TrackIN: Development of a Tracking App and Analysis of Altitude Data

Technische Hochschule Ingolstadt
Project Coordinator: Prof. Dr. Robert Gold

Summer Semester 2025

Contents

1 Project Overview

This project aims to develop a cross-platform tracking application. The app records satellite-based GPS data, with special focus on the algorithm and accuracy of tracking. The application is designed to run on Android initially. The team later decided to include support for iOS devices as well. Full development has been done initially in Flutter, and Java version has developed later on as simplicity.

Features

- Record GPS + Altitude data
- View and analyze GPX files
- Notification system
- Modular UI
- Altitude correction using smoothing algorithm
- Share GPX files

2 Installation

To install and run the TrackIN app, follow these steps:

Requirements

- Flutter SDK (version X.X.X or higher)
- Android Studio or Xcode (for Android/iOS development)
- Git

Steps

1. Clone the repository:

   ```
   git clone https://github.com/CAIProj/Frontend.git
   ```

2. Navigate to the project folder:

   ```
   cd Frontend
   ```

3. Install dependencies:

   ```
   flutter pub get
   ```

4. Run the app on an emulator or physical device:

   ```
   flutter run
   ```

Note

Make sure your development environment is configured correctly (see official Flutter docs at `https://docs.flutter.dev`).

3 Project Management

- Weekly meetings every Thursday at 14:55 (Room K013)
- Tasks assigned to each member
- Working hours tracked via Google Sheets
- Communication in Discord/Microsoft Teams

4 Licensing

This project uses the MIT License. All code is open source.

5 Notable Events

- **03.04.2025** – First GPX data recorded around campus
- **05.06.2025** – Project Excursion

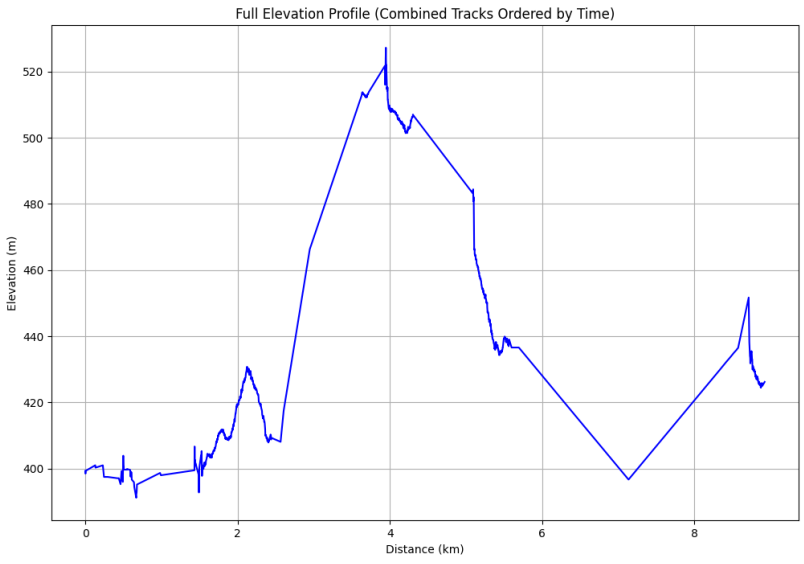

Figure 1: First GPX file recorded by our app

6 Project Members

- Robin Roßnagel
- Thomas Williams
- Himanka Ashan
- Susheel Kumar
- Prashil Rupapara
- Isaac Ye
- Pamirbek Almazbekov
- Baatarbileg Erkhembayar
- Syed Aayan Ahmed

7 Tasks and Responsibilities (Summary Table Format)

Member	Task ID	Title	Description & Result
Robin Roßnagel	T4	Communication & Tracking	Discord used for team communication; Excel for working hours.
	T14	Android Dev Research	Android setup faced team device constraints; iOS prioritized.
	T15	GPS & Altimeter App	Flutter app developed; iOS tested; Android pending.
Syed Aayan Ahmed	T4	Communication & Tracking	Same as Robin.
	T14	Android Dev Research	Same as Robin.
	T15	GPS & Altimeter App	Same as Robin.
Thomas Williams	T1	Open Source License	Researched options; MIT License selected.
	T4	Communication & Tracking	Set up Discord and Excel.
	T6	Altitude DB Comparison	Chose Open-Elevation; Python script written.
	T18a	Route Comparison	Investigated synchronizing GPX recordings.
	T24	Modularization	Refactored and split Python scripts into modules.
Baatarbileg Erkhembayar	T2	Git Options	Chose GitHub after university option failed.
	T4	Time Sheet Improvements	Auto-sum working hours in Excel.
	T5	Altitude Reference Systems	Analyzed GPS and altimeter methods (Jupyter notebook).
	T12	Git Setup	Created GitHub repos for backend, frontend, docs.
	T17/T18	Curve Smoothing	Loess-v2 selected as best method (Jupyter notebook).
	T28	Frontend Testing	iOS tested using Xcode and MacBook.
	T28+	iOS testing	Code coverage testing
	T30	Documentation	Structured this documentation.
	T30+	GitHub structure	Improved the structure.
Himanka Ashan	T2	Git Options	Chose GitHub after university option failed.
	T4	Time Sheet Improvements	Auto-sum working hours in Excel.
	T5	Altitude Reference Systems	Analyzed GPS and altimeter methods (Jupyter notebook).

Member	Task ID	Title	Description & Result
	T12	Git Setup	Created GitHub repos for backend, frontend, docs.
	T17/T18	Curve Smoothing	Loess v1 , Loess v2, Spline fit.
	Task	Curve Smoothing Evaluation	Loess-v2 selected as best method
	T26	Backend Framework	Backend server implemented with FastAPI
	Task	Android studio java	Implemented a tracking application in android studio from scratch.
Susheel Kumar	T3	Git Structure	Suggested initial repo structure.
	T6	Altitude DBs	Researched databases: SRTM, ASTER, Copernicus, etc.
	T16	Use Case Diagram for Tracking App	App's operational workflow.
	T23	Modules of app	Modularization of app.
	T23	Class diagram for Tracking App	Created class diagram for the tracking app.
	T27	Backend Testing	Test plan created
	Task	Backend Testing	3 tests (plotter.py, models.py, evelation_api.py) documented
Prashil Rupapara	T3	Git Structure	Suggested initial repo structure.
	T6	Altitude DBs	Researched databases: SRTM, ASTER, Copernicus, etc.
	T16	Use Case Diagram for Tracking App	App's operational workflow.
	T23	Modules of app	Modularization of app.
	T23	Class diagram for Tracking App	Created class diagram for the tracking app.
	T24	Unifying scripts	Unified the scripts of the backend and update it if needed.
	Task	Pull request merging	Merged pull requests for backend repo.
	Task	Final Unification	Did final unification needed after addition of synchronised plotting.
	Task	Update plotting with main	Added functionality to have multipe plots(with any of the api and smoothing method) with CLI usage.
	Task	Python dependency management	Made possible to have easy ython setup locally with just few commands in terminal.
	Task	Create class diagram	Created class diagram with the relationship in between for backend.
	Task	Update README.md	Updated and added needed information and links in README.md file.

Member	Task ID	Title	Description & Result
Isaac Ye	T8	GPX file parsing logic	Wrote parsing logic, distance vs elevation graph.
	T22	UI prototypes for use cases	Created prototypes according to T16.
	T25	App Review& Fixes	Testing, reviewing, giving feedback to the app
	Task	Andriod fixes	Changed a dependency not supported on Android.
	Task	Flutter app	Modularized codebase and added more pages
	Task	Flutter app	Added notification system, initial UI changes and fixes.
	Task	Flutter app	Business logic fixes and remove deprecated status state. Mostly finalized UI change and fixes.
	Task	Flutter app	Changes to interact with the Framework backend.
	Task	Flutter app	Improved local-upload GPX file mapping and optimized UI.
Pamirbek Almazbekov	T8	GPX file parsing logic	Wrote parsing logic, distance vs elevation graph.
	T22	UI prototypes for use cases	Linking to UML diagram from T16.
	T25	App Review& Fixes	Testing, reviewing, giving feedback to the app
	T29	Technical Docs	Wrote technical documentation.
	Task	Android studio java	Implemented a tracking application in android studio from scratch.

Detailed task summary

Isaac Ye T8: Wrote a Python helper file that included GPX file parsing and graph plotting.

T22: Following the UML diagram in T16, we created UI prototypes for the Flutter application using Figma for multiple pages, such as the home page (idle state), home page (recording state), a page displaying all local GPX files, and a page for displaying detailed information for a selected GPX file. These prototypes are summarized with their respective use cases in a use case document. Some changes were made to UML diagram as well to include missing use cases.

T25: Tested and reviewed the Flutter application on Android (emulated) and Chrome and provided feedback.

Android Fixes: Changed a dependency that was not supported on Android, which made the Flutter application work on the Android platform.

Flutter App: Reorganized the codebase from a singular main file to be more modular, added two new pages (one for displaying all recorded tracks, one that displayed information about individual tracks), and started to change the UI to be closer to the UI prototypes.

Flutter App: Add new notification system for displaying messages, 'Are you sure?' dialogue when deleting a track, fixed filepicker dependency on Android when importing a GPX file, and implement initial home UI changes.

Flutter App: Fixes to stopwatch logic, moved legacy status messages to notifications instead, and disallow recording UI to be shown if location permission is denied.

Flutter App: Fixed display of statistics and convert timestamps of GPX files from UTC to local time, worked to finalizing the styling of the home page, tracks page and track page.

Flutter App: Removed point limit for a recording and allowed tracking while not in the application (only on Android, this feature for iOS would require a Apple Dev account, which requires a fee of €100 per year). Changed GPS to stream instead of querying for the location and changed the GPS query interval to be less frequent. Limited the amount of points rendered for the widget displaying detailed point information (since we could have infinite points).

Flutter App: Added a client for interacting with the Framework to upload GPX files. Added an account page (to login and register to interact with the Framework). Added buttons to upload / delete files to / from the Framework, which only allows the user to do so if logged in. Attempted to map local files to uploaded files (to ensure we don't upload twice and can delete them). The approach used would fail in the case where the user switches accounts, however. More UI fixes and cleaning up.

Flutter App: Instead of managing a local file that holds the mapping of local to uploaded files, I opted to download the remote files on login and use hashing to compare the contents of GPX files (which fixes the case of switching accounts). Loading the tracks page now doesn't try to spawn too many threads that it hangs the render thread. Added header text to statistics for clarity.

Himanka Ashan Research Git options. Result: GitHub used due to lack of university contact

Research altitude reference systems and techniques Result: Comprehensive explanation of altitude tracking systems, including the reasoning on the difference between actual elevation and elevation from apis

Set up Git Results: Created and managed repositories in github

Curve smoothing algorithms Results: implemented custom smoothing algorithms. Namely, Loess v1 , Loess v2, Spline fit. All algorithms are implemented in a way that they can be imported easily.

Evaluate smoothing algorithms Results: Applied mean error and mean squared error to evaluate the smoothing algorithms. Loess v2 was proven to be better from the evaluation report.

Framework Results: Backend server implemented with FastAPI with following methods

- POST /register : Register a user

- POST /login : Login method POST /upload : Upload a gpx file to the database

- GET /files/ : Return all files from a user

- GET /files/fileid : Return Specific GPX file and download

- DELETE /files/fileid : Deletes a gpx file from database Hosting the server : Used Render.com's free hosting account to deploy the server on the internet. Special render configuration was needed.

