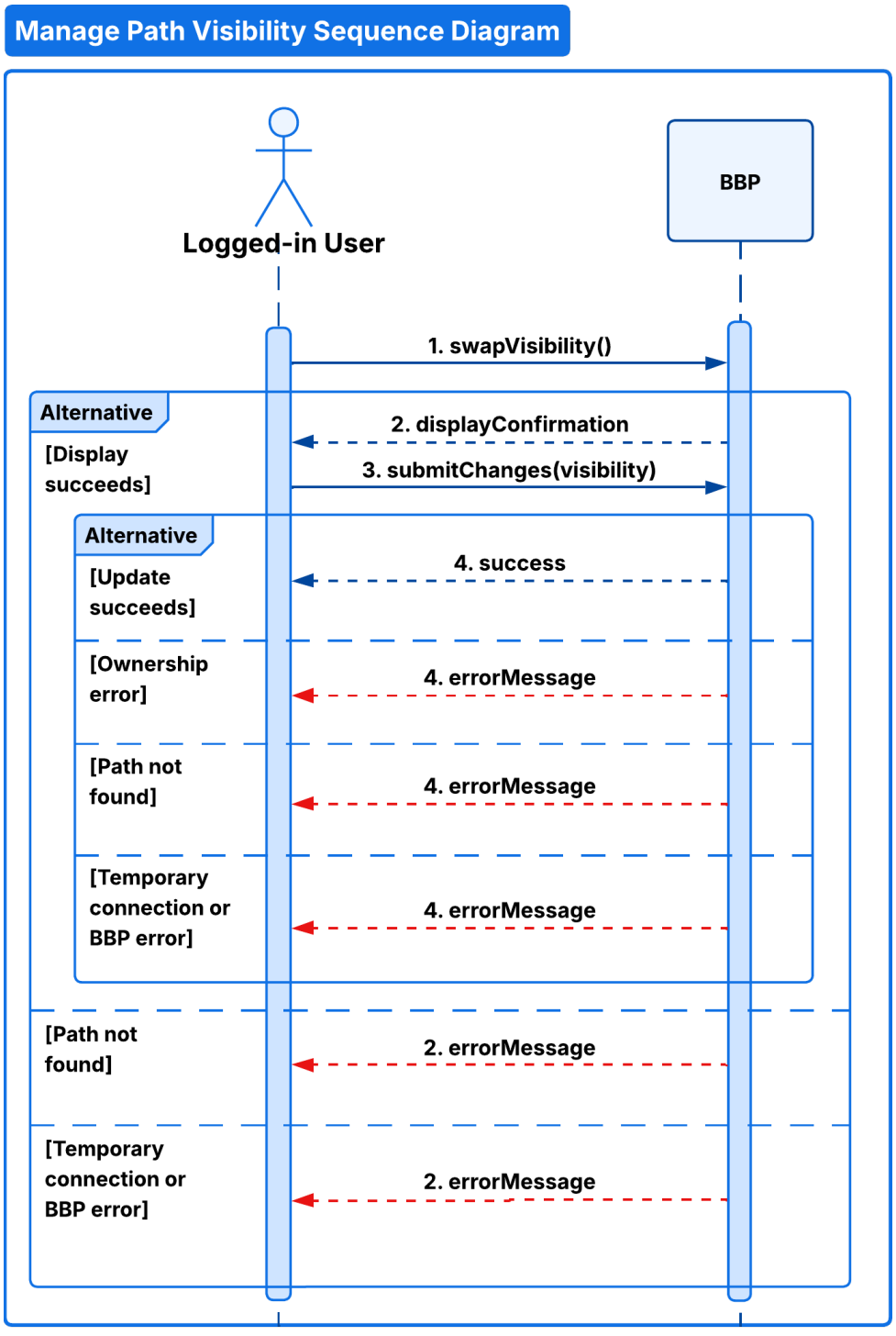

Figure 3.12: Manage Path Visibility Sequence Diagram

[UC11] - 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 his path 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 3.10: Delete a Path Process Detail

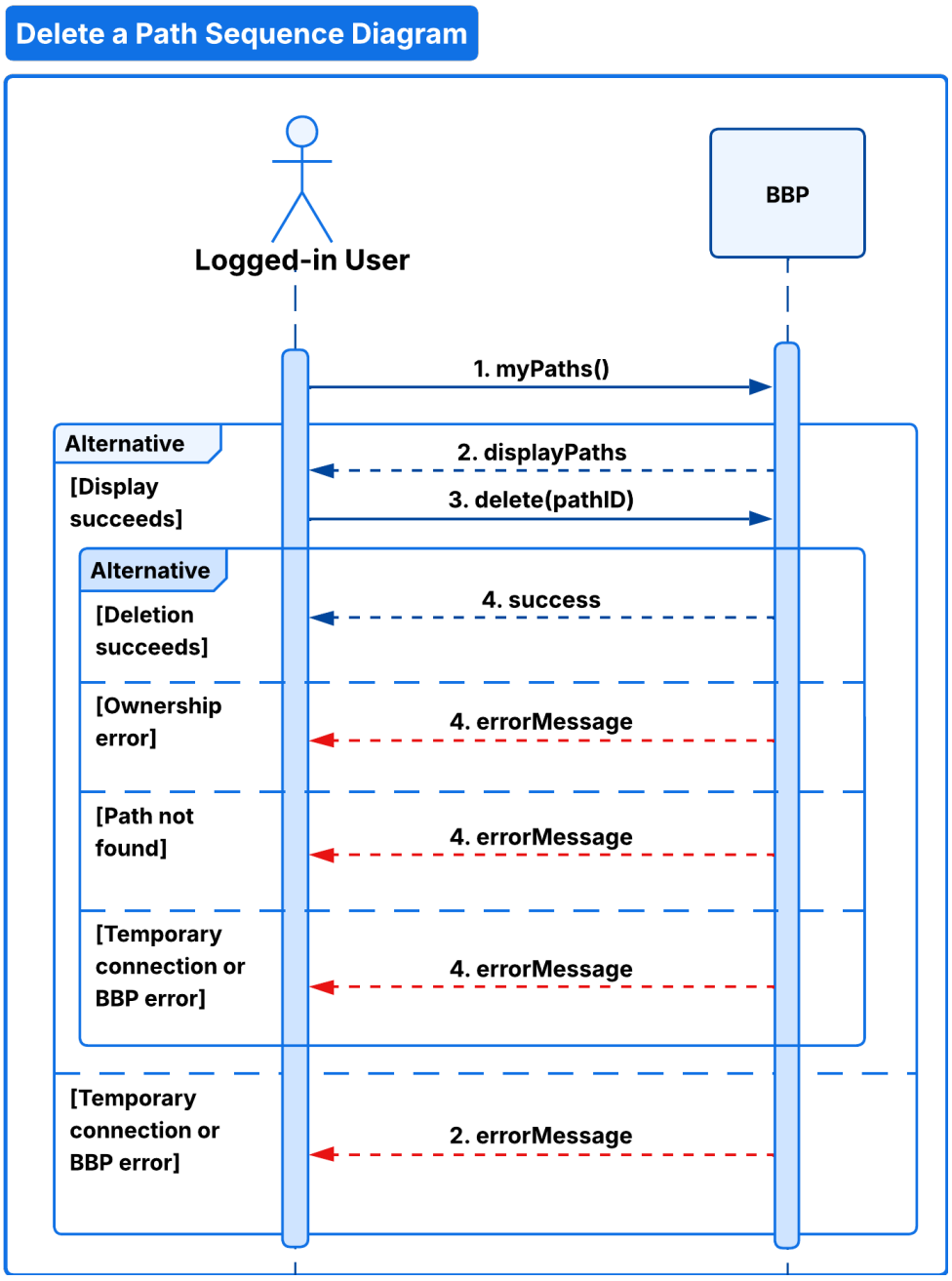

Figure 3.13: Delete a Path Sequence Diagram

[UC12] - Start a Trip as Guest User

Actor(s)	Guest User
Entry Condition	The actor selects a path whose origin corresponds to their current location and intends to follow it without authenticating
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 3.11: Start a Trip as Guest User Process Detail

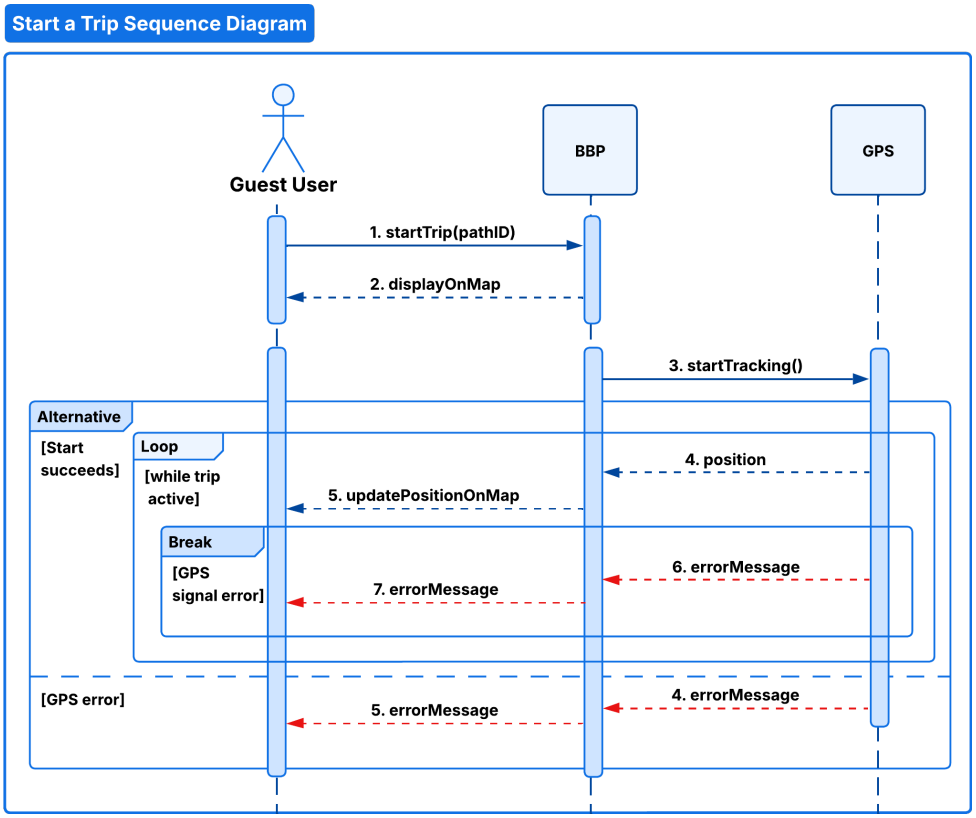

Figure 3.14: Start a Trip as Guest User Sequence Diagram

[UC13] - 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 whose origin corresponds to their current location, and wants to start the trip in manual mode.
Event Flow	1. The actor chooses to start the trip on the selected path. 2. The system asks the user if he wants to enable automatic mode. 3. The actor declines. 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 records trip data (e.g., time, distance, speed) for the actor's statistics.