Android Studio Java Application (2-weeks) Results: Implemented a tracking application in android studio from scratch. - Introduced Altitude measurement to the app and adapted the database and the receiver

- Dynamic START/STOP button to start tracking

- App Structuring: Introduced Fragments (TrackFragment, DashboardFragment)

- Added Accumulating time and distance to the tracking page (Haversine distance)

- Altitude, Lat:long, Distance, Time are displayed in stylish cards to present a modern look

- UI improvements

- Code cleanups

Susheel Kumar, Prashil Rupapara Git research We decided on how our project on Git should be structured and what should be the rules of using git, the result was rules and directory structure of git project

Freely accessible databases We identified freely accessible altitude databases and evaluate their accuracy. The findings revealed variations in altitude precision among available resources, with some datasets offering higher resolution for specific regions. Open-source accessibility played a key role in determining the usability of the databases, providing valuable data for geographic and environmental analysis. The comparison underscores the importance of selecting appropriate altitude datasets based on accuracy requirements and intended applications.

Use case diagram We created a detailed use case diagram was developed for the Android tracking app, outlining complete use cases and their corresponding actions. The diagram specifies interactions between users, system components, and key functionalities, providing a structured representation of the app's operational workflow.

Modules of app We had to determine different modules of the app so that they could be assigned to different development teams, the result was modularization of app and suggestion for tasks which could be assigned to different teams

Susheel Kumar Backend test plan A test plan was developed to evaluate the backend functionality of the app. Key features requiring testing were identified, and appropriate test approaches were determined to ensure comprehensive validation. The plan aimed to enhance reliability, optimize system performance, and address potential issues before deployment.

Prashil Rupapara Extensive contributions were made to the backend structure, CLI plotting functionality, environment setup, and project documentation, with a focus on maintainability, usability, and modular design.

Various backend Python scripts—including those for GPX parsing, elevation API integration, smoothing, and plotting—were unified into a modular and logically structured architecture. Final refactoring was performed after the addition of synchronized plotting capabilities to maintain a coherent and testable codebase.

The command-line interface (`main.py`) was extended to support multiple elevation sources and smoothing techniques such as `--add-openelevation` and `--add-loess1`. For synchronized plotting, validation logic was implemented to ensure only one comparison source could be used at a time, while non-synced plotting supports multiple sources simultaneously. These enhancements provided a flexible and user-friendly CLI experience.

Python dependency management was handled using `Poetry`. Both `pyproject.toml` and `requirements.txt` files were added to support different setup preferences. The environment setup was tested and verified on Windows 11 machine.

UML class diagrams for backend components were created independently using Mermaid syntax. Key classes such as `Track`, `ElevationProfile`, `OpenStreetMapElevationAPI`, and `Plotter` were included, along with their internal relationships. These diagrams served as a useful reference for both implementation and documentation.

The `README.md` file was updated to include local development setup instructions, links to example Jupyter notebooks, and embedded references to backend class diagram files. These documentation efforts enhanced onboarding and usability for future contributors.

Tom Williams Deciding which License to use. Under the directive that the license must be open source, I looked into the different kinds of open source license available to use. I wrote a summary of the different kinds, which generally came under the categories of "Permissive" and "Copyleft", and their advantages and disadvantages. I presented my findings to the team, giving my thoughts on which license we should use, and we all agreed to use the MIT License.

Research freely accessible altitude databases I looked into which altitude databases we could use to compare our recorded data with to measure the accuracy. I found Open-Elevation, OpenTopography, NASA Earth data, and Google Earth engine. I found that Open-Elevation was the best to use as the API was easy to use and it returned numerical data from the request that we could use. Whereas OpenTopography for example, would only return an image representing topographical data which would have required a lot more work to implement and risked not working at all. The other APIs either didn't offer the geographical region I was working with, or provided identical data to Open-Elevation, so I decided we should only use Open-Elevation and I wrote a script that would plot the elevations from the GPX recording alongside the elevations provided by the API.

Modularisation of the python scripts As I had done the GPX file processing, plotting, API calls and command line parsing all in a single file, I worked to separate the functionality into separate files to keep the code clean and easy to read.

Improving plot functionality As part of my plotting programme, I tried to find a way to plot two different gpx files of the same, or a similar route, in a way that made sense. I

tried to implement a strategy of having a primary plot and a secondary plot, where the secondary plot would only show on the graph when it was within a certain distance of the primary plot, and leave gaps where the two plots diverged, so the elevations would represent the same geographical spot. But this turned out to be incredibly difficult, and after discussing with the team, we decided that we would leave this idea for now, try to implement a simpler way of plotting two routes for comparison, and if time permits, try to solve this strategy later.

Baatarbileg Erkhembayar

Git Options We considered using the University's GitLab instance, but due to unsuccessful contact with Prof. Dr. Sebastian Apel, we opted for GitHub.

Working Hours Excel: A simple summary table was created for automatic calculation of working hours.

Reference Systems Several reference systems were identified, introduced, and evaluated using a Jupyter notebook.

GitHub https://github.com/CAIProj – Three repositories were created: *Backend*, *Frontend*, and *Docs*. All findings and documentation are stored in the *Docs* repository.

Curve Smoothing: `curve_smoothing_algo.ipynb` was developed to compare different curve smoothing algorithms. The *Loess-v2* algorithm performed best.

Frontend Testing Plan The task was similar to T25. Simulator testing was successful; however, testing on a real iOS device was not possible due to incomplete app state requiring debugging. After the modification with the real phone connection, using Apple developer tools, I downloaded the app locally and tested with test cases. More appear in Testing section

Project Documentation This document serves as the official project documentation. Project is written in Latex, and uploaded into Github with supporting photos as a zip file.

Frontend iOS Testing iOS frontend was tested using a MacBook and Xcode. Test cases were created, and code coverage analysis was performed. Code coverage only had 12 percent, which is due to having less test cases in widget.test app.

Syed Aayan Ahmed, Robin Roßnagel During the implementation of our cross-platform tracking application, we developed a modular and extensible system for recording, processing, and comparing geospatial data, specifically targeting GNSS-derived altitude values. The app was built using Flutter to ensure native performance on both Android and iOS, with backend logic primarily written in Dart and integration of native plugins where required.

We implemented a persistent background tracking service using Dart Isolates incombination with the flutterforegroundtask package. This design allowed the app to collect GPS coordinates and altitude data (from GNSS) at 5-second intervals, even when running in the background or with the screen off. The service communicated with the main isolate using a ReceivePort, ensuring asynchronous, thread-safe data flow. Each recorded point was time-stamped and included latitude, longitude, and altitude. The location stream

was sourced via the geolocator package, with accuracy settings tuned to LocationAccuracy.best for maximum vertical resolution

1. Track Alignment Function: We implemented aligntrackendpoints, which processes two GPX tracks and truncates them such that both start and end at approximately the same spatial positions. The algorithm performs a bi-directional search within the first and last 20of each track to identify the closest start and end pairs within a configurable threshold (default: 100 meters). These indices are then used to slice the point arrays, resulting in aligned sub-tracks suitable for direct comparison. Robust error handling was added to manage edge cases such as non-overlapping routes, insufficient point density, or excessive drift.

2. Track Interpolation Function: We further implemented interpolatetomatchpoints, an algorithm that resamples the second track to ensure it has the same number of points as the first. This is accomplished using cumulative arc-length interpolation based on Haversine distance. The resampled track maintains spatial continuity and allows for pointwise analysis (e.g., elevation delta at fixed intervals), which is crucial for evaluating vertical accuracy and identifying anomalies.

In parallel, we addressed several practical implementation challenges. The Android-specific foreground service required careful lifecycle management to ensure it remained active under aggressive power management policies. We also resolved early-stage issues related to file handling and serialization of GPX data, particularly on Android 13+, where scoped storage restrictions affected write permissions. We migrated file writing to use pathprovider and ensured compliance with platform-specific permissions. The GPS

sampling interval and filter thresholds were empirically tuned to balance temporal resolution, power consumption, and altitude stability. To verify the correctness and reliability of our implementation, we developed a comprehensive test suite comprising over eight unit and integration tests. These covered a range of scenarios: identical tracks, varying sampling densities, partial overlaps, route drift, noise-injected altitude profiles, and real-world walking/cycling recordings. All tests were successfully validated, and debug visualizations were used to qualitatively assess alignment and interpolation performance. The resulting application and backend form a complete open-source framework for altitudefocused GPS analysis and provide a solid foundation for the integration of rule-based or AIassisted correction techniques in future work.

8 Use case diagram

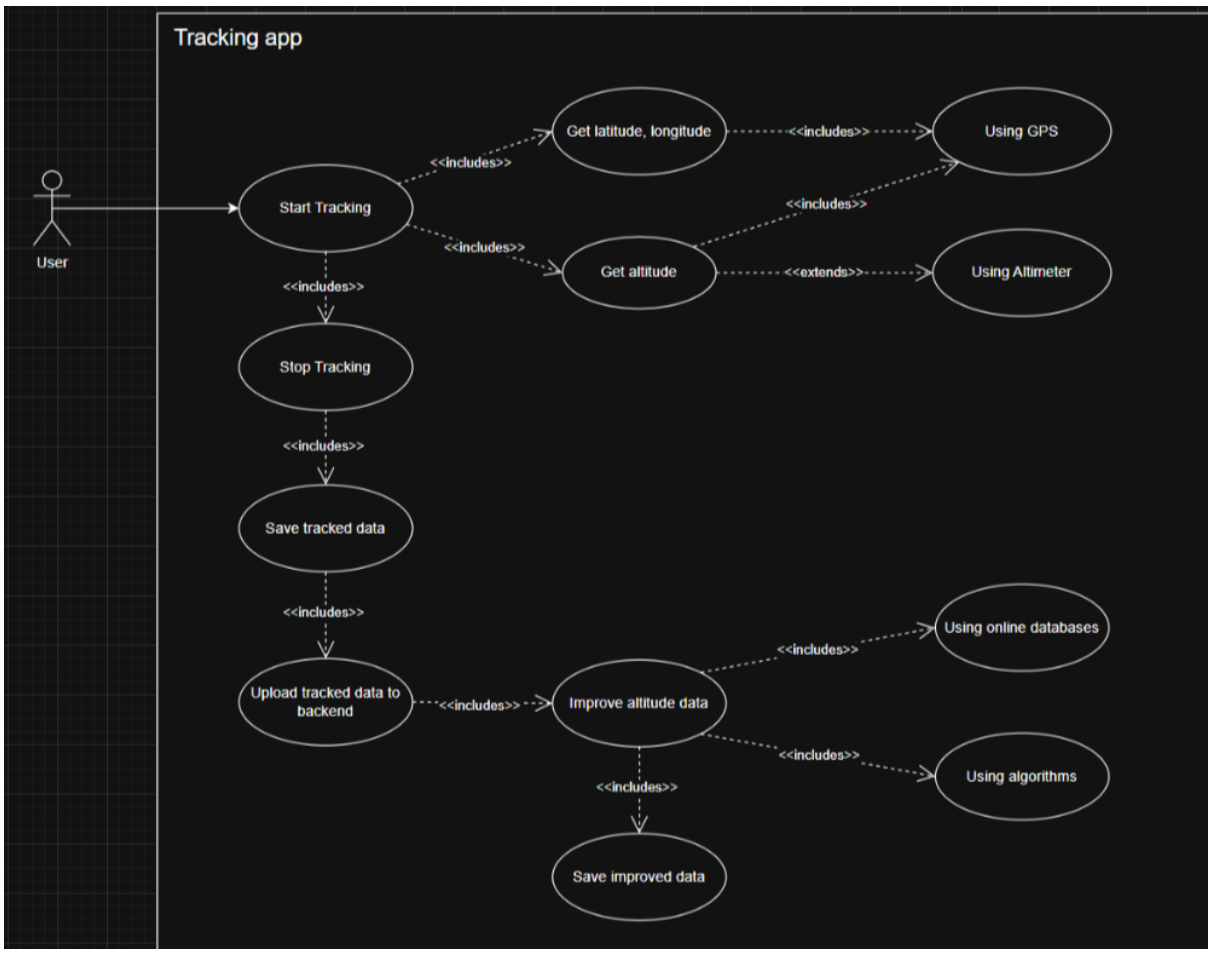

Figure 2: Use case diagram for TrackIN

Field	Description
Use Case	Home Screen Behavior Based on Time of Day
Description	The system displays one of the screens shown upon start-up (home screen), depending on the time of day.
Precondition	The user starts the application.
Sequence	N/A
Postcondition	The user has the choice to start a recording (tap and hold) or view any previous recordings (swipe left).
Comments	Colours here aren't final and just suggestions for morning / afternoon / evening. It's also possible to just have a grayscaled theme (see below use cases). The "swipe left to see recordings" could be replaced by having an icon in the top right corner.