Table 3.12: Start a Trip in Manual Mode as a Logged-in User Process Detail

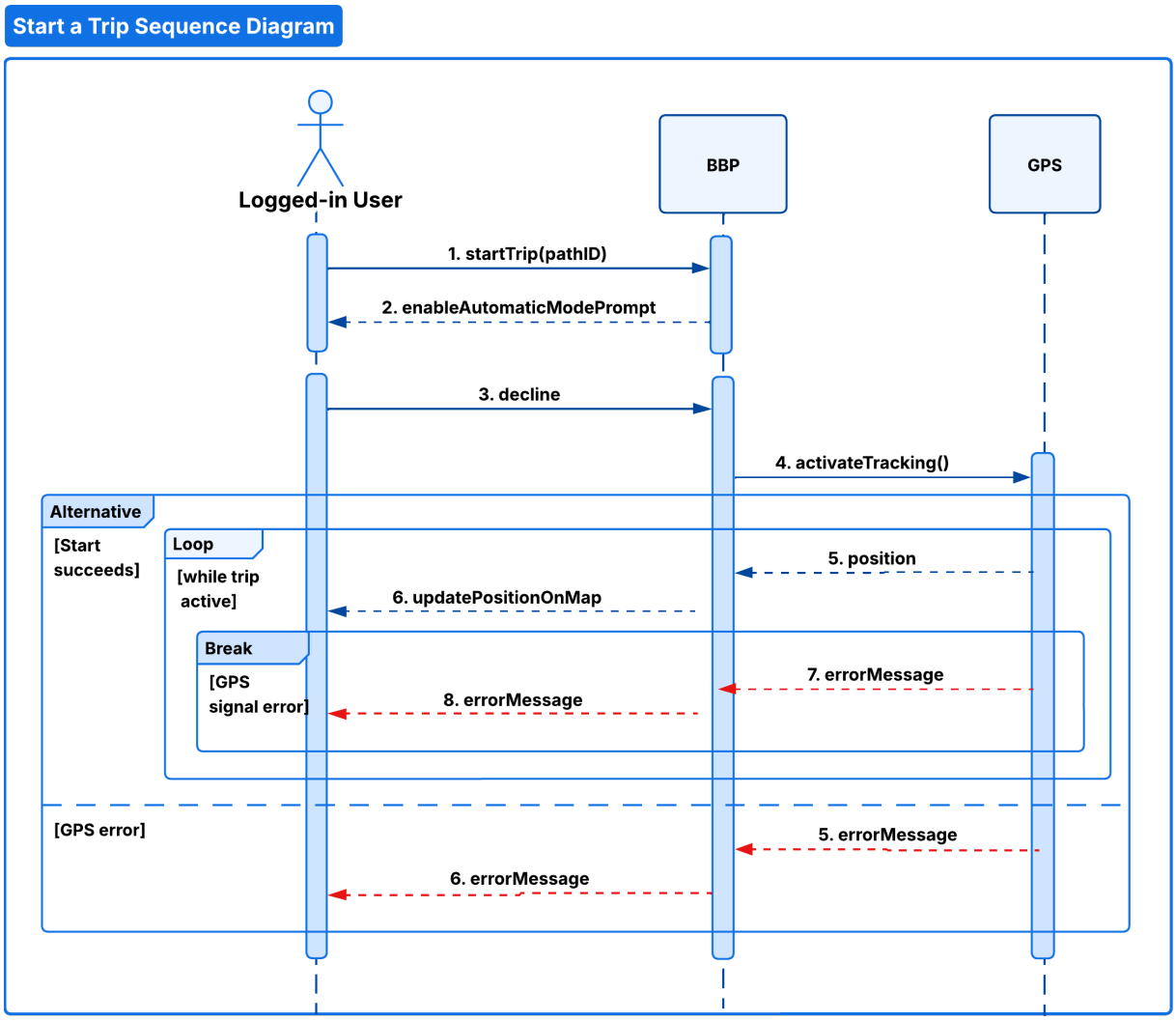

Figure 3.15: Start a Trip in Manual Mode as a Logged-in User Sequence Diagram

[UC14] - 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 whose origin corresponds to their current location, 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 asks the user if he wants to enable automatic mode. 3. The actor accepts. 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 3.13: Start a Trip in Automatic Mode as a Logged-in User Process Detail

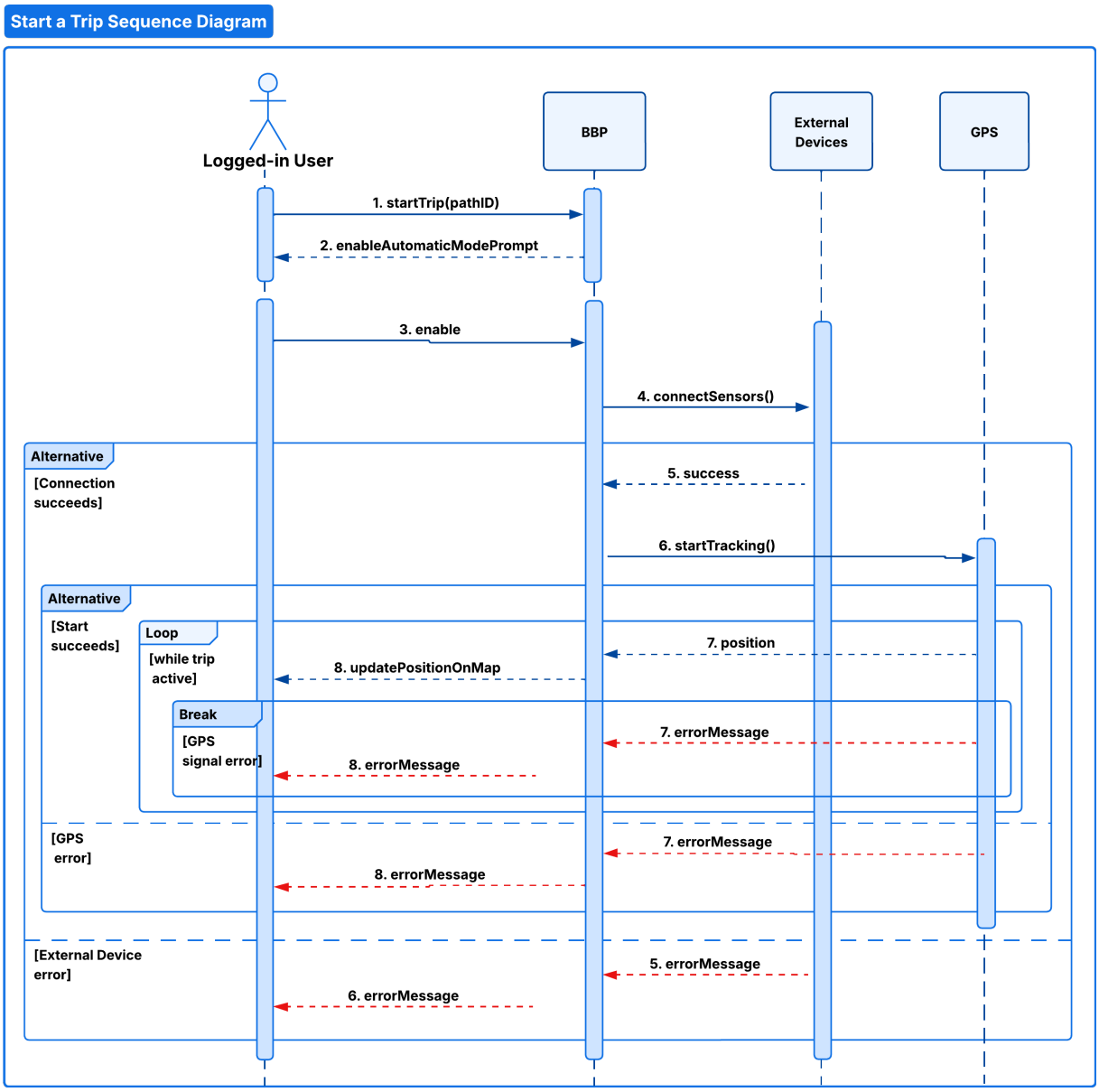

Figure 3.16: Start a Trip in Automatic Mode as a Logged-in User Sequence Diagram

[UC15] - 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 3.14: Stop a Trip as a Guest User Process Detail

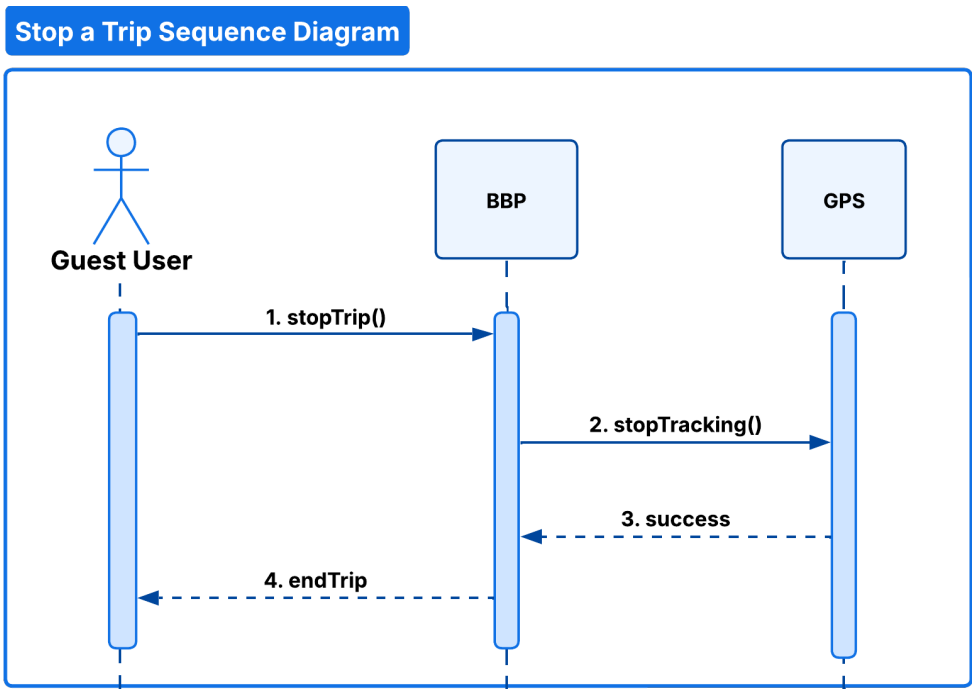

Figure 3.17: Stop a Trip as a Guest User Sequence Diagram

[UC16] - 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). 4. If possible, the system enriches the trip data with meteorological information (e.g., weather, temperature), retrieved from an external service. 5. 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 3.15: Stop a Trip as a Logged-in User Process Detail

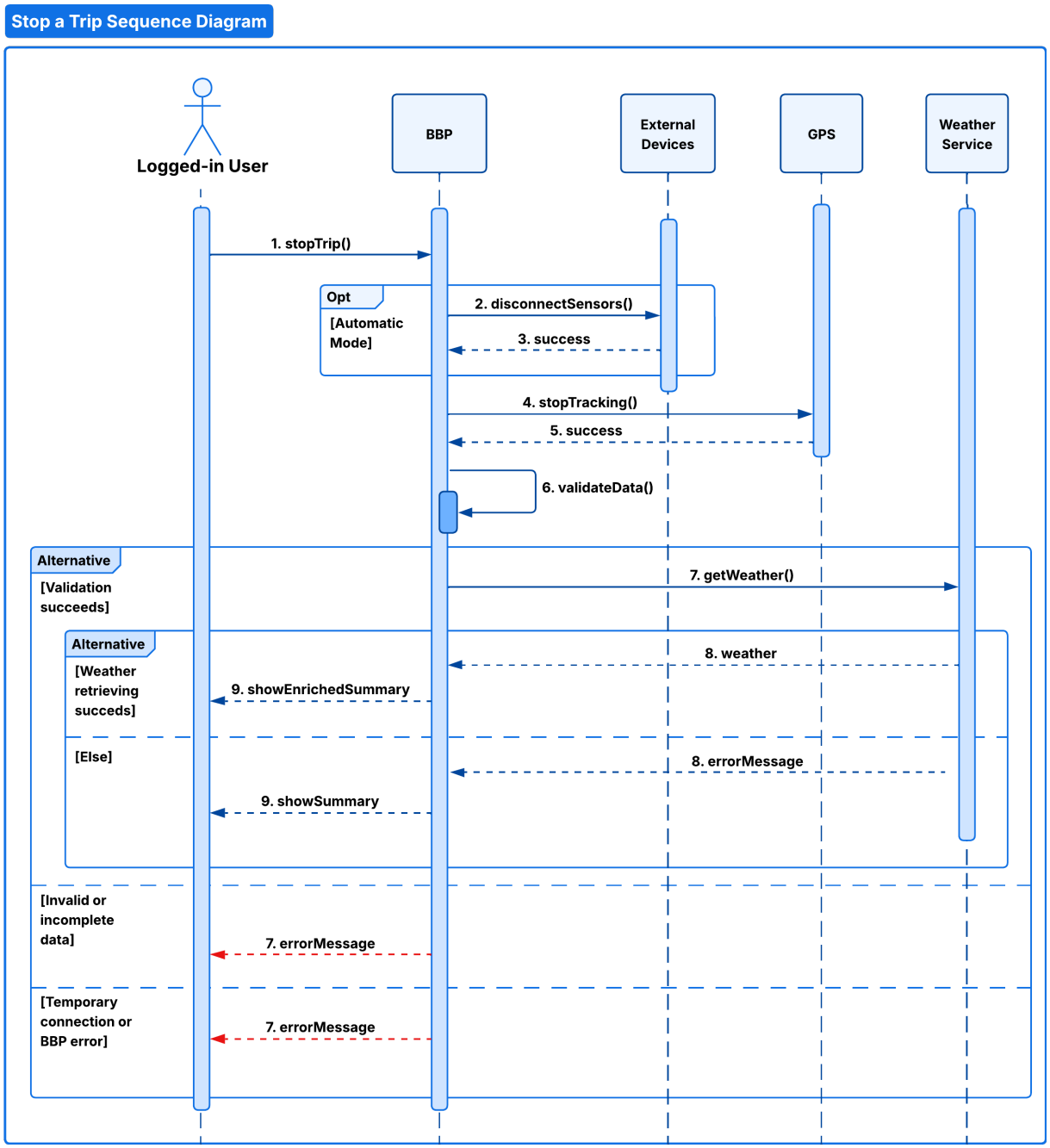

Figure 3.18: Stop a Trip as a Logged-in User Sequence Diagram

[UC17] - 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 3.16: Make a Report in Manual Mode Process Detail