Table 2: Use Case: Home Screen Behavior

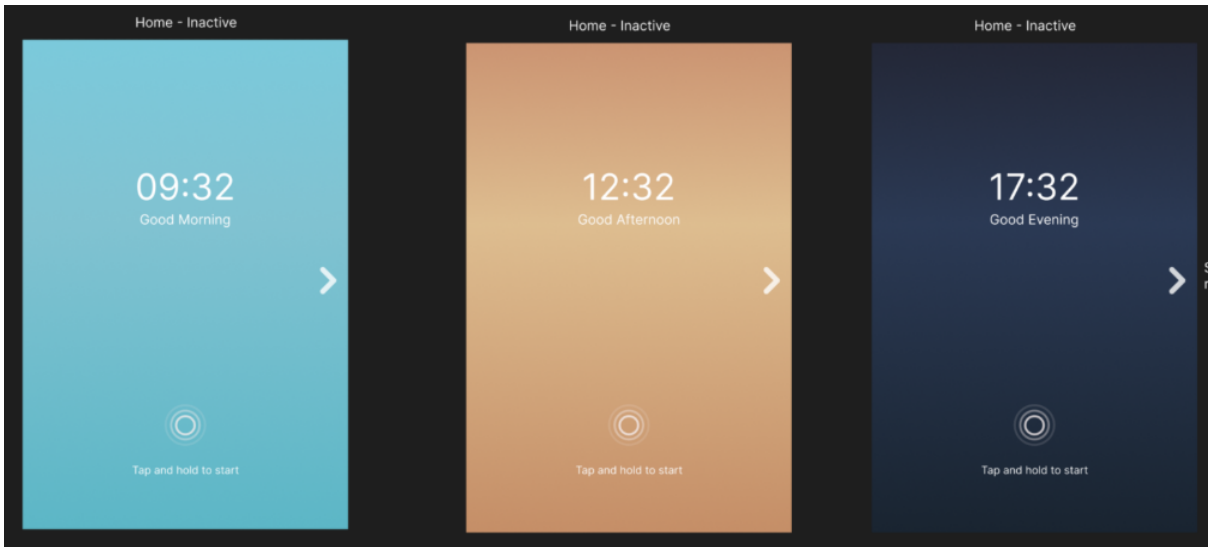

Figure 3: Home screen concepts

Field	**Description**
Use Case	Track Recording
Description	The application records a track.
Precondition	The application is in the idle state (on the home screen) and is not in the recording state.
Sequence	While in the idle state, the user taps and holds the circle shown on the screen (see previous use case). The application starts recording the current position and elevation through GPS (and altimeter if applicable), displays the current time, distance and elevation through text and a graph.
Postcondition	Swiping up results in the termination of the recording, saves the recorded data and returns to the idle home screen.
Comments	The colour here is grayscaled, which can either be in the final design or replaced by the alternating colours shown in the previous use case. The top right icon displays the current GPS signal strength.

Table 3: Use Case: Track Recording

Field	**Description**
Use Case	Saving and Uploading Data
Description	The application saves the recorded track.
Precondition	The application was previously recording.
Sequence	While recording, the user swipes up to terminate the recording and saves the track. The application saves the data locally onto the device. The application uploads the data to the backend.
Postcondition	The user can view the saved data and it is visible on the backend.
Comments	N/A

Table 4: Use Case: Saving and Uploading Data

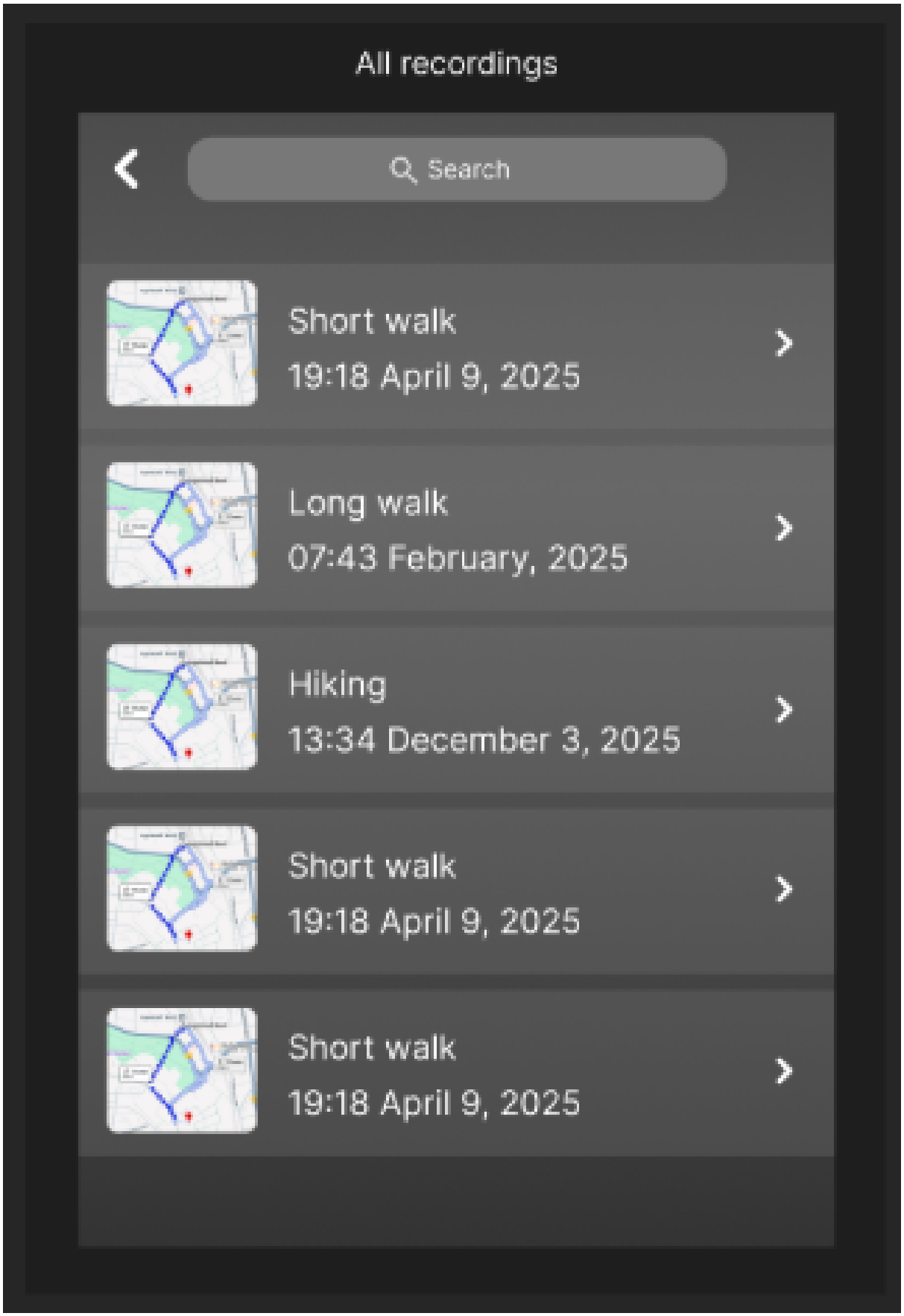

Figure 4: Saving, uploading data

Field	Description
Use Case	Viewing Recordings
Description	The application displays a list of all recordings.
Precondition	The application is in the idle state and is not actively recording a track.
Sequence	While in the idle state, the user swipes towards the left direction.
Postcondition	The user can inspect any of the listed recordings or return to the home screen (back button).
Comments	The colour here is grayscaled, which can either be in the final design or replaced by the alternating colours shown in the first use case. The button on the top left could be replaced with swiping towards the right to return to the home screen.

Table 5: Use Case: Viewing Recordings

Field	Description
Use Case	Viewing Track Details
Description	The application displays more details about a selected recorded track.
Precondition	There exists at least one recorded track.
Sequence	While in the list of all recordings (see previous use case), the user clicks on any of the displayed recordings.
Postcondition	The user has the option to rename, share, delete, or export the track to GPX.
Comments	The 'share' button and 'export to GPX' button could be merged. The presence of an altimeter is shown here as well (the last icon, next to 'Has'). This icon could be changed if it isn't clear enough.

Table 6: Use Case: Viewing Track Details

Field	Description
Use Case	Post-Processing Data
Description	Tracks on the backend are processed.
Precondition	The application client(s) has uploaded tracks to the backend.
Sequence	Upon track upload, the backend uses data from external databases and/or smoothing algorithms to correct the elevation according to the current position of the user.
Postcondition	The data can be analyzed further and visualized through graphs.
Comments	Figure out backend server hosting solution.

Table 7: Use Case: Post-Processing Data

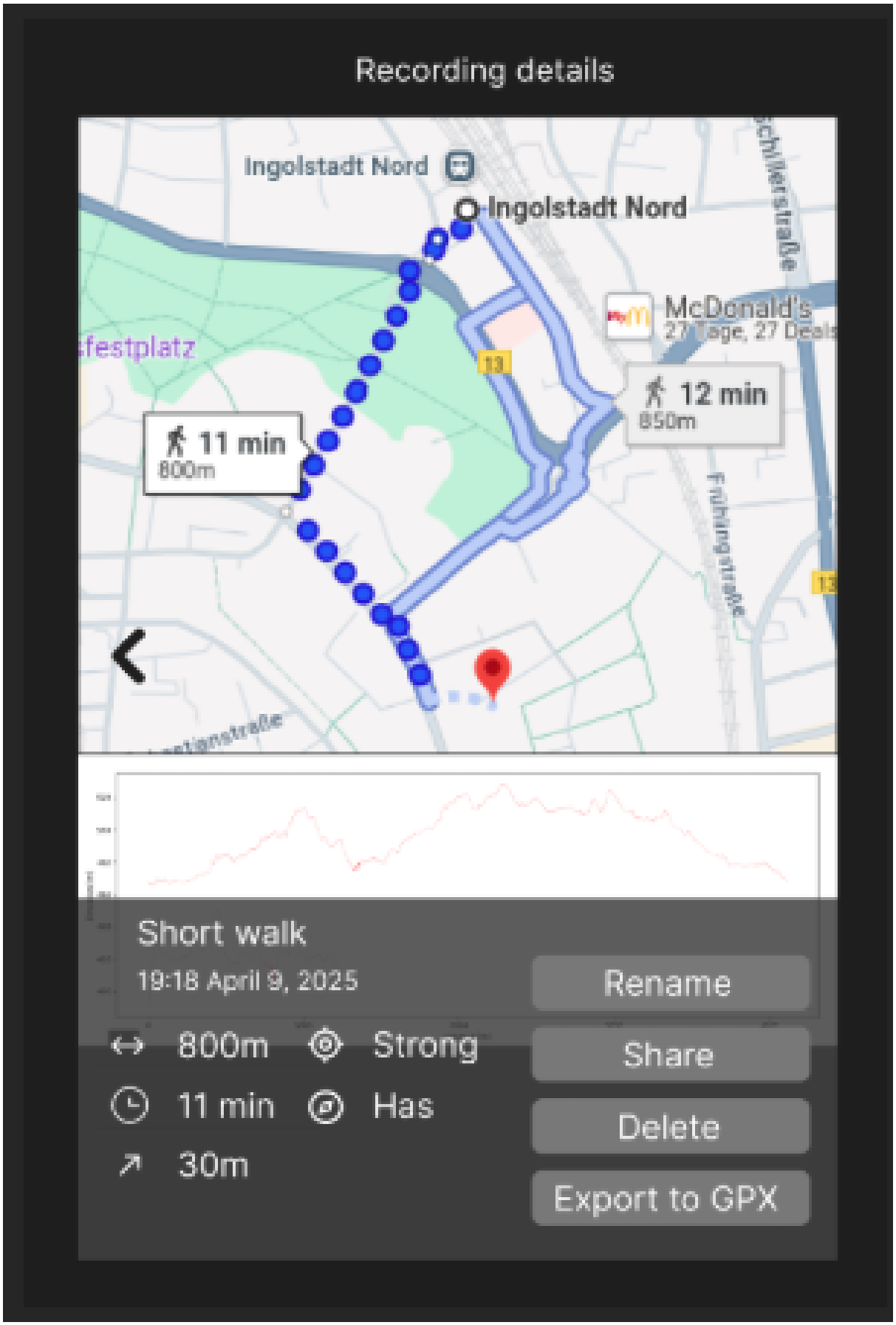

Figure 5: Track details, Post-processing

9 Backend Class Diagrams

The following UML diagrams provide an overview of the core backend components implemented in this project. These class diagrams cover data models, elevation APIs, and plotting logic.

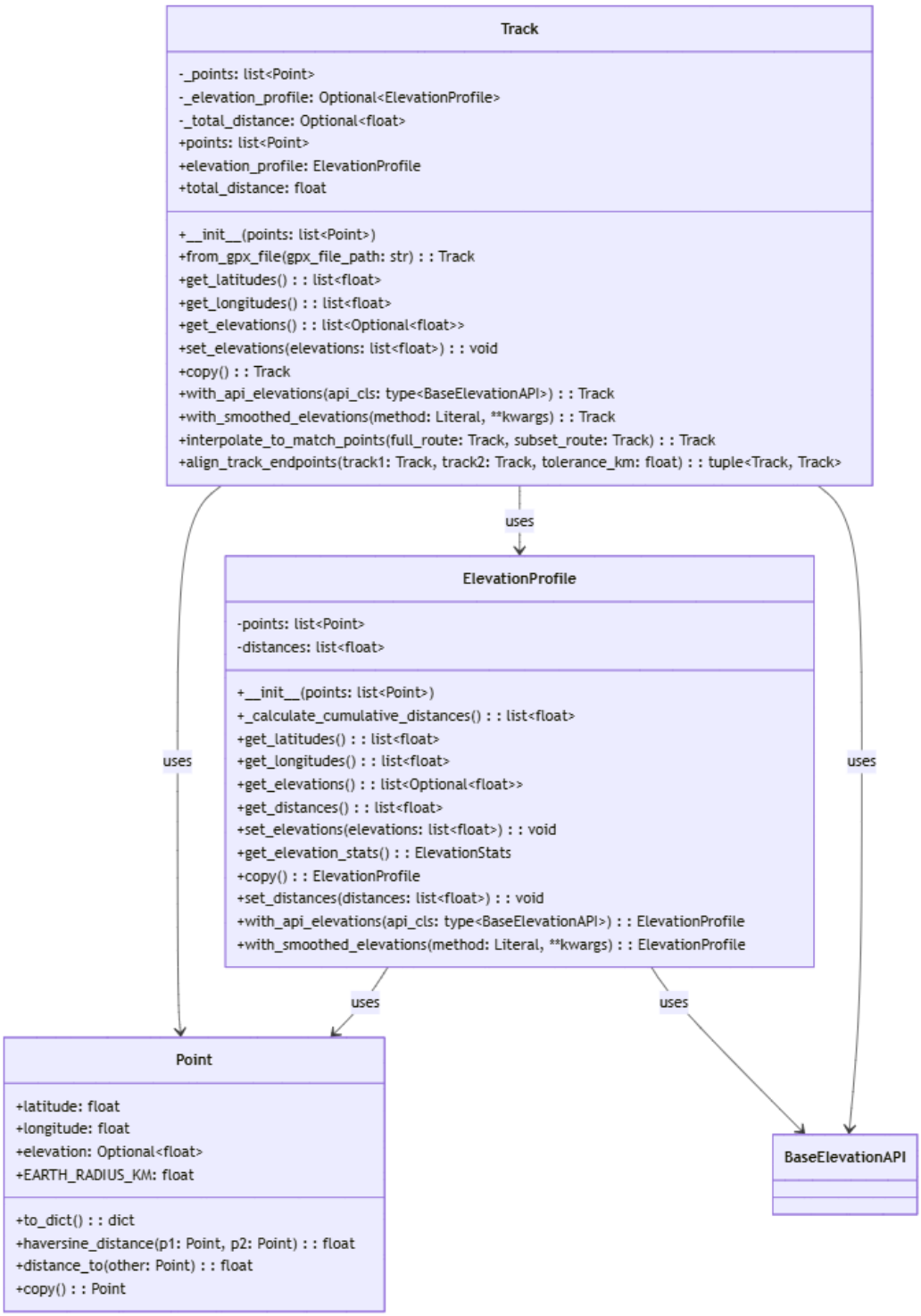

Figure 6: Class diagram of backend data models (`Point`, `Track`, `ElevationProfile`)

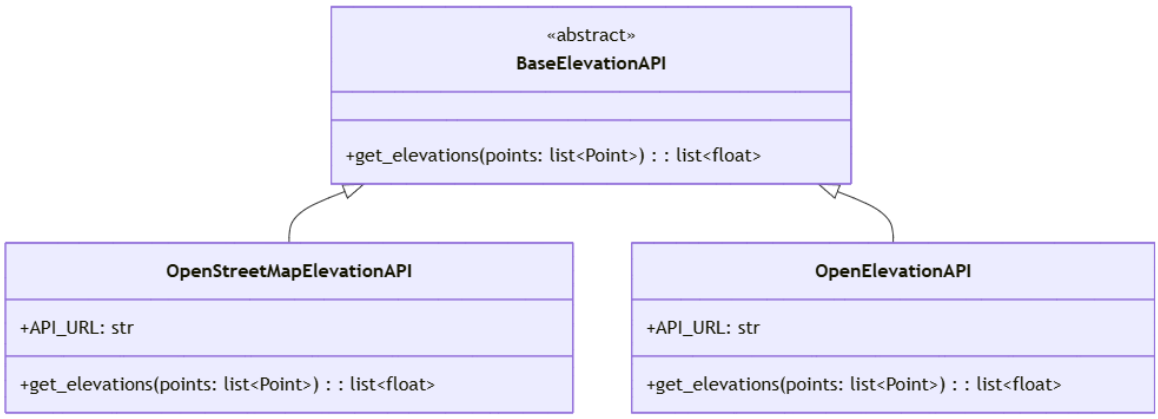

Figure 7: Class diagram of elevation API components (`OpenElevationAPI`, `OpenElevationAPI`, etc.)

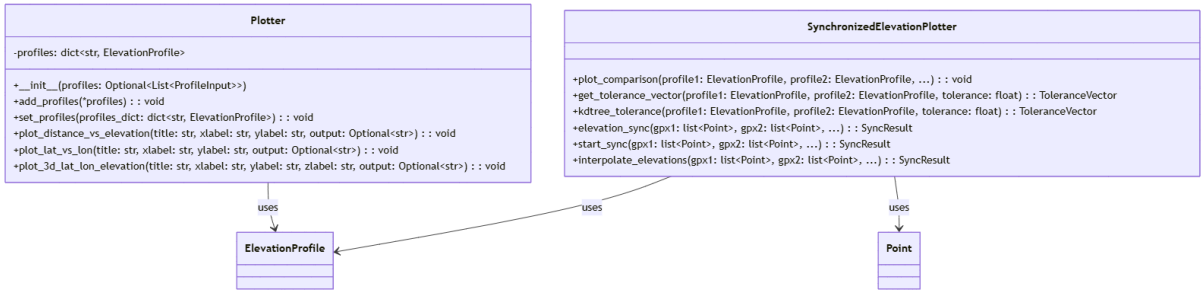

Figure 8: Class diagram of plotting classes (`Plotter`, `SynchronizedElevationPlotter`)

10 Backend Setup and CLI Usage

Backend Setup

To run the backend tools locally, the following steps can be followed for setup:

Steps

1. Clone the repository:

```
git clone https://github.com/CAIProj/backend.git
```

2. Navigate to the project folder:

```
cd backend
```

3. Install dependencies using Poetry:

```
pip install poetry
poetry install --no-root
```

4. Alternatively, install dependencies using pip and requirements.txt:

```
pip install -r requirements.txt
```

Both installation methods have been tested and verified on Windows environments.

Command-Line Interface (CLI) Usage

The backend provides a flexible CLI tool through `main.py`. The general structure of usage is as follows:

```
python main.py [plot type] [base_gpx] [add source(s)]
               [sync method (optional)] [general options]
```

The command consists of multiple components, each configurable through flags:

Plot Type:

- `3d` — Generate a basic 3D elevation plot.
- `elevation` — Plot elevation profiles with optional synchronization.
- `surface` — Generate a 2D surface plot of the elevation profile.

Base GPX: The GPX file path is passed as a positional argument after the plot type. This is the base track used for all comparisons.

Optional Sources (Elevation Data / Smoothing):

- `--second-gpx <path>` — Compare with another GPX file.
- `--add-openelevation` — Add elevation using OpenElevation API.
- `--add-openstreetmap` — Add elevation from OpenStreetMap.

- `--add-loess1`, `--add-loess2`, `--add-spline` — Apply elevation smoothing.

Synchronization (elevation plot only):

- `--sync-method <elevation_sync | start_sync | interpolate_elevations>` — Choose sync method.
- `--tolerance <km>` — Set distance tolerance.
- `--tolerance-method <standard | kdtree>` — Select comparison method.

General Options:

- `--output <filename>` — Save the plot instead of displaying it.
- `--title <text>` — Add a custom title to the plot.

A validation is enforced such that for synchronized plots, only one additional elevation source is allowed at a time. In contrast, for non-synced plots, multiple sources can be added simultaneously. These combinations provide a flexible yet constrained structure for diverse elevation comparisons.

Example usage and output are shown in the demo notebook `backend_feature_examples.ipynb` (see `docs/backend_feature_examples.ipynb` in the backend repository).