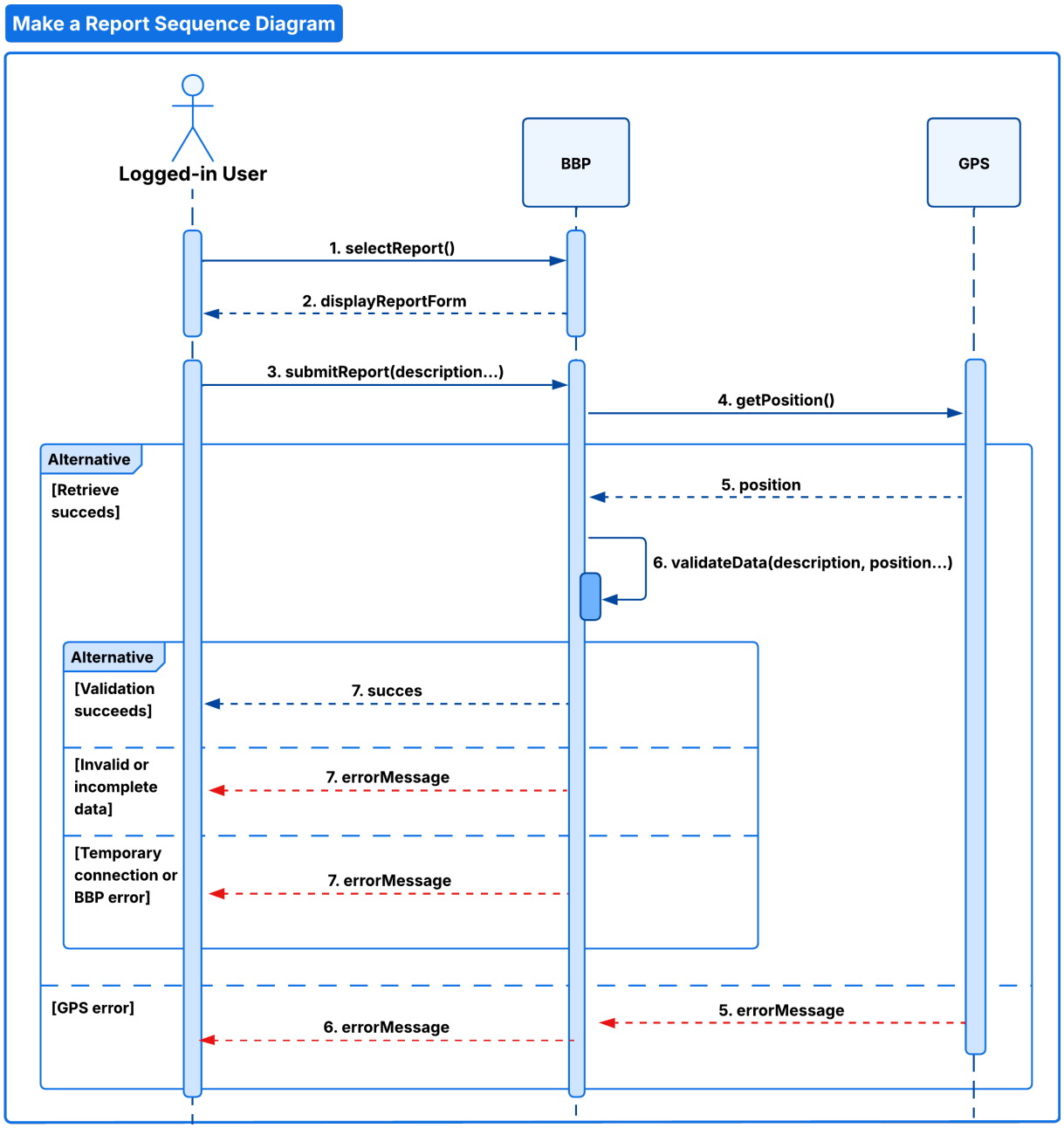

Figure 3.19: Make a Report in Manual Mode Sequence Diagram

[UC18] - 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. 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 3.17: Make a Report in Automatic Mode Process Detail

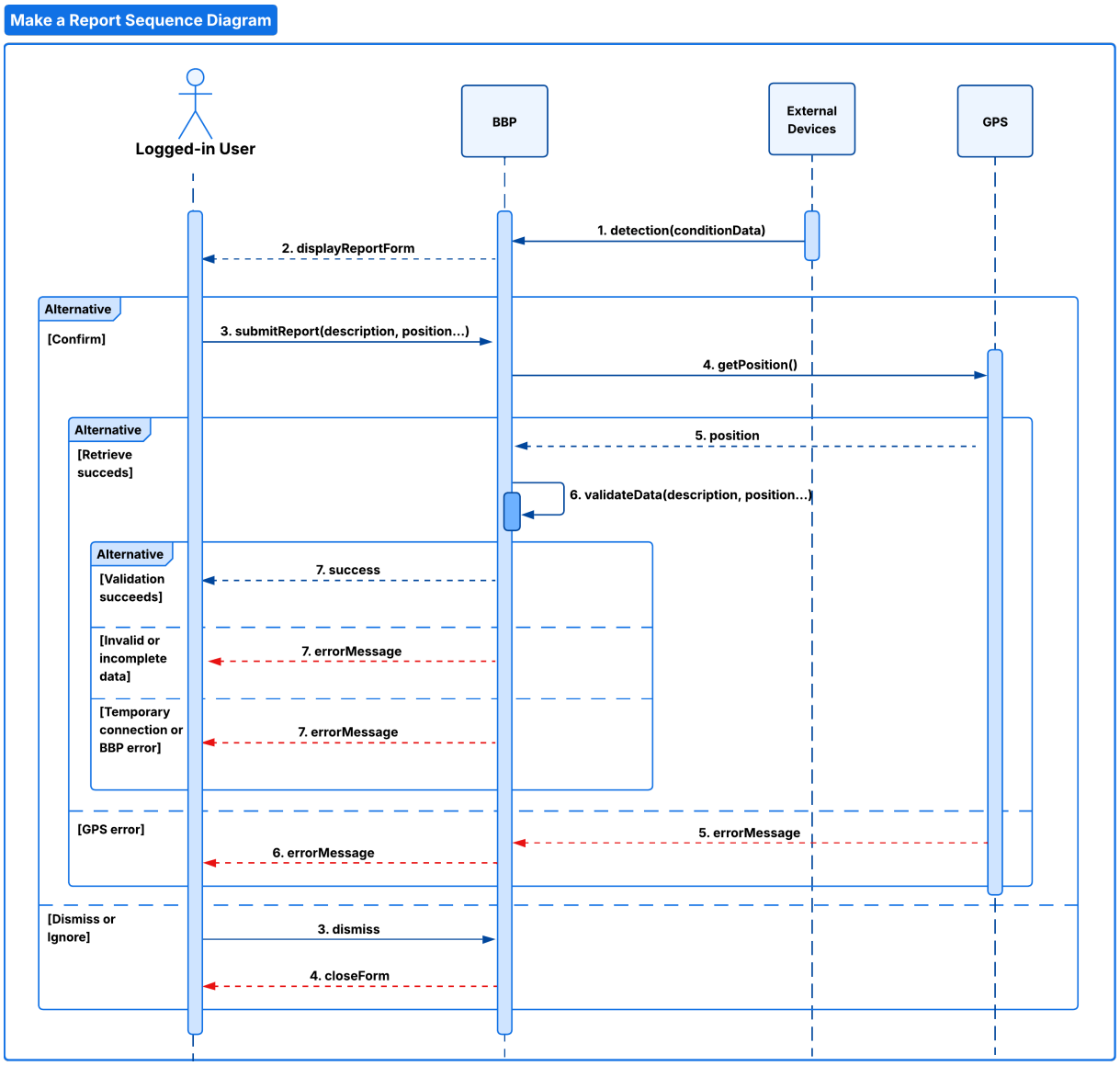

Figure 3.20: Make a Report in Automatic Mode Sequence Diagram

[UC19] - 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. 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 3.18: Confirm a Report Process Detail

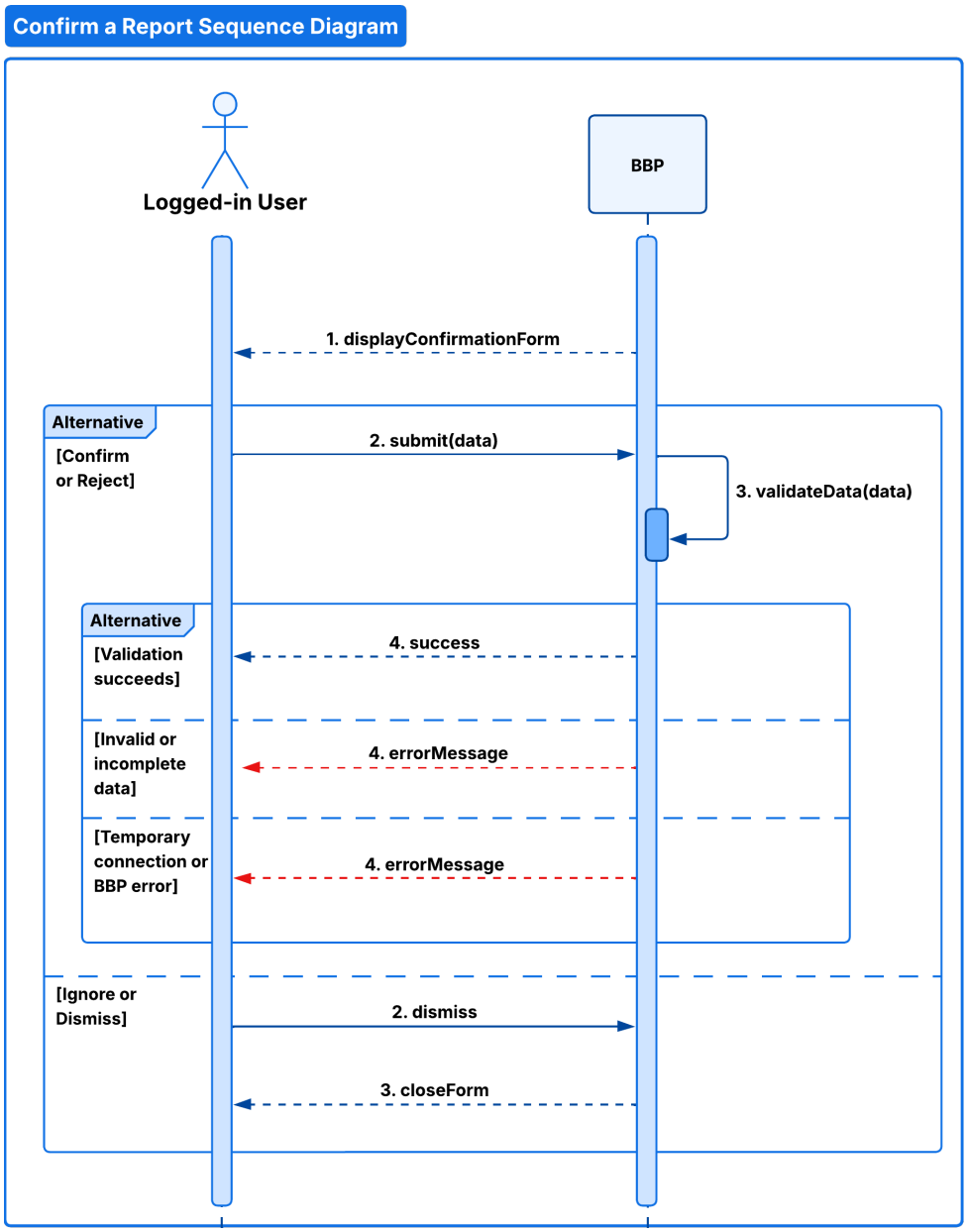

Figure 3.21: Confirm a Report Sequence Diagram

[UC20] - View Trip History

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated and has completed at least one trip.
Event Flow	1. The actor accesses his trip history section. 2. The system displays the trips associated with the actor. 3. The actor can select a specific trip to view detailed data (map, statistics, reports). 4. 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 3.19: View Trip History Process Detail

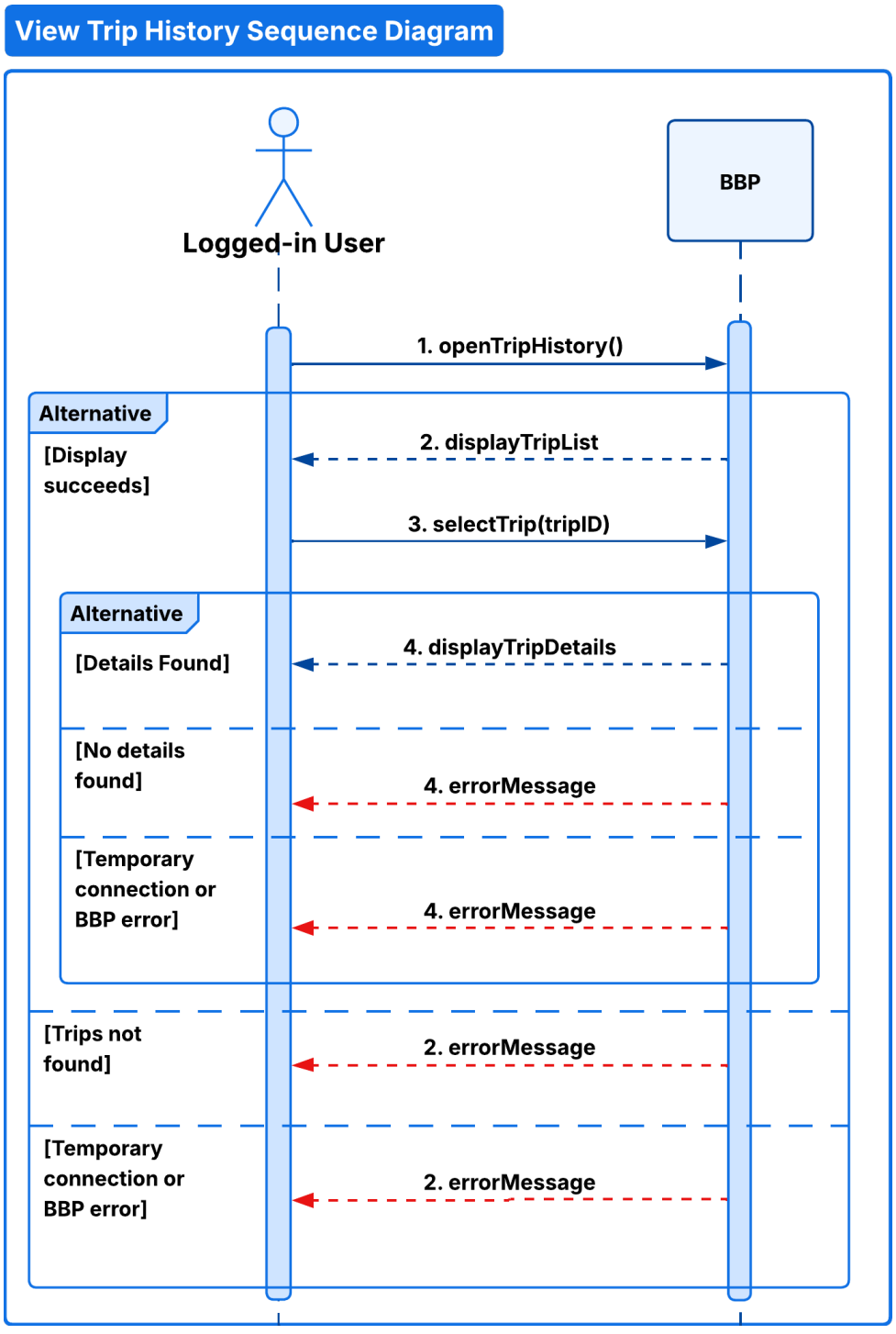

Figure 3.22: View Trip History Sequence Diagram

[UC21] - Delete a Trip

[UC22] - View Trip Statistics

Actor(s)	Logged-in User
Entry Condition	The actor is authenticated, has completed one or more trips and is visualizing his trips section.
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	If 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

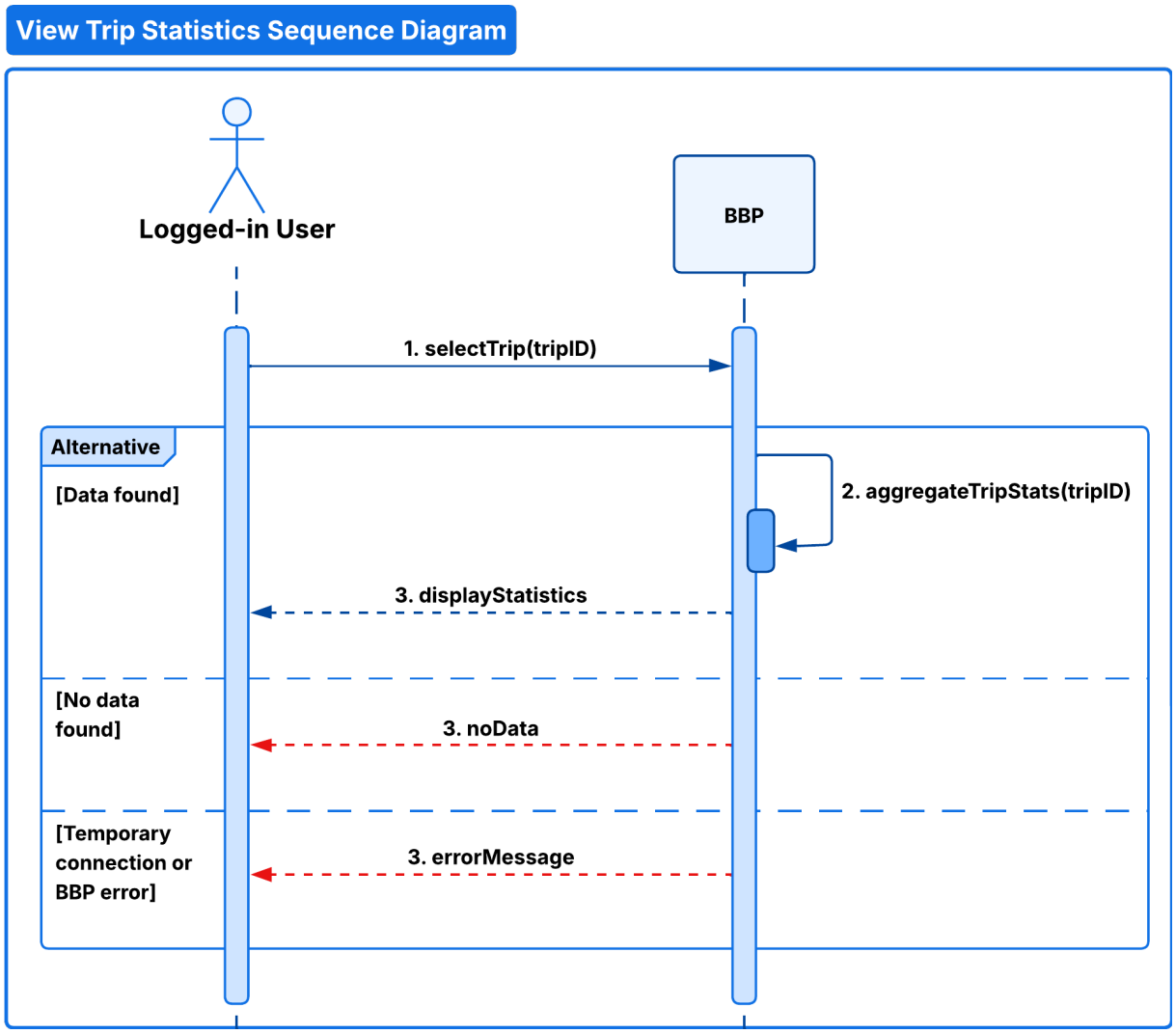

Figure 3.23: View Trip Statistics Sequence Diagram

[UC23] - 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 accesses his statistics section. 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 3.21: View Overall Statistics Process Detail

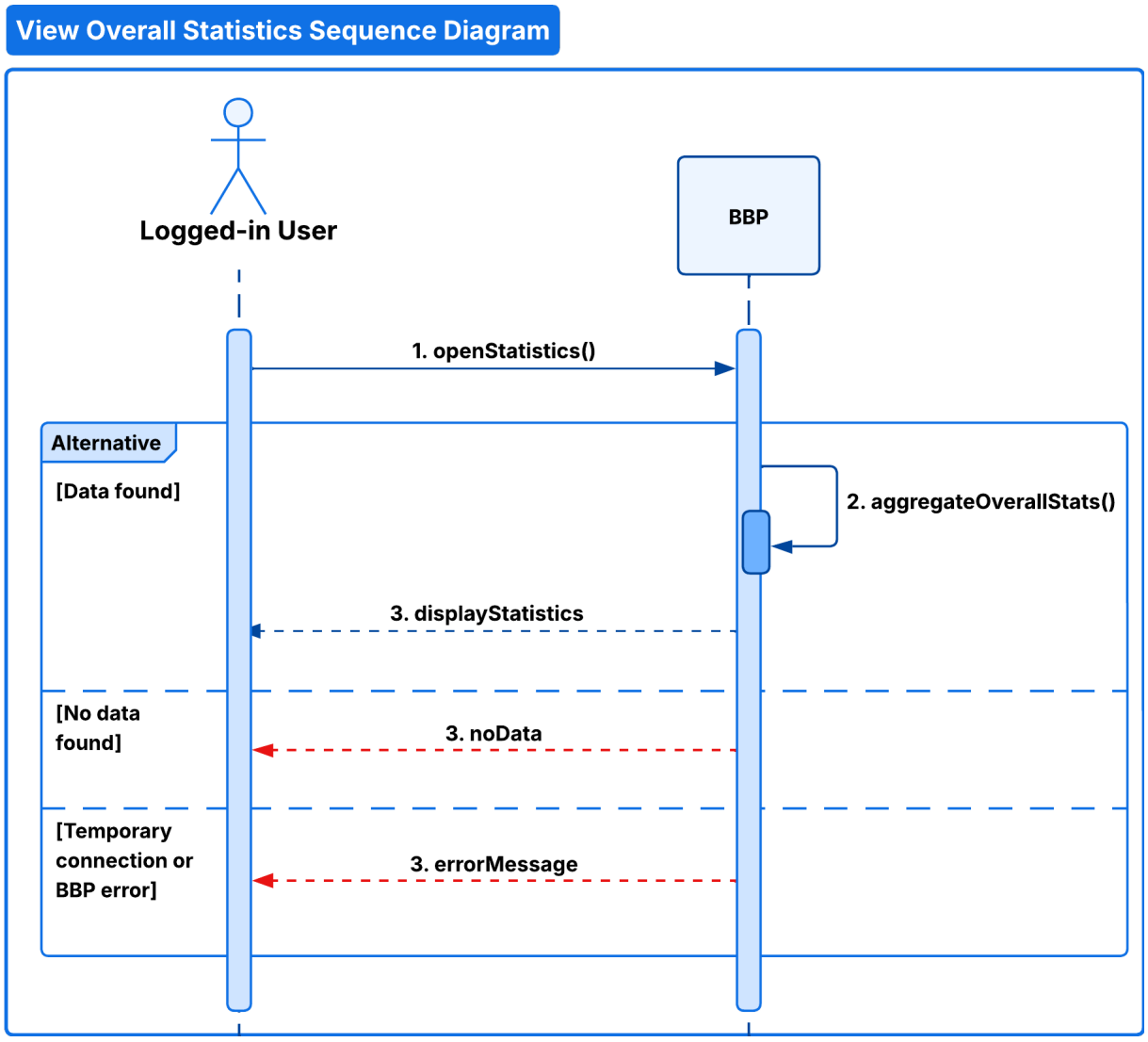

Figure 3.24: View Overall Statistics Sequence Diagram

3.2.4. Requirement Mapping

Goal	Requirements	Domain Assumptions
G1	R5, R31, R32, R33, R34, R35, R36, R37, R38, R41, R42	D1, D5, D6, D8, D9
G2	R5, R31, R32, R33, R34, R35, R36, R37, R38, R41, R42	D1, D5, D6, D8, D9
G3	R5, R6, R7, R8, R9, R10, R11, R14, R15, R31, R34, R35, R36, R39, R40, R41, R42	D1, D3, D4, D6, D7, D8, D9
G4	R1, R2, R3, R4, R12, R13, R14, R15, R16, R19, R20, R21, R22, R41, R42	D1, D2, D3, D4, D6
G5	R1, R2, R12, R14, R15, R22, R23, R24, R25, R26, R27, R28, R29, R30, R41, R42	D1. D3, D4, D5, D6, D7, D8
G6	R1, R2, R3, R4, R12, R13, R14, R15, R16, R17, R18, R21, R22, R41, R42	D1. D2, D3, D4, D6

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

3.3. Performance Requirements

The system shall provide timely feedback to user interactions. Under normal network conditions, core functionalities such as map visualization, path search, and path selection shall return a result within 5 seconds. User actions that require immediate feedback, such as starting or stopping a trip, submitting a report, or confirming a report, shall be acknowledged within 2 seconds.