11 Plotting

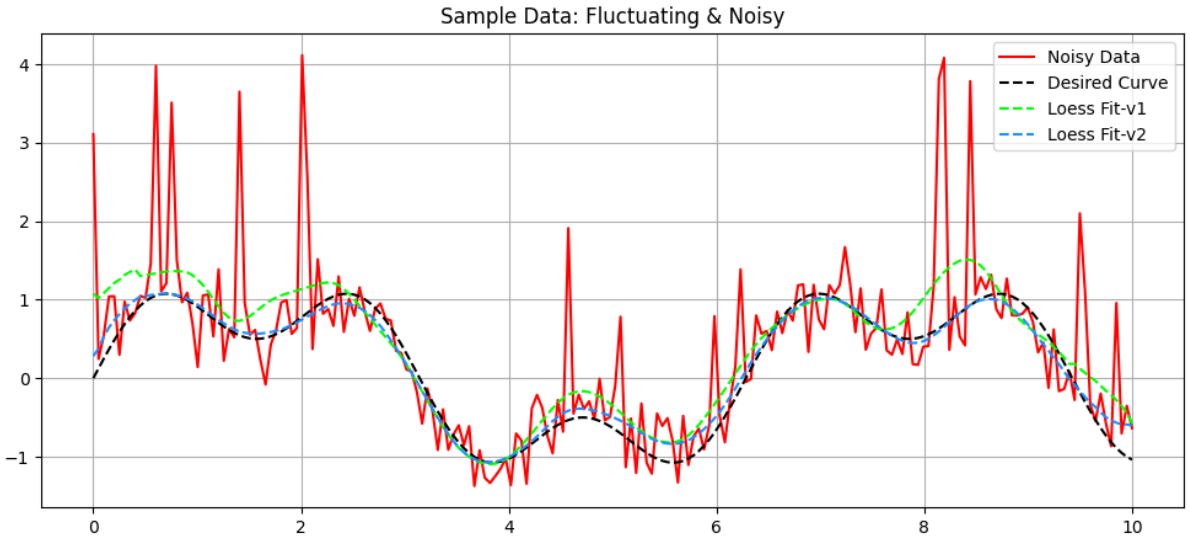

Figure 9: Curve smoothing algorithms

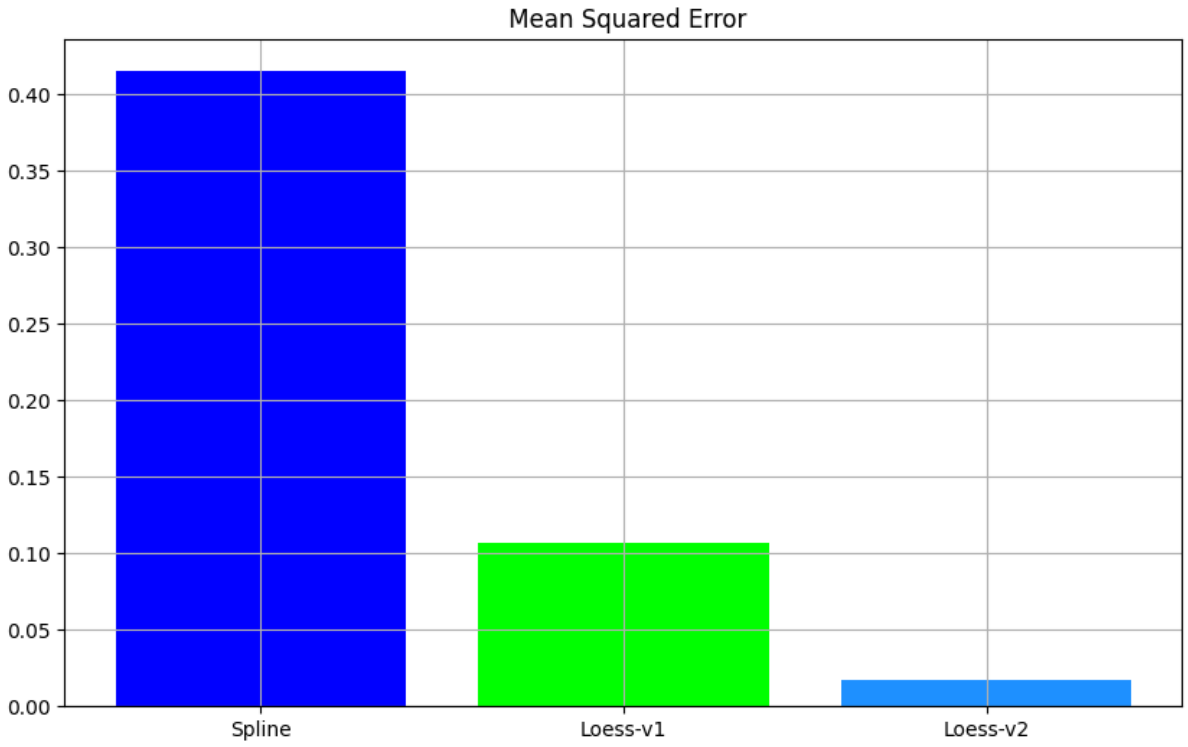

Figure 10: Curve smoothing algorithm evaluations

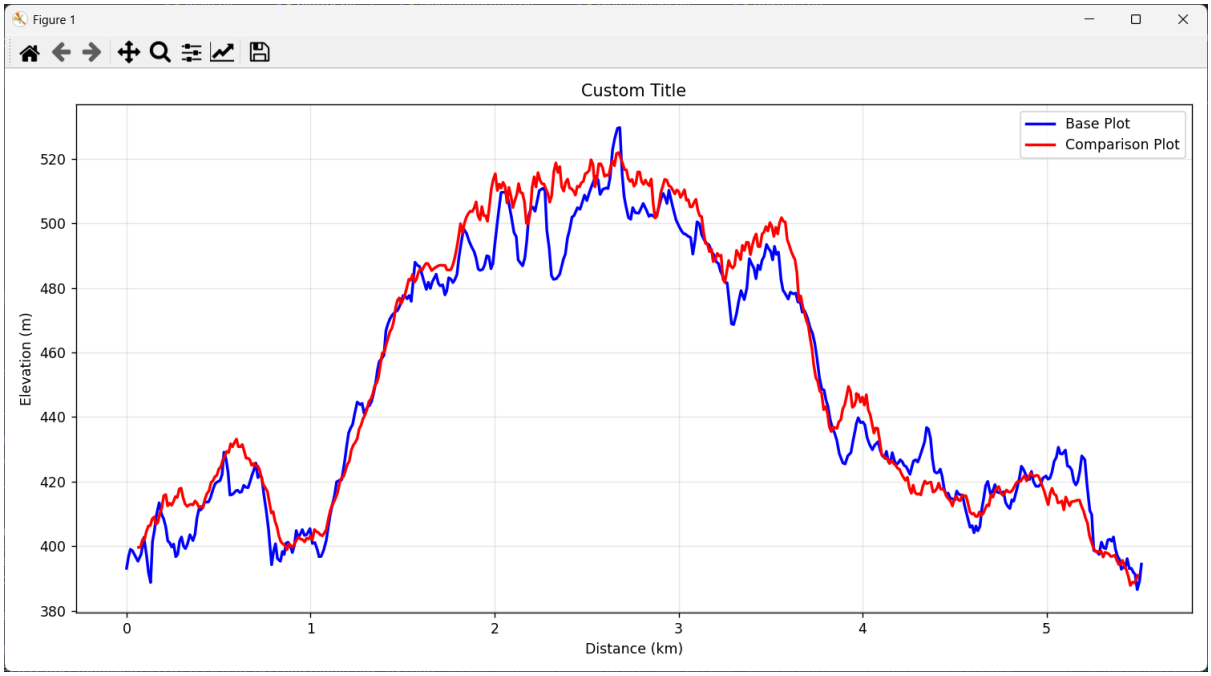

Figure 11: Elevation plot with Elevation sync

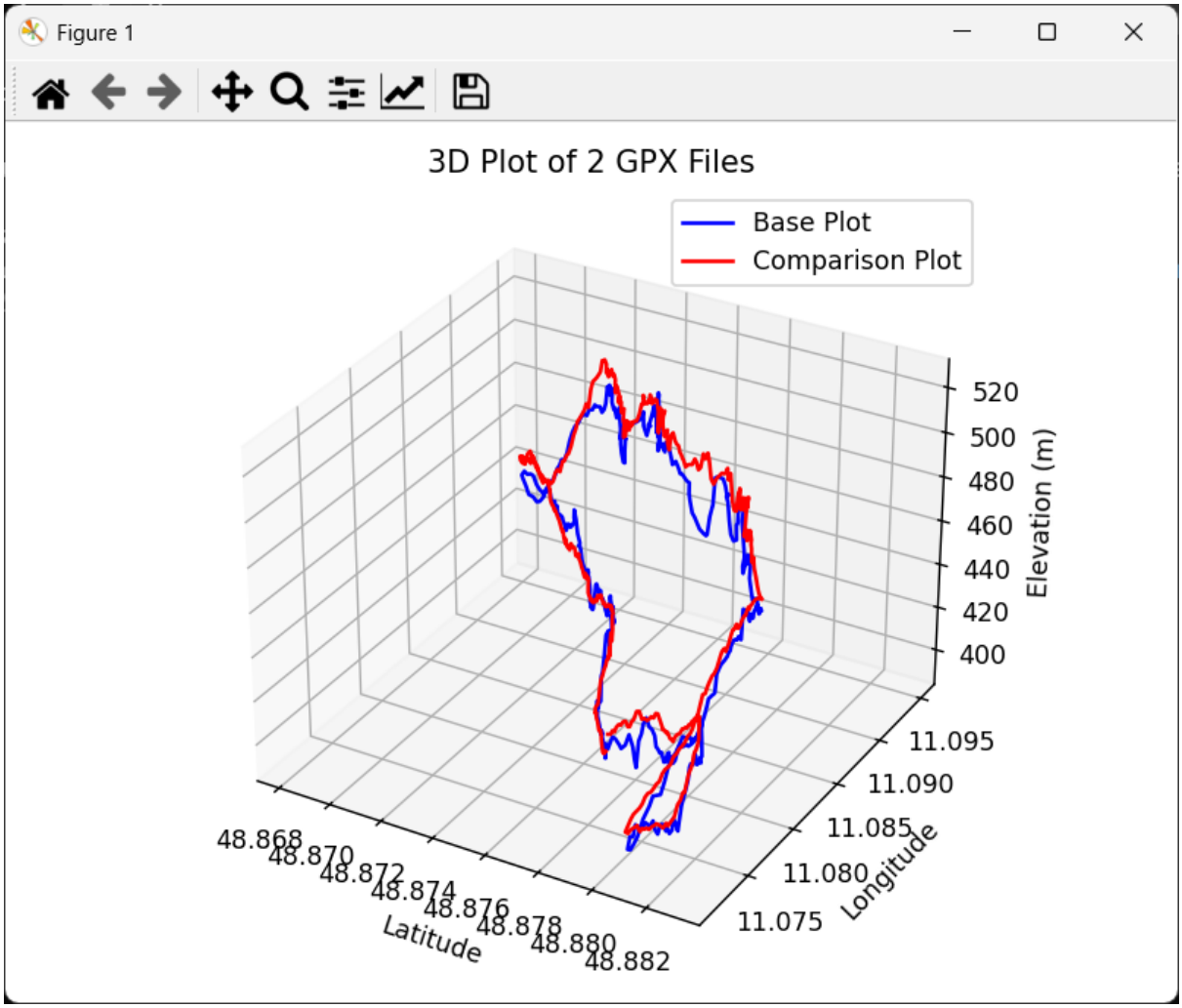

Figure 12: Comparison 3D plot

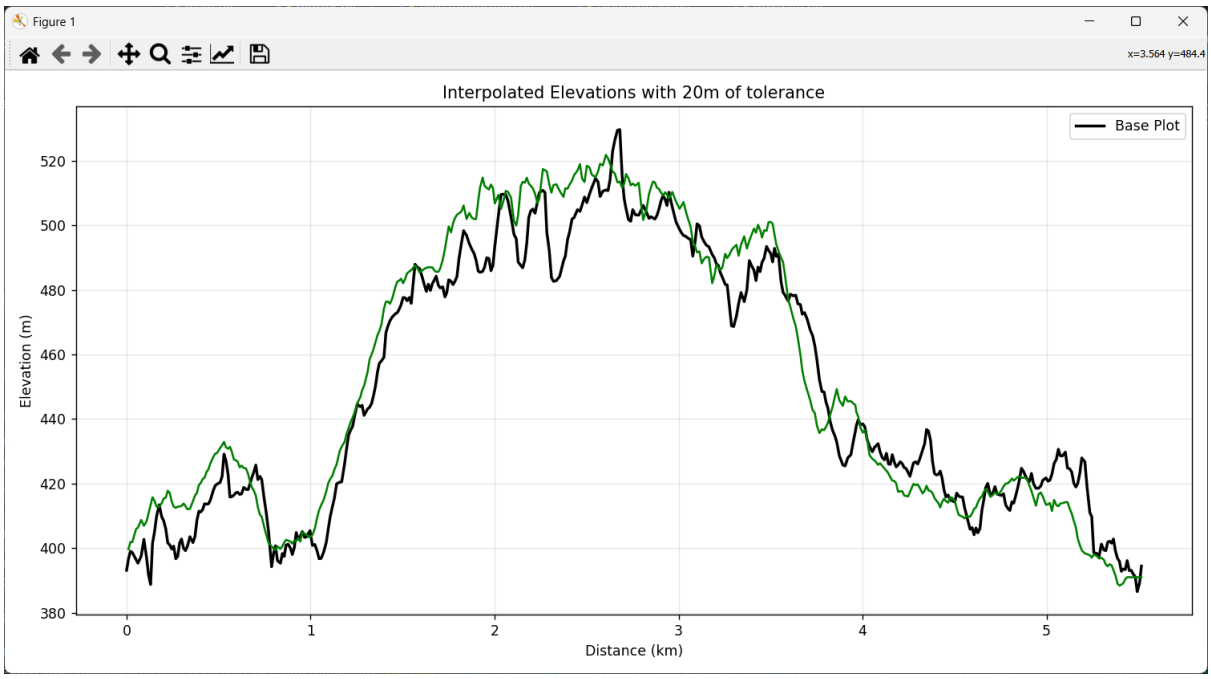

Figure 13: Elevation Plot with Interpolated Elevations and Tolerance

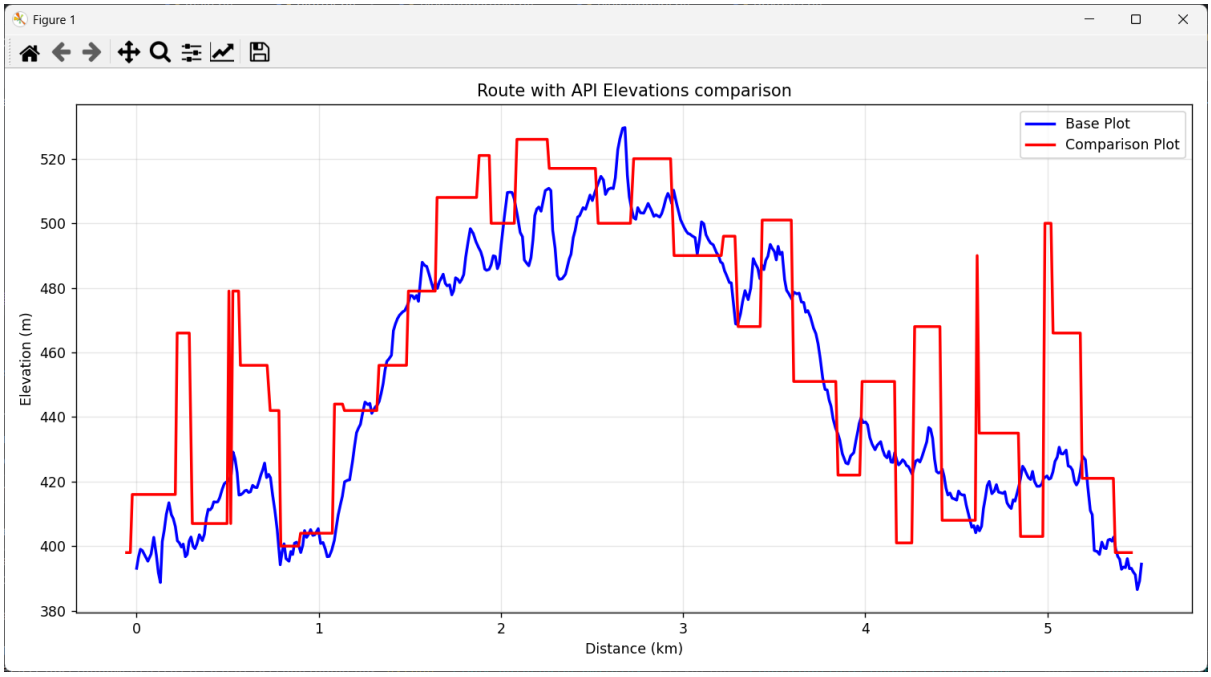

Figure 14: Elevation PLotusing API plot

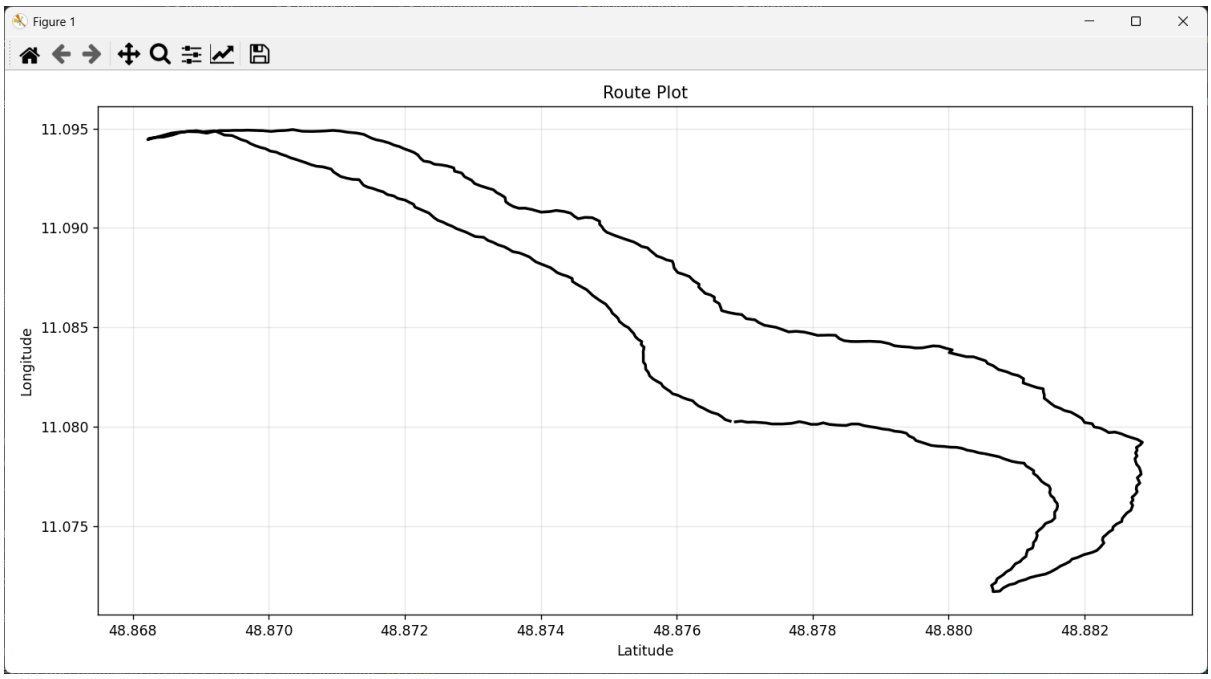

Figure 15: Single surface plot

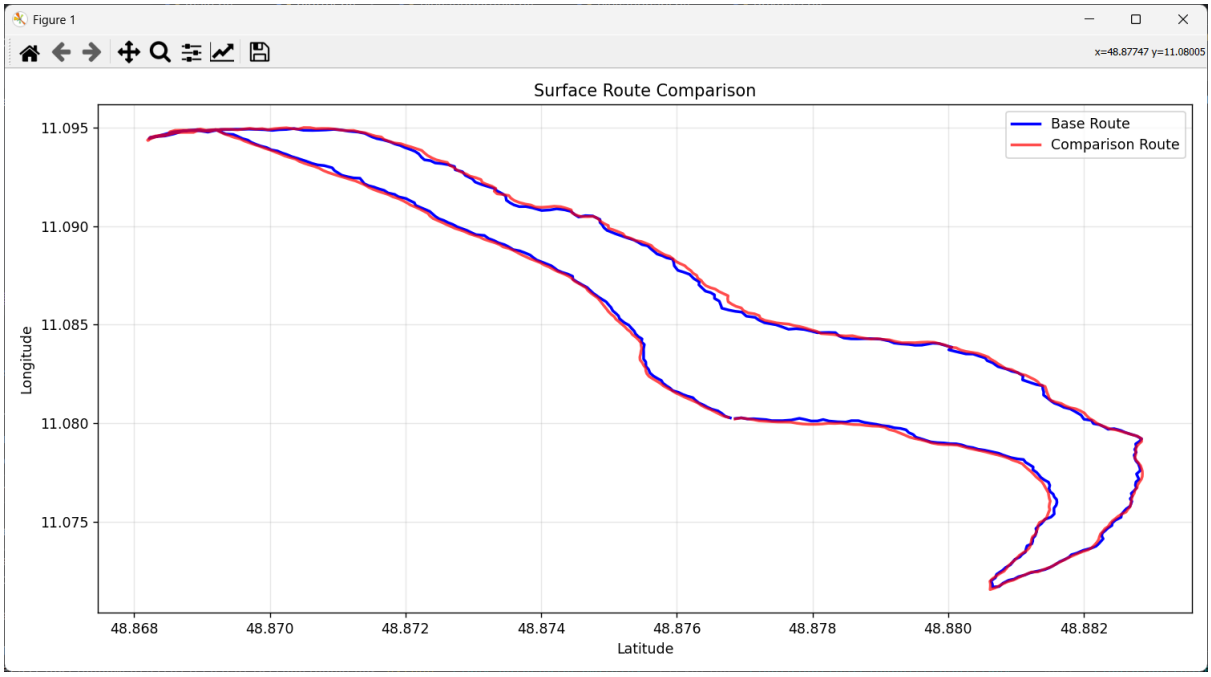

Figure 16: Surface comparison plot

Tom Williams: analysing discrepancies in recorded elevations. To ensure flexibility, I designed the system with optional divergence detection, beyond a user-defined tolerance, allowing users to focus solely on plotting the tracks if desired. The final tool offers three synchronisation methods, each with their own pros and cons, two divergence measurement options, an argument parser supporting various plot types, synchronisation methods, and tolerance settings inputted in the command line and the choice to either display graphs on screen or save them for later use. This modular approach provides users with adaptable and precise control over track comparison and visualisation.