During an active trip, the system shall support continuous tracking of the user's position. Location updates shall be processed at least every 2 seconds to enable a smooth visualization of movement along the selected path.

The system shall support multiple users performing common operations concurrently without significant degradation of response times. Under normal operating conditions, the system shall be able to handle at least 200 concurrent users executing standard actions such as browsing paths, starting trips, and submitting reports.

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.

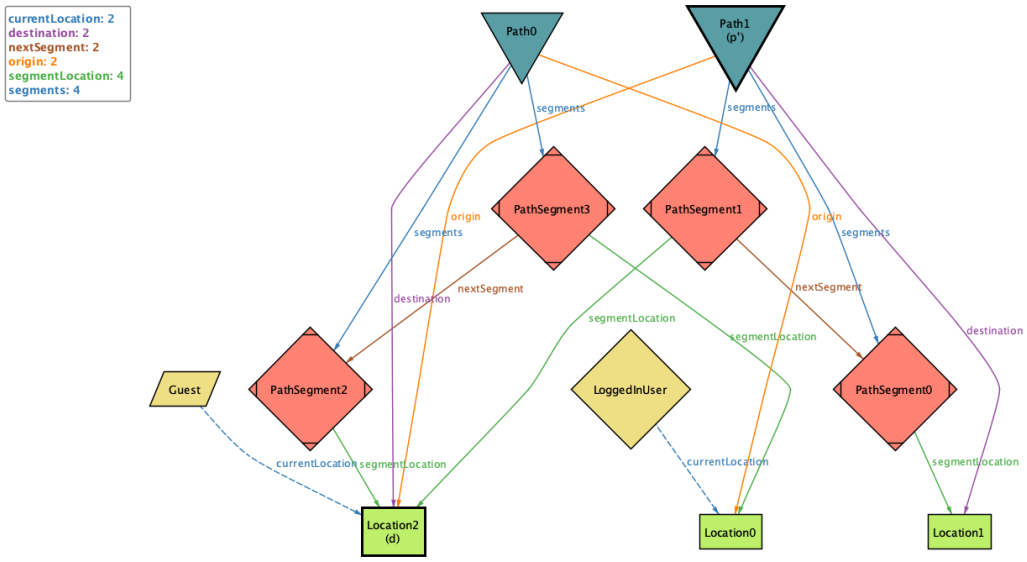

Figure 4.1: Initialization of the route selection process

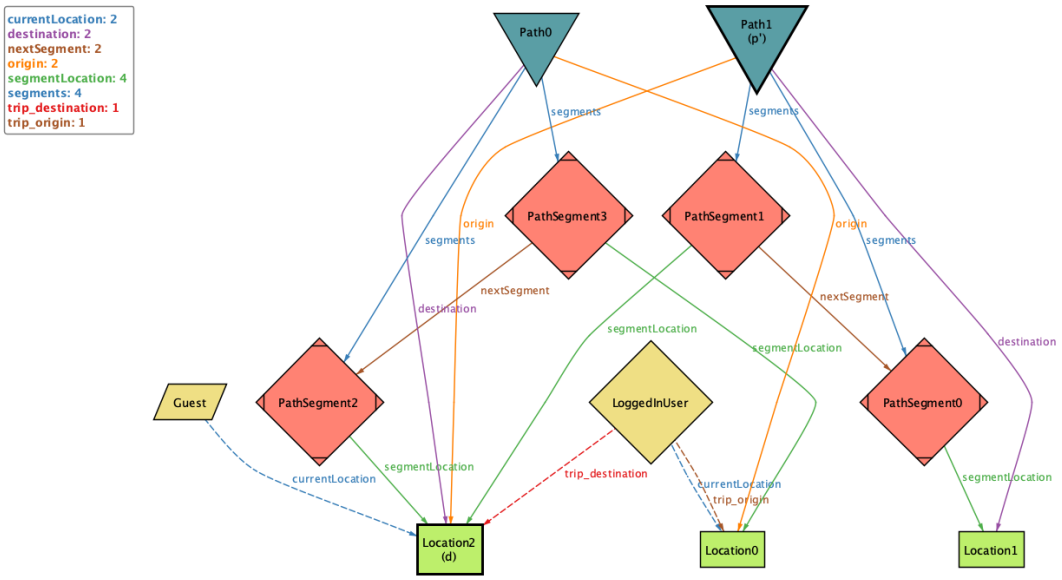

Figure 4.2: Beginning of the route selection process

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

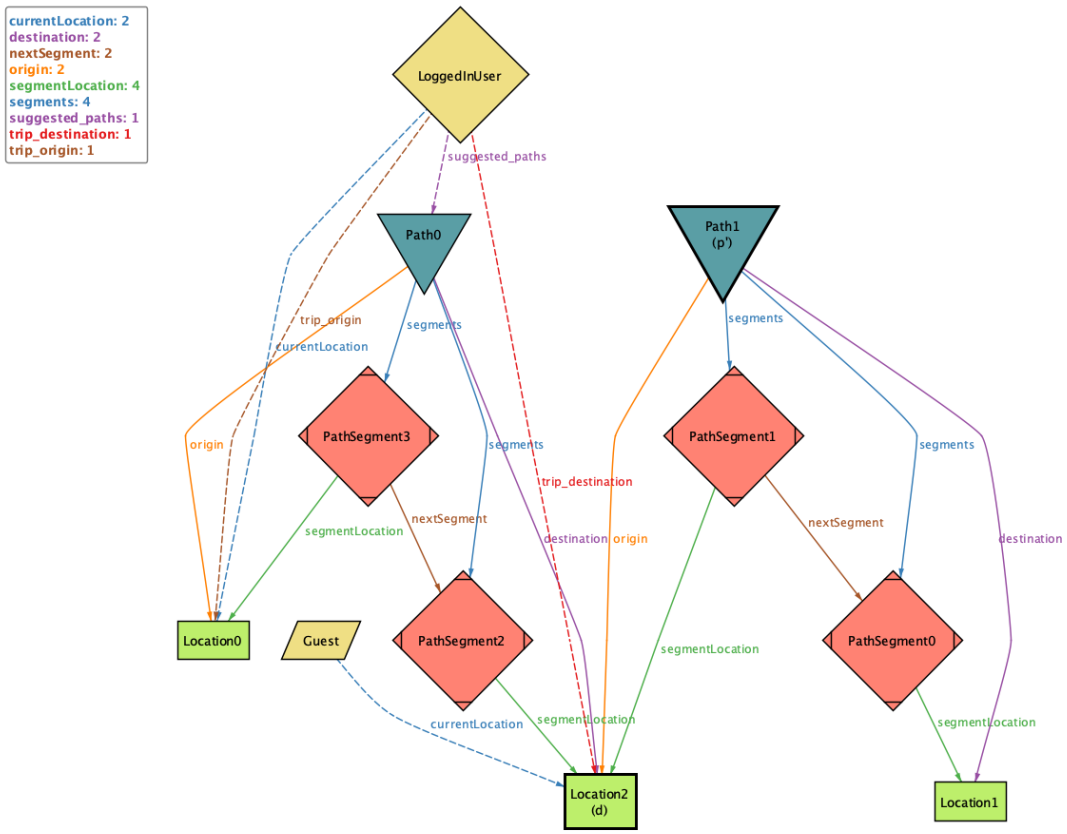

Figure 4.3: System suggesting available paths

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.

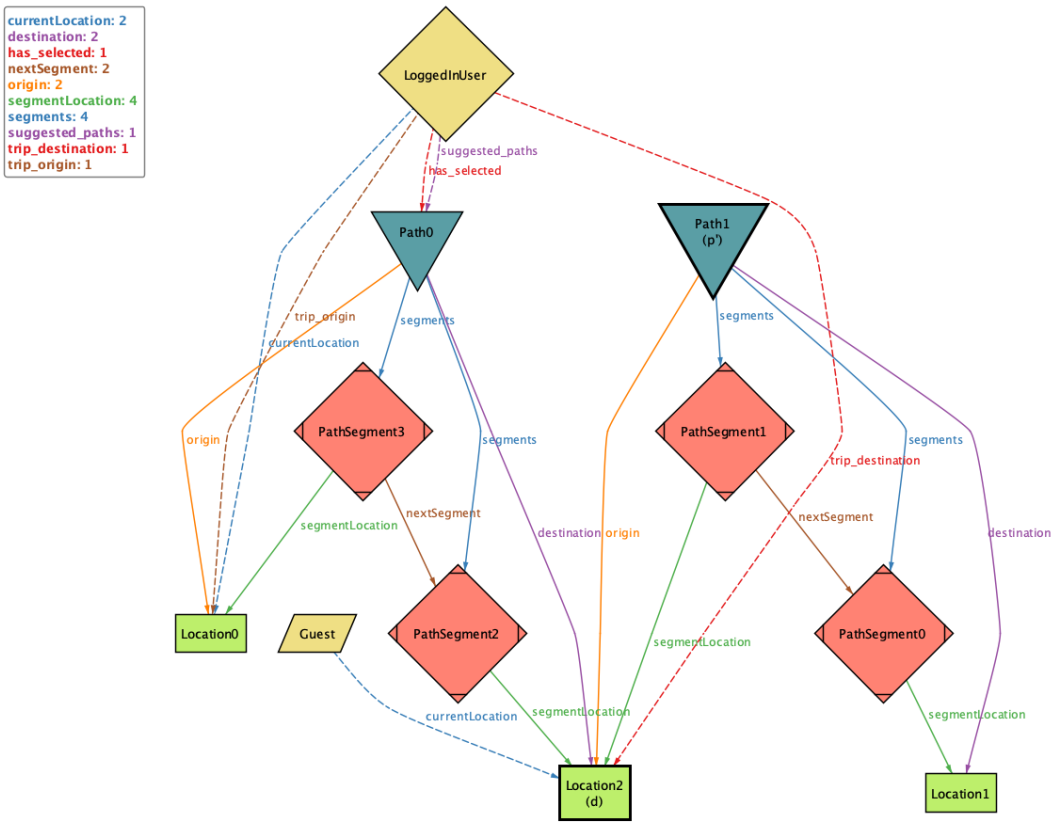

Figure 4.4: User selects a suggested path

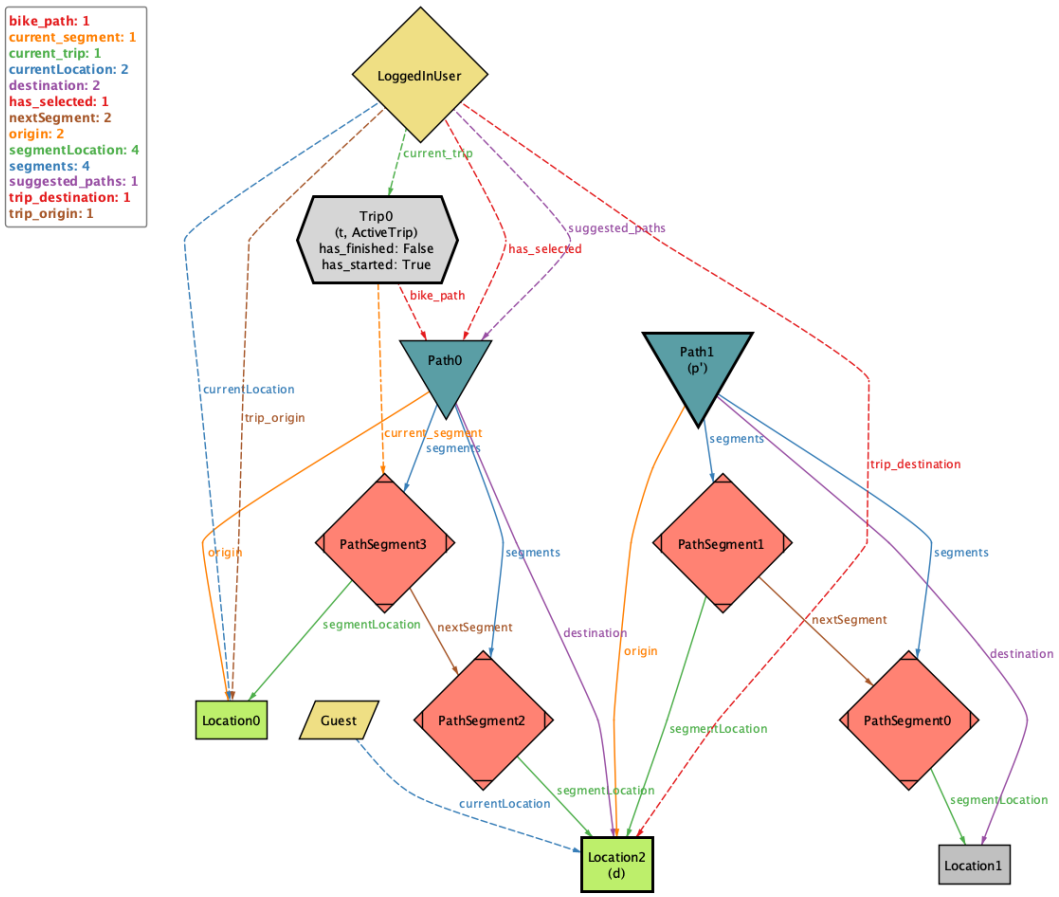

Figure 4.5: User starting the trip

As shown in Figure 4.6, 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).

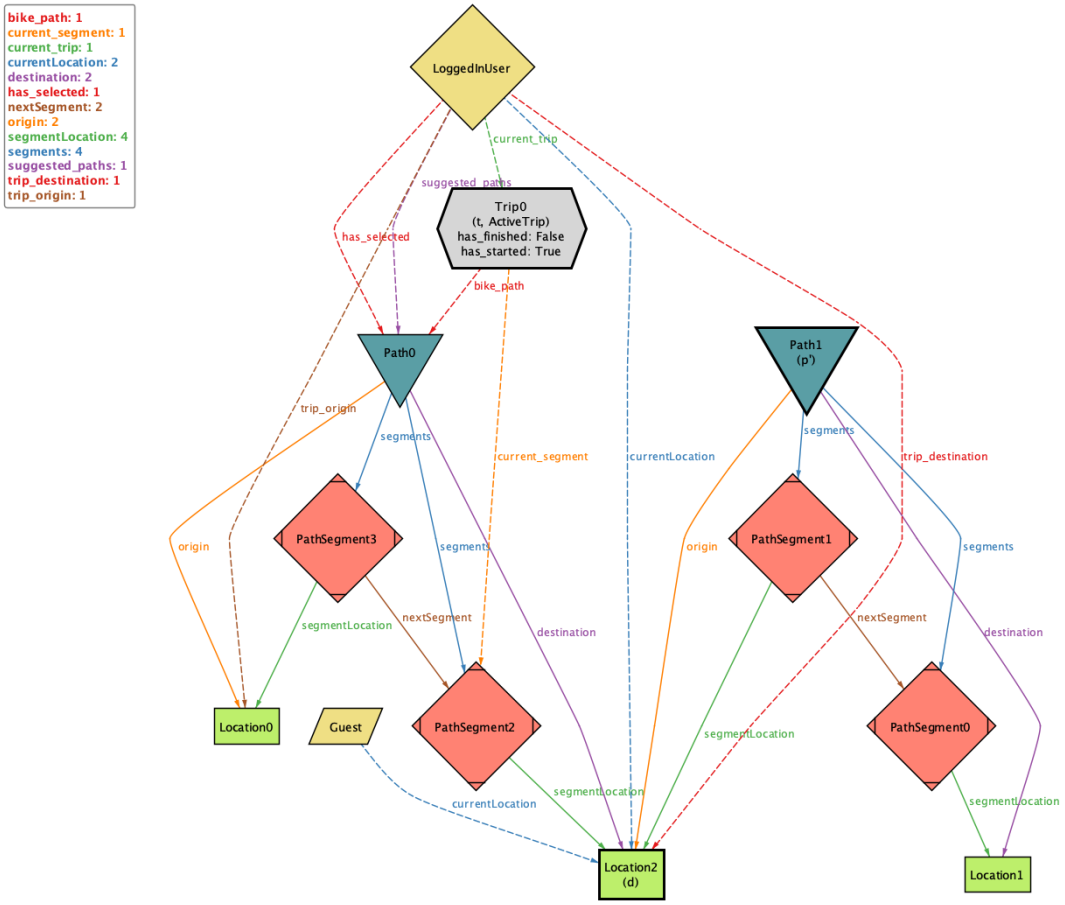

Figure 4.6: User moving to the next segment

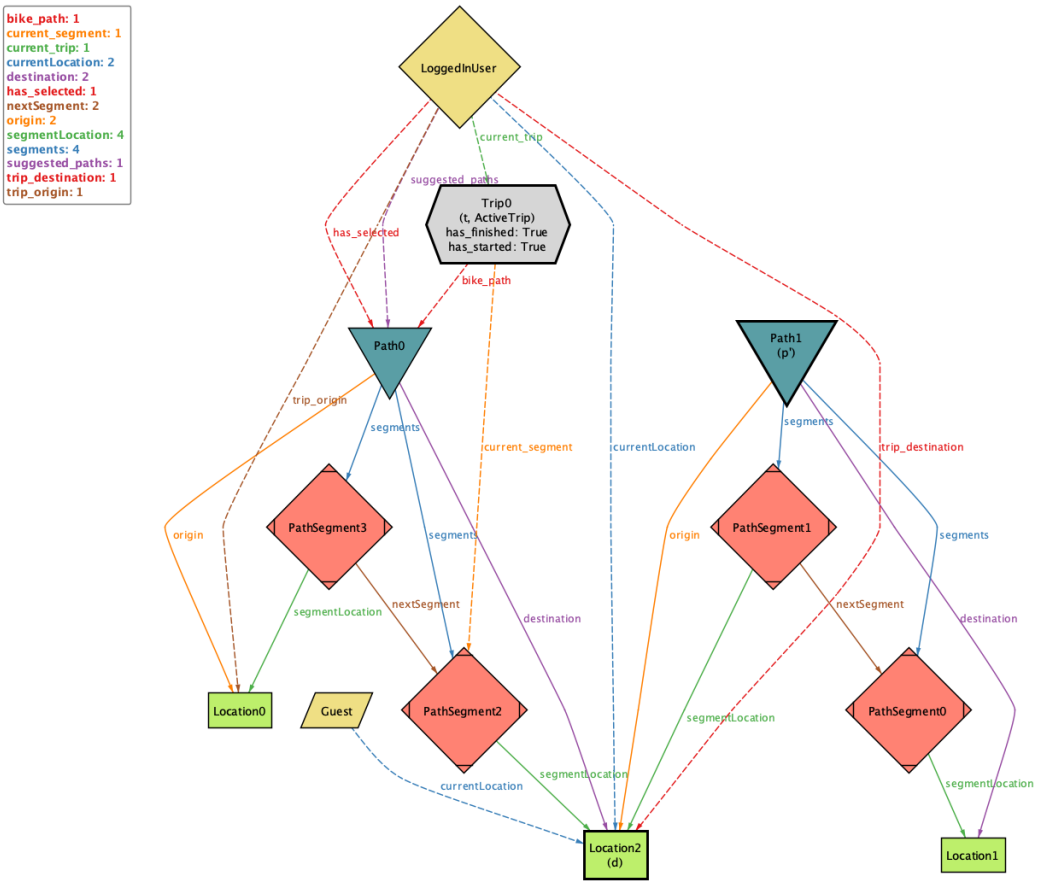

Figure 4.7: User arrives at destination

Then there is another extra step to show the transition of the trip from ActiveTrip to CompletedTrip, as depicted in Figure 4.8. 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.

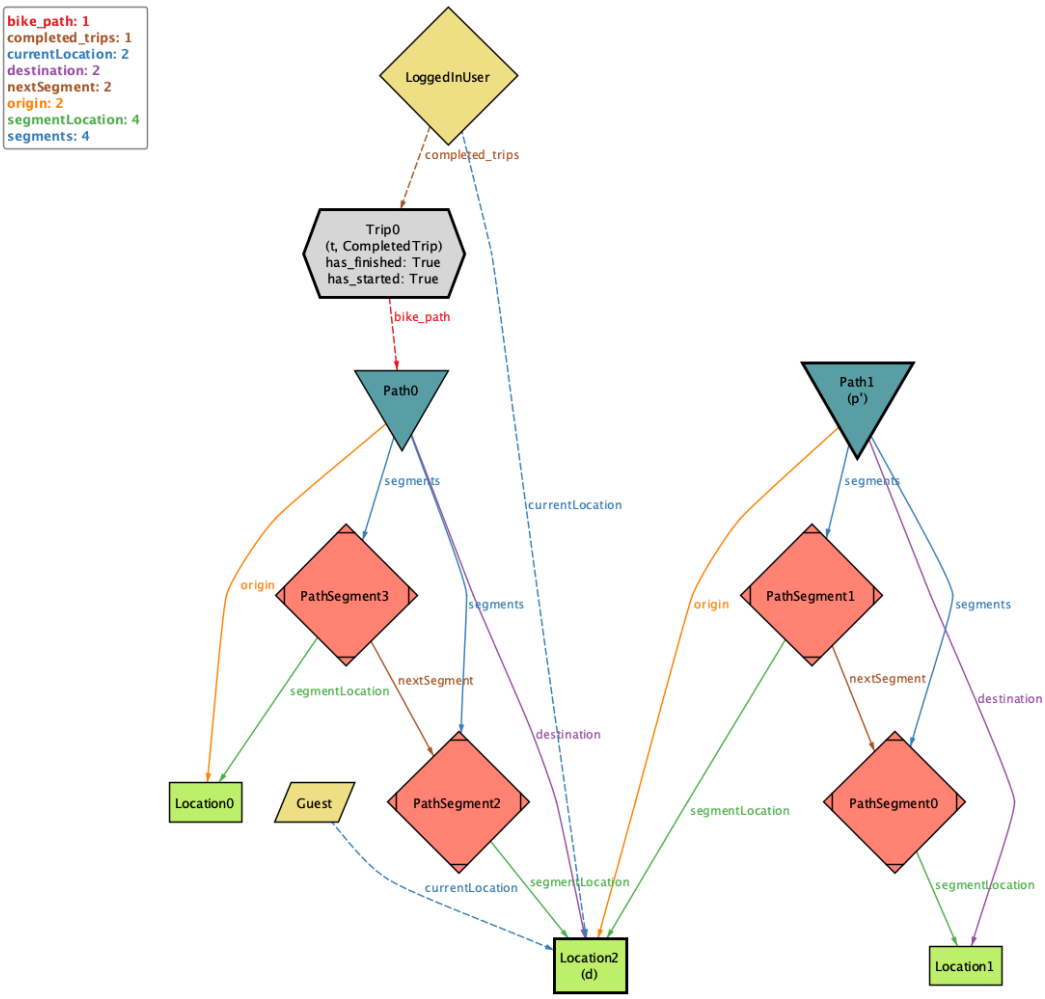

Figure 4.8: Transition from ActiveTrip to CompletedTrip

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 4.11.