12 APIs

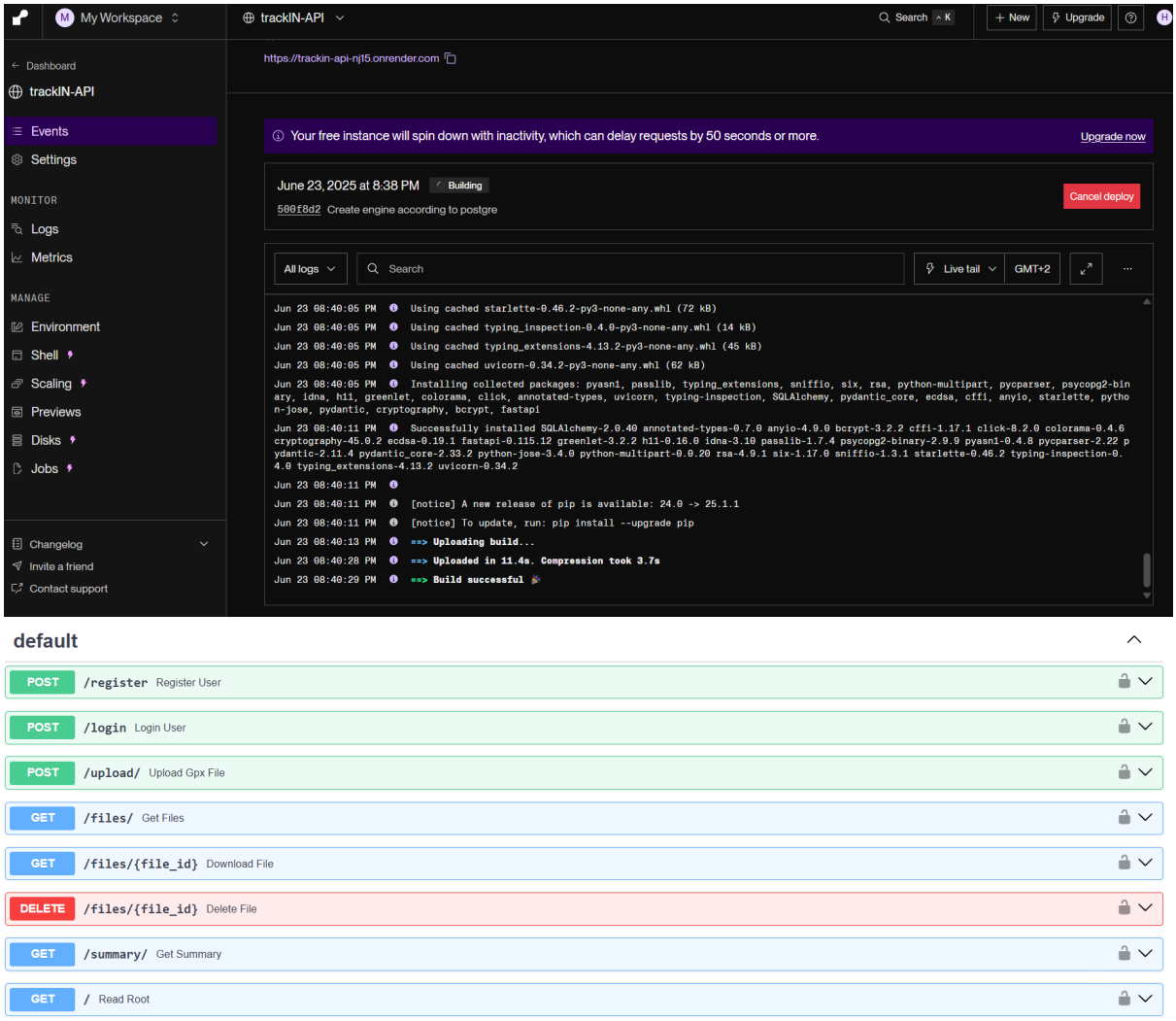

Figure 17: API and framework server

13 Frontend Testing

This document outlines the test protocol for the iOS version of the TrackIN app. The goal is to validate the app's user interface, functionality, data handling, error recovery, performance, and plotting methods.

Project: TrackIN
Platform: iOS
Version: [18.5]
Tester: [Baatarbileg]
Date: June 2025

13.1 User Interface Tests

Test Case ID	Description	Expected Result	Status / Notes
UI-01	Launch the app	Splash → Home screen appears	Pass
UI-02	Tap "Start" button	Starts tracking location/time	Pass
UI-03	Tap "Stop & GPX" button	Prompt to save/export GPX	Pass
UI-04	Toggle dark/light mode	UI adjusts without layout issues	Pass
UI-05	Rotate device	UI adapts to orientation	Pass

13.2 Functionality Tests

Test Case ID	Description	Expected Result	Status / Notes
FUNC-01	Start and end tracking session	GPX file is saved/shared	Pass
FUNC-02	Track while moving	Distance and altitude increase	Pass
FUNC-03	Stop without starting	No effect or show error	Pass
FUNC-04	Export GPX file	GPX exported with valid XML	Pass
FUNC-05	Import and view GPX file	Points/elevation shown on map	Needs Fix – file empty

13.3 Data Management Tests

Test Case ID	Description	Expected Result	Status / Notes
DATA-01	Altitude recording accuracy	Altitude changes recorded accurately	Pass
DATA-02	Distance recording accuracy	Distance increases with movement	Needs Check

DATA-03	Simulate GPS noise	App filters or interpolates properly	Needs Work
DATA-04	Timestamp consistency	All points timestamped correctly	Pass
DATA-05	GPX XML format validation	XML well-formed and valid	Needs Fix – remove namespace

13.4 Error Handling Tests

Test Case ID	**Description**	**Expected Result**	**Status / Notes**
ERR-01	Disable location permission	App requests permission or shows error	Pass
ERR-02	Airplane mode during tracking	Show GPS error or no signal warning	Pass
ERR-03	Kill app during session	App resumes or informs session loss	Pass – New session starts
ERR-04	Deny file write permission	Error shown; app remains stable	Pass

13.5 Performance Tests

Test Case ID	**Description**	**Expected Result**	**Status / Notes**
PERF-01	Run app for 30+ minutes	No crash or memory leak	Pass
PERF-02	Background tracking	Tracks data when screen is off	Pass

13.6 Plotting Method Tests

Test Case ID	**Description**	**Expected Result**	**Status / Notes**
PLOT-01	Spline interpolation	Smooth elevation graph	Needs Fix – sudden peaks
PLOT-02	Loess interpolation (v2)	Smooth and stable plot	Pass

The iOS app passes the majority of test cases across UI, functionality, error handling, and performance. Improvements are recommended in data smoothing, GPX importing, and XML formatting. Code coverage testing is enabled and will guide future test improvements.

Overall coverage testing had 12.8% usage: out of 1699 lines of code, 217 lines were hit.

LCOV - code coverage report

			Coverage	Total	Hit
Current view:	top level	Lines:	12.8 %	1699	217
Test:	lcov.info	Functions:	-	0	0
Test Date:	2025-06-24 21:58:03				

Directory	Line Coverage ≎ Rate	Total	Hit
lib/	80.8 %	26	21
lib/constants/	40.0 %	15	6
lib/domain/	0.0 %	4	
lib/functional/	1.2 %	242	3
lib/pages/	16.6 %	1098	182
lib/services/	1.6 %	314	5

Note: 'Function Coverage' columns elided as function owner is not identified.

Generated by: LCOV version 2.3.1-1

```
            :   /* ---------- PERMISSION HANDLING ---------------------------------- */
          1 :   Future<void> _checkLocationPermission() async {
          1 :     bool serviceEnabled = await Geolocator.isLocationServiceEnabled();
            :     if (!serviceEnabled) {
          0 :       _showLocationDialog();
            :       return;
            :     }
            :
          0 :     LocationPermission permission = await Geolocator.checkPermission();
          0 :     if (permission == LocationPermission.denied) {
          0 :       permission = await Geolocator.requestPermission();
          0 :       if (permission == LocationPermission.denied) {
            :         return;
            :       }
            :     }
            :
          0 :     if (permission == LocationPermission.deniedForever) {
          0 :       _showPermissionSettingsDialog();
            :       return;
            :     }
            :
            :     // Get position once to show initial coordinates
          0 :     _getInitialPosition();
            :
          0 :     setState(() {
          0 :       _permissionsEnabled = true;
            :     });
            :   }
            :
          0 :   void _showLocationDialog() {
          0 :     showDialog(
          0 :       context: context,
            :       barrierDismissible: false,
          0 :       builder: (ctx) => AlertDialog(
            :         title: const Text('Location Services Required'),
            :         content: const Text(
            :             'This app needs location services to track your position and altitude. '
            :             'Please enable location services in your device settings.'),
          0 :         actions: [
          0 :           TextButton(
          0 :             onPressed: () {
          0 :               Navigator.pop(ctx);
          0 :               Geolocator.openLocationSettings();
            :             },
            :             child: const Text('Open Settings'),
            :           ),
          0 :           TextButton(
          0 :             onPressed: () => Navigator.pop(ctx),
            :             child: const Text('Cancel'),
            :           ),
            :         ],
            :       ),
            :     );
            :   }
            :
          0 :   void _showPermissionSettingsDialog() {
          0 :     showDialog(
          0 :       context: context,
            :       barrierDismissible: false,
          0 :       builder: (ctx) => AlertDialog(
            :         title: const Text('Location Permission Required')
```

Figure 18: Coverage test for iOS

14 Backend Testing

14.1 Plotter Class Testing (`test_plotter.py`)

This document outlines the comprehensive testing strategy applied to the Plotter class within `plotter.py`. The tests leverage `pytest` for its testing framework and `unittest.mock` for effective mocking, ensuring thorough validation of the Plotter's functionality without actual graphical rendering.

1. Mocking Strategy (`mock_matplotlib` Fixture)

The `mock_matplotlib` fixture is a cornerstone of this test suite, designed to isolate the Plotter class's logic from `matplotlib`'s graphical output.

Purpose: To prevent `matplotlib` from opening actual plot windows during test execution. This significantly speeds up tests, eliminates reliance on a display environment, and makes the test suite suitable for continuous integration (CI) pipelines.

Implementation Details:

- It uses `patch('plotter.plt')` to replace the `matplotlib.pyplot` object (aliased as `plt` in `plotter.py`) with a `MagicMock`.
- Crucially, it simulates the `matplotlib` object hierarchy:
  - `mock_plt.figure.return_value = mock_fig`: Ensures that when `plt.figure()` is called, it returns a mock figure object (`mock_fig`).
  - `mock_fig.add_subplot.return_value = mock_ax`: Configures `mock_fig` so that calling `add_subplot()` on it returns a mock axes object (`mock_ax`).
- **Explicit Mocking:** Key `matplotlib.pyplot` functions (`plot`, `title`, `xlabel`, `show`, etc.) and `matplotlib.axes` methods (`plot`, `set_title`, etc.) that `Plotter` directly calls are individually mocked using `MagicMock()`.

Benefit: Enables tests to verify what `matplotlib` functions are called and with what arguments, rather than attempting to validate visual output.

2. Dummy Data Fixtures

To ensure self-contained and repeatable tests, controlled dummy data is used.

- `dummy_elevation_profile`: Provides a basic, valid `ElevationProfile` instance created with a list of `Point` objects.
- `another_dummy_elevation_profile`: Supplies a second, distinct `ElevationProfile` with different (but similarly structured) point data.