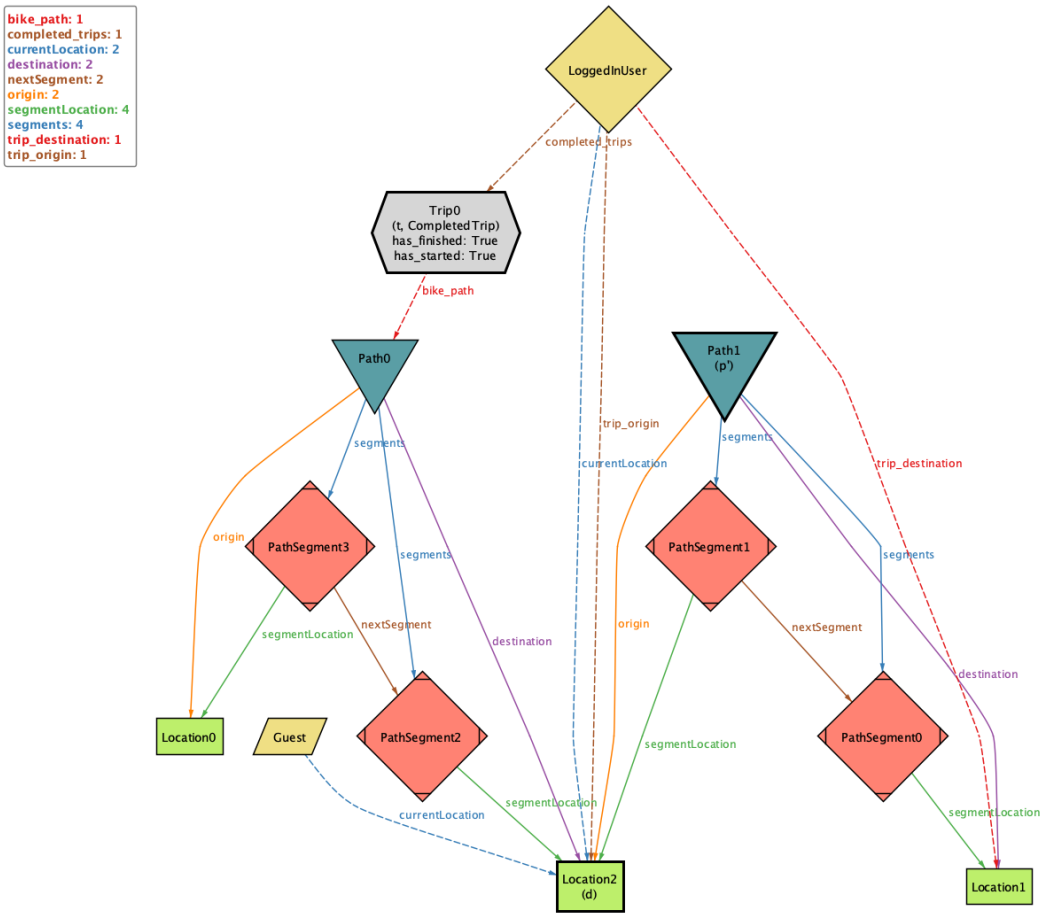

Figure 4.9: Second origin/destination selection

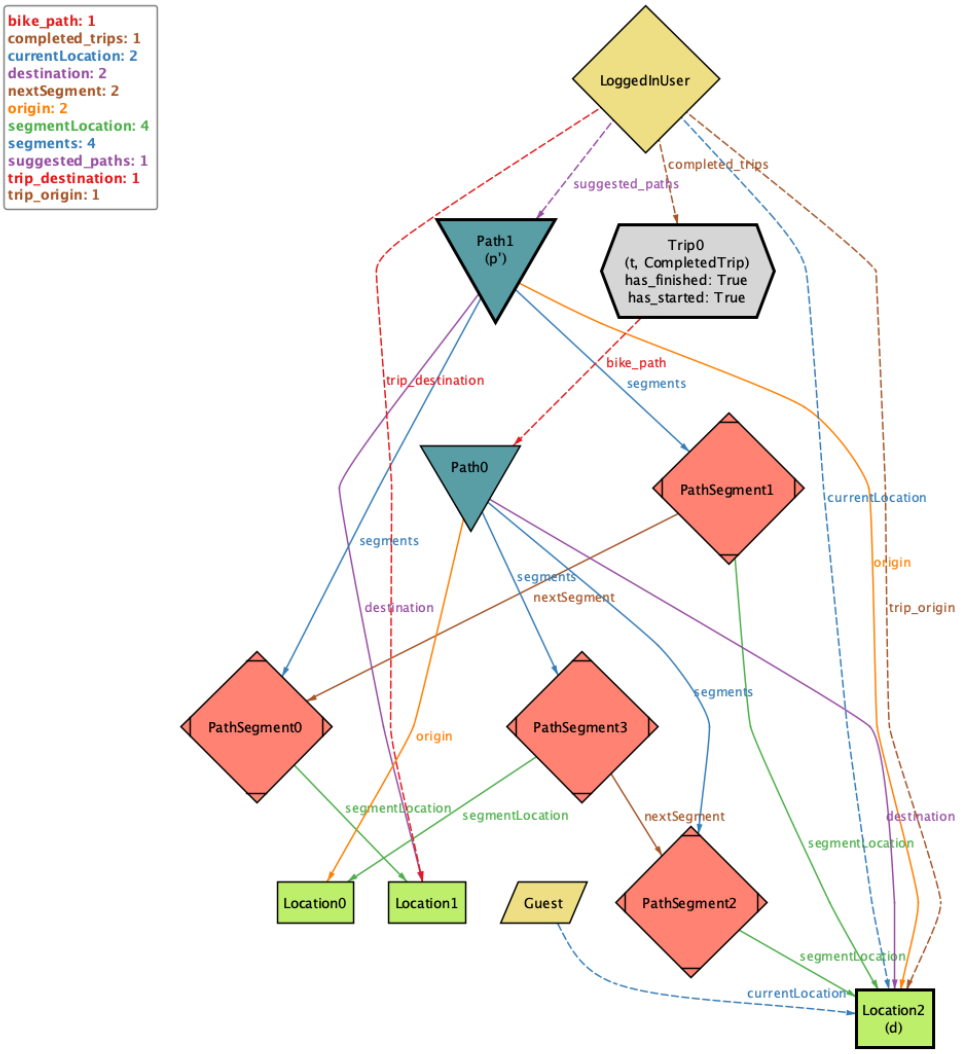

Figure 4.10: Second path suggestion

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

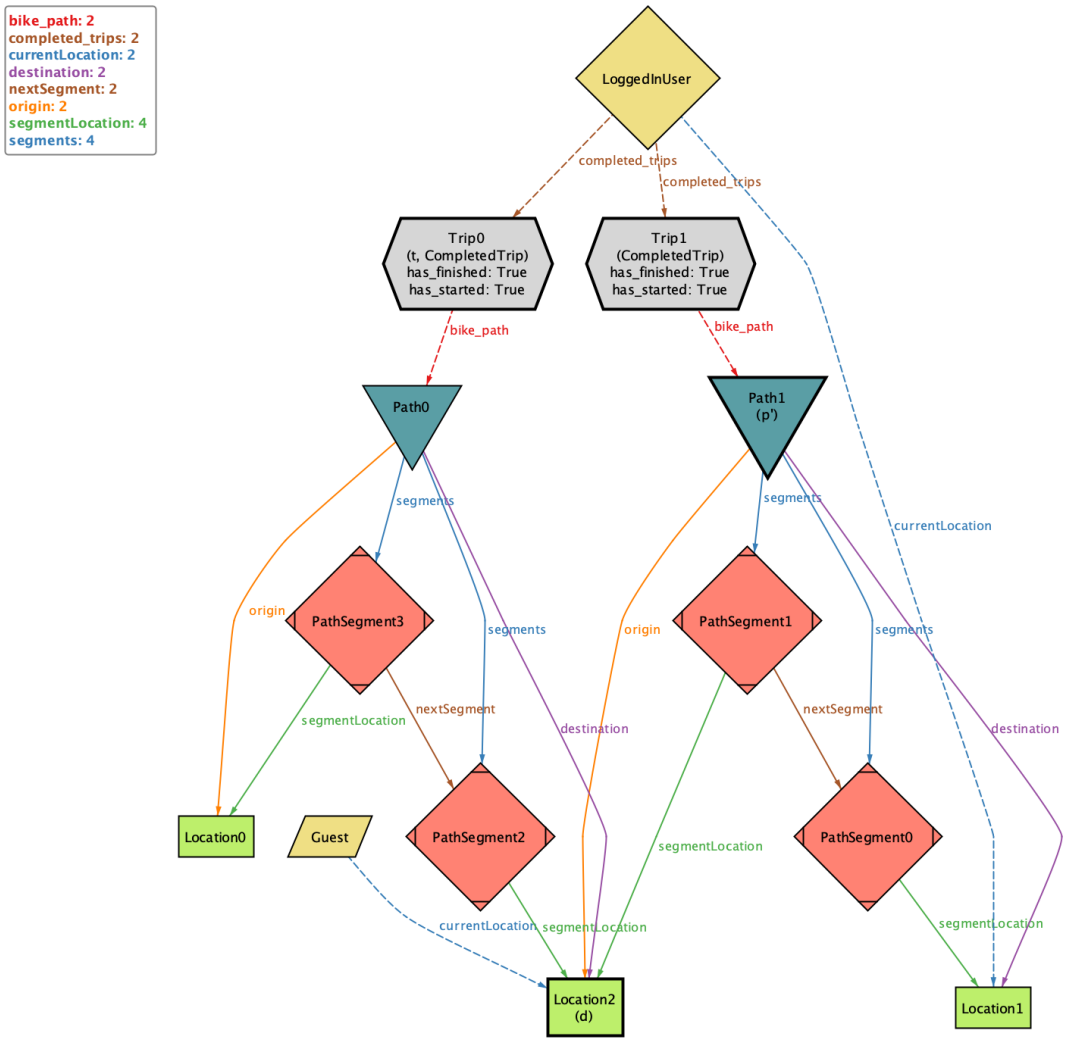

Figure 4.11: Finalization of the second trip

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).

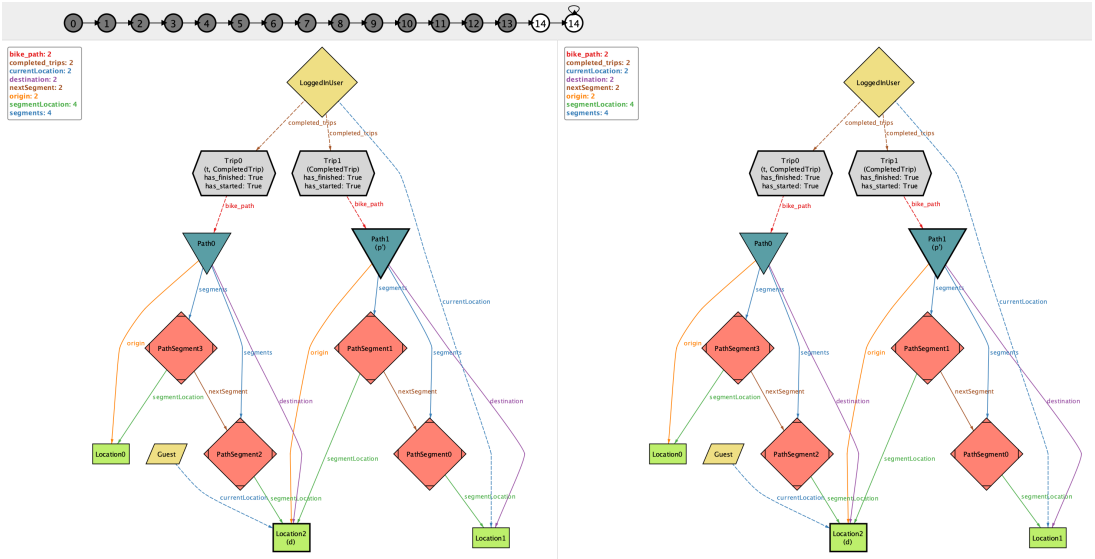

Figure 4.12: Total number of steps produced by the model

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 4.16, 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.

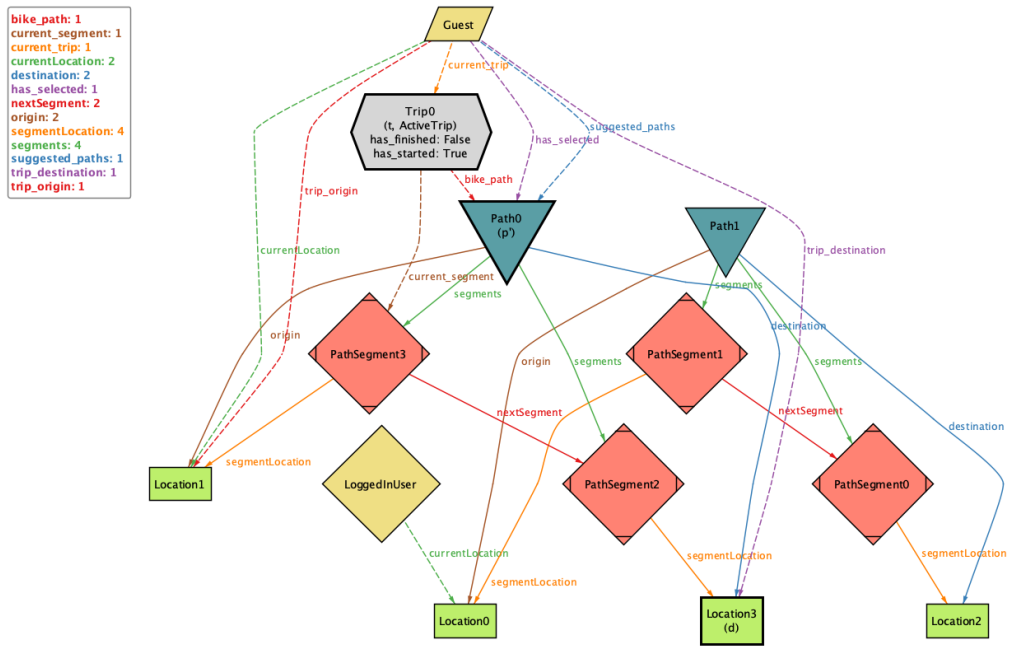

Figure 4.13: User starting the trip

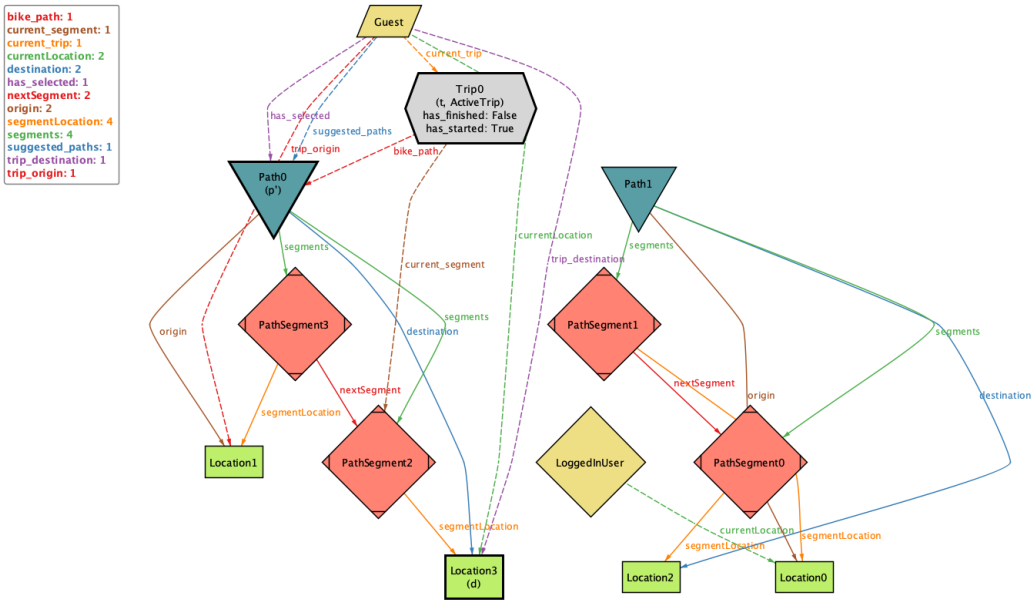

Figure 4.14: User moving to the next segment

Figure 4.15 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`.

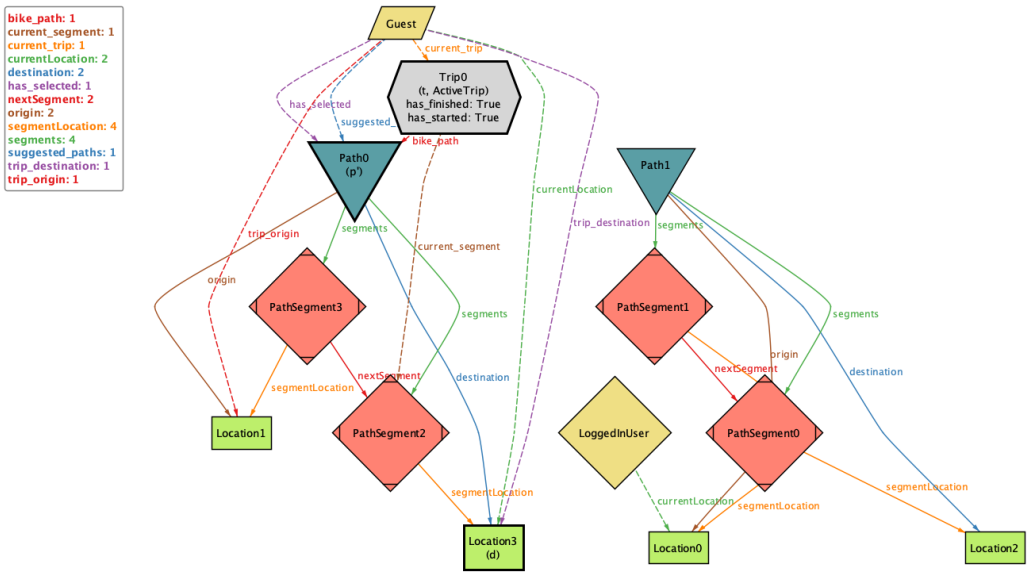

Figure 4.15: User arrives at destination

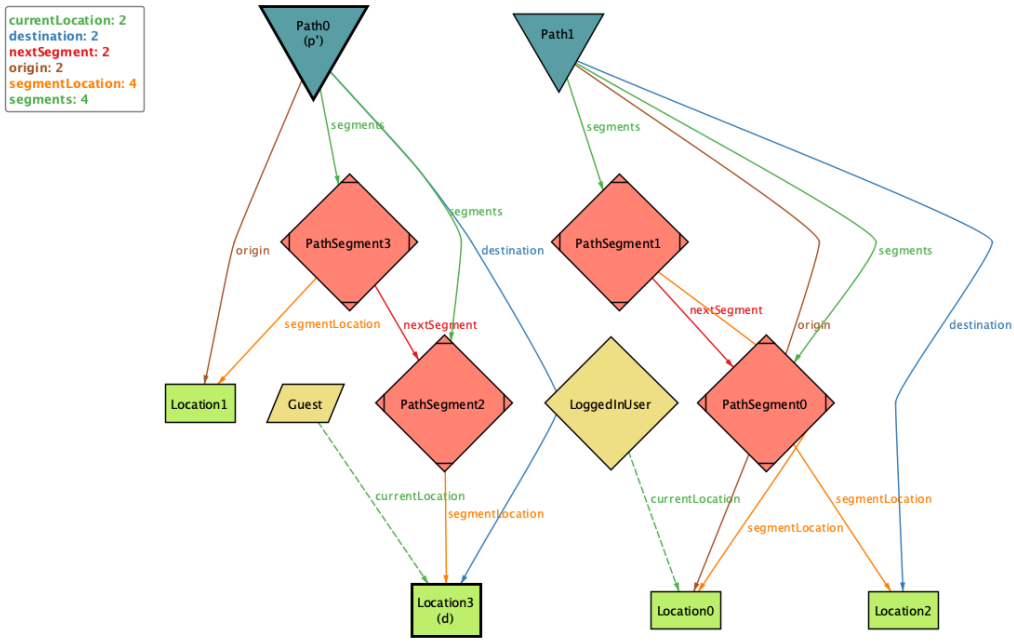

Figure 4.16: State after trip finalization

In the final part of the trace (Figure 4.17), 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.

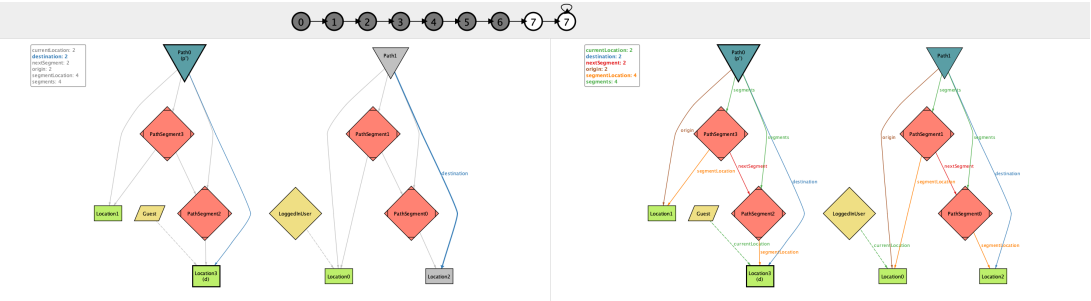

Figure 4.17: Total number of steps produced by the Guest scenario

All assertions defined in the model were checked using scopes consistent with the scenario. No counterexamples were generated (figure 4.18), 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 4.18: Verification of assertions

5 | References

5.1. Reference Documents

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

- IEEE Standard for Software Requirement Specifications [2];
- 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 [8];
- Slides of the Software Engineering 2 course available on WeBeep [9];
- The paper *Deriving Specifications from Requirements: An Example* by Jackson and Zave [3].

5.2. Software Used

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

- **Visual Studio Code**: editing of source code and documentation (LaTeX), with project-wide search and formatting support [7].
- **LaTeX**: typesetting system used to produce the final RASD document in a consistent format [5].
- **Git**: version control used to track changes and support collaborative development [10].
- **GitHub**: remote repository hosting and collaboration platform used for versioning, reviews, and issue tracking [1].
- **Lucidchart**: creation of UML diagrams (use case diagrams, state diagrams, domain class diagram) [6].
- **Alloy** : execution of the Alloy model, trace generation, and assertion checking [4].

5.3. Use of AI Tools

AI tools were part of our workflow in a way similar to other software we used during the project. Rather than producing content autonomously, they helped us reason about how to present certain sections, reorganize ideas, and make the document more coherent.

We used AI mainly while drafting the text, for example to compare alternative ways of explaining scenarios, reorganize long paragraphs, or check whether the wording of some requirements could lead to misinterpretations. In several cases, discussing a passage with an AI assistant helped us clarify our own understanding of the underlying concept before writing the final version.

5.3.1. Tools Used

The AI tools employed during the project were:

- Gemini
- ChatGPT

5.3.2. Typical Prompts

AI tools were queried using prompts such as:

- "Suggest a clearer way to express this requirement without changing its meaning."
- "Can this sentence be misinterpreted? If yes, propose an alternative phrasing."
- "Provide a more concise version of this paragraph while keeping the content intact."
- "Format the following text in LaTeX"
- "Write the basic layout of a LaTeX table with a header row."
- "Help debug formatting or build issues related to VS Code or LaTeX"
- "Apply the Alloy LaTeX style to this code snippet."
- "Given the predicate: predadd[b, bpost: Book, n: Name, a: Addr] { bpost.addr= b.addr+ n-> a }. What is the meaning of "->"?"
- "What is the difference between doing b.addr instead of addr[b]"

5.3.3. Input Provided

The input given to AI tools consisted mainly of:

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

5.3.4. Constraints Applied

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

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

5.3.5. Outputs Obtained

The interaction with AI tools resulted in:

- Clearer or more concise formulations of existing statements.
- Identification of potentially ambiguous sentences.
- Terminology suggestions to improve internal coherence.
- LaTeX formatting assistance for tables and code snippets.

5.3.6. Refinement Process

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

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

6 | 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	5 hours	5 hours	5 hours	15 hours
Total Hours	**46 hours**	**45 hours**	**43 hours**	**134 hours**

Table 6.1: Time spent on document preparation

Bibliography

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

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

[3] 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.

[4] Julien Brunel, David Chemouil, Alcino Cunha, Nuno Macedo. Alloy 6 documentation. Online documentation, 2021. https://haslab.github.io/formal-software-design/.

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

[6] Lucid Software Inc. Lucidchart: Diagramming and visualization tool. Online platform, 2025. https://www.lucidchart.com/.

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

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

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

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

List of Figures

List of Tables