3. Test Cases (`TestPlotter` Class)

The `TestPlotter` class is structured to systematically test each component of the `Plotter` class.

a. Initialization Tests (`test_init_*`)

- `test_init_no_profiles`: Initializes `Plotter` without any profiles. Verifies `plotter.profiles` is empty.
- `test_init_with_single_profile_auto_named`: Adds one `ElevationProfile` without a name. Profile is auto-named `"Profile 1"`.
- `test_init_with_single_profile_named`: Adds one profile with a given name.
- `test_init_with_single_profile_named_none`: Name is `None` → auto-named `"Profile 1"`. (Edge Case)

- `test_init_with_multiple_profiles_auto_named`: Adds multiple unnamed profiles → named sequentially.
- `test_init_with_multiple_profiles_mixed_names`: Mix of named and unnamed profiles added correctly.

b. Unique Name Generation Tests (`_generate_unique_name`)

- `test_generate_unique_name_empty`: Returns `"Profile 1"` when no profiles exist.
- `test_generate_unique_name_existing`: Returns `"Profile 2"` when `"Profile 1"` exists.
- `test_generate_unique_name_with_gap`: Handles skipped names like `"Profile 1"` and `"Profile 3"` → returns `"Profile 2"` (Edge Case).

c. Adding Profiles Tests (`add_profiles`)

- `test_add_profiles_single_auto_name`: Adds unnamed profile → auto-named.
- `test_add_profiles_single_named`: Adds named profile.
- `test_add_profiles_multiple_named`: Adds multiple named profiles.
- `test_add_profiles_mixed_input`: Mix of named and unnamed profiles in one call.
- `test_add_profiles_duplicate_name_overwrites`: Duplicate name replaces old one (Edge Case).
- `test_add_profiles_empty_string_name_auto_named`: Empty string name → auto-named (Edge Case).
- `test_add_profiles_none_name_auto_named`: `None` as name → auto-named (Edge Case).
- `test_add_profiles_invalid_input`: Invalid inputs raise `ValueError` (Edge Case).

d. Setting Profiles Tests (`set_profiles`)

- `test_set_profiles_valid`: Replaces profiles with valid dict.
- `test_set_profiles_empty_dict`: Clears all profiles (Edge Case).
- `test_set_profiles_invalid_type`: Non-dict input → `TypeError` (Edge Case).
- `test_set_profiles_invalid_value`: Invalid value types → `ValueError` (Edge Case).

e. Plotting Method Tests

- `test_*_no_profiles`: No profiles → "No profiles to plot." printed; no matplotlib functions called (Edge Case).
- `test_plot_distance_vs_elevation_single_profile`: Single profile, custom title/labels → all plot functions called once.
- `test_plot_distance_vs_elevation_multiple_profiles`: Multiple profiles → verify plot call count and arguments.

- `test_plot_3d_lat_lon_elevation_single_profile`: 3D plot for single profile → verify projection, labels, and plot call.
- `test_plot_3d_lat_lon_elevation_multiple_profiles`: Multiple 3D profiles → verify correct call sequence for each.

Conclusion: This detailed testing approach ensures the `Plotter` class is robust, handles various valid and invalid inputs gracefully, and correctly interacts with its `matplotlib` dependency via mock objects.

14.2 Model Classes Testing (`test_models.py`)

This document outlines the testing strategy, covered functionalities, and specific edge cases addressed in the `test_models.py` file. The primary goal of these tests is to ensure the robust and correct behavior of the `Point`, `ElevationProfile`, and `Track` classes defined in `models.py`.

1. Introduction

`test_models.py` contains unit tests designed to verify the core data structures used throughout the GPX data processing application. These tests are isolated from external dependencies wherever possible through the use of mocking, ensuring fast, reliable, and repeatable execution.

2. Tests for the `Point` Class

- `test_point_init_with_elevation`: Verifies initialization with latitude, longitude, and elevation.
- `test_point_init_without_elevation`: Elevation defaults to `None` if not provided.
- `test_point_to_dict_with_elevation` / `without_elevation`: Checks dictionary output depending on elevation presence.
- `test_point_haversine_distance_identical_points`: Ensures distance between identical points is 0.
- `test_point_haversine_distance_known_values`: Verifies correct distance computation with known locations using `pytest.approx`.
- `test_point_distance_to`: Ensures `distance_to()` internally calls static `haversine_distance()` (mocked).
- `test_point_copy`: Confirms `copy()` produces a new object with same values.

3. Tests for the `ElevationProfile` Class

- Initialization:
  - `test_elevation_profile_init_empty_list`, `single_point`, `multiple_points`
  - For multiple points, cumulative distances are verified with mocked `Point.haversine_distanc`
- Accessors:

  - `get_latitudes`, `get_longitudes`, `get_elevations`, `get_distances`
- Mutators:
  - `set_elevations_valid`, `with_none_values`, `empty_list`, `length_mismatch`
- Statistics:
  - `get_elevation_stats_normal_case` `get_elevation_stats_with_none_elevations_...`: Highlights a known `TypeError` bug with `None` values.
  - `get_elevation_stats_empty_profile`, `all_none_elevations`, `single_point`
- `test_elevation_profile_copy`: Verifies deep copying of profiles.

4. Tests for the `Track` Class

Focus: The `from_gpx_file` factory method using mocked I/O and external dependencies.

Mocking Strategy:

- `patch('builtins.open', mock_open(...))`: Prevents file I/O, supplies dummy GPX.
- `patch('gpxpy.parse')` returns a mocked GPX object.
- Mocked internal GPX hierarchy: `tracks`, `segments`, `points`.
- `patch('models.GeoidKarney')` returns 0.0 when called as function.
- `patch('pygeodesy.ellipsoidalKarney.LatLon')` returns mock `LatLon`.
- `patch.object(Point, 'haversine_distance')` and `distance_to` mocked to return 1.0.

Track Test Cases:

- `test_track_from_gpx_file_valid_data`: Creates `Track` with points + elevations.
- `test_track_from_gpx_file_no_elevation_data`: Expects `TypeError` on `None` elevation (known issue).
- `test_track_from_gpx_file_empty_gpx_object`: Yields an empty `Track`.
- `test_track_from_gpx_file_gpx_parse_exception`: Simulates parse error → raises `ValueError`.
- `test_track_from_gpx_file_multiple_tracks_and_segments`: Verifies correct point concatenation.

5. Testing Practices Used

- `pytest` for discovery, assertion, and execution.
- `unittest.mock`: Extensive use of `patch`, `MagicMock`, `mock_open`.
- `pytest.approx`: Used for floating-point tolerance.
- Clear assertion patterns: `assert`, `pytest.raises`, equality checks.

6. Conclusion and Developer Notes

All tests in `test_models.py` are passing, supporting a reliable data layer.

Important Observation: The following tests reveal a runtime bug:

- `test_elevation_profile_get_elevation_stats_with_none_elevations_...`
- `test_track_from_gpx_file_no_elevation_data`

Both expose that `None` elevation values cause `TypeError` in arithmetic.

Recommendation: Modify `models.py` to gracefully handle `None` elevations:

- Skip or interpolate `None` values in stats.
- Avoid subtracting float from `None` in `Track.from_gpx_file`.

This will improve the system's robustness when handling real-world GPX data with incomplete elevation profiles.

14.3 Elevation API Testing (`test_elevation_api.py`)

This section documents the unit testing performed on the `elevation_api.py` module, which defines two classes: `ElevationAPI` and `OpenElevationAPI`. These classes are responsible for retrieving elevation data based on geographic coordinates via external APIs.

1. Introduction

The goal of these tests is to ensure the robustness of both API clients in handling successful responses as well as a wide range of failure scenarios.

2. Testing Approach

All tests use Python's built-in `unittest` framework. Extensive mocking is employed to isolate logic from actual web API calls.

Mocking Strategy:

- `requests.post` is patched to:
  - Prevent real HTTP calls
  - Return custom mock responses for different scenarios
  - Simulate exceptions (e.g., `ConnectionError`)
  - Assert correctness of request URL and payload
- `sys.stdout` is patched using `io.StringIO` to capture and verify printed error messages.

3. Tested Components

- `ElevationAPI.get_elevations()`
- `OpenElevationAPI.get_elevations()`

4. Detailed Test Scenarios

The following test scenarios and edge cases are implemented for both classes.

4.1 Handling Empty List of Points

- **Purpose:** Ensure method returns an empty list if input is empty.
- **Behavior:** No API call is made.

4.2 Successful API Response

- **Mocking:** `requests.post` returns a mocked object with `ok = True` and valid JSON.
- **Behavior:**
  - Parsed elevations match mock data.
  - Request URL and payload are validated.

4.3 HTTP Error Response (4xx/5xx)

- **Mocking:** `ok = False`, e.g., status code 404 or 500.
- **Behavior:**
  - Returns empty list.
  - Prints specific API error message to `stdout`.

4.4 Network Connection Error

- **Mocking:** `requests.post.side_effect = ConnectionError("Test error")`
- **Behavior:**
  - Returns empty list.
  - Prints connection error message.

4.5 Generic Unexpected Exception

- **Mocking:** `side_effect = Exception("Some unexpected issue")`
- **Behavior:**
  - Returns empty list.
  - Prints "Unexpected error: ..." to `stdout`.

5. Conclusion

All test cases for `ElevationAPI` and `OpenElevationAPI` passed successfully. The use of extensive mocking enables fast, reliable testing and confirms that the APIs handle:

- Successful responses

- API-level HTTP errors
- Network failures
- Unexpected exceptions

This demonstrates that the module is well-prepared for real-world deployment where connectivity and data integrity cannot always be guaranteed.

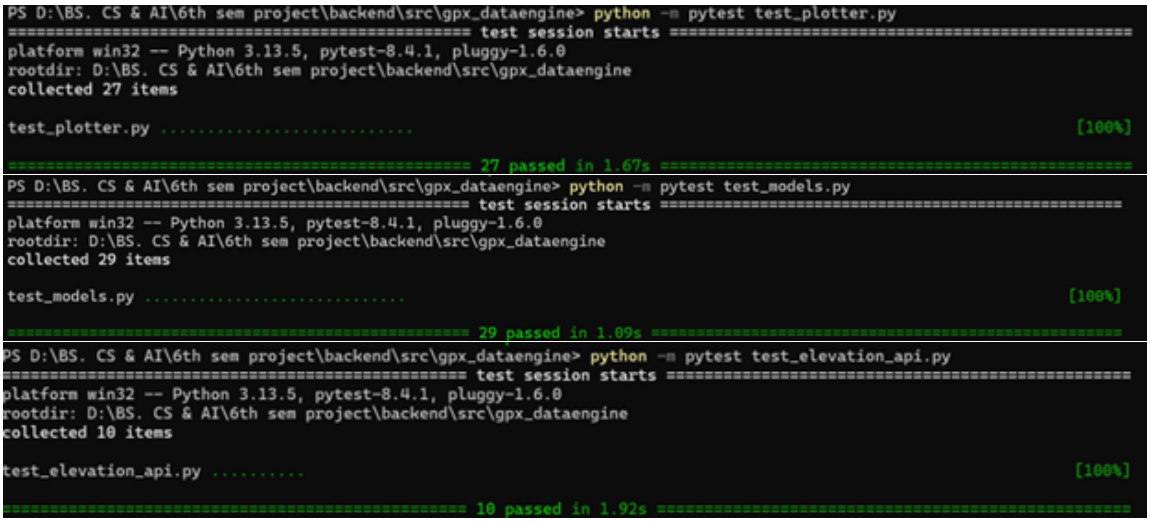

Figure 19: Execution of each Python test file on terminal to verify passing status.

15 GUI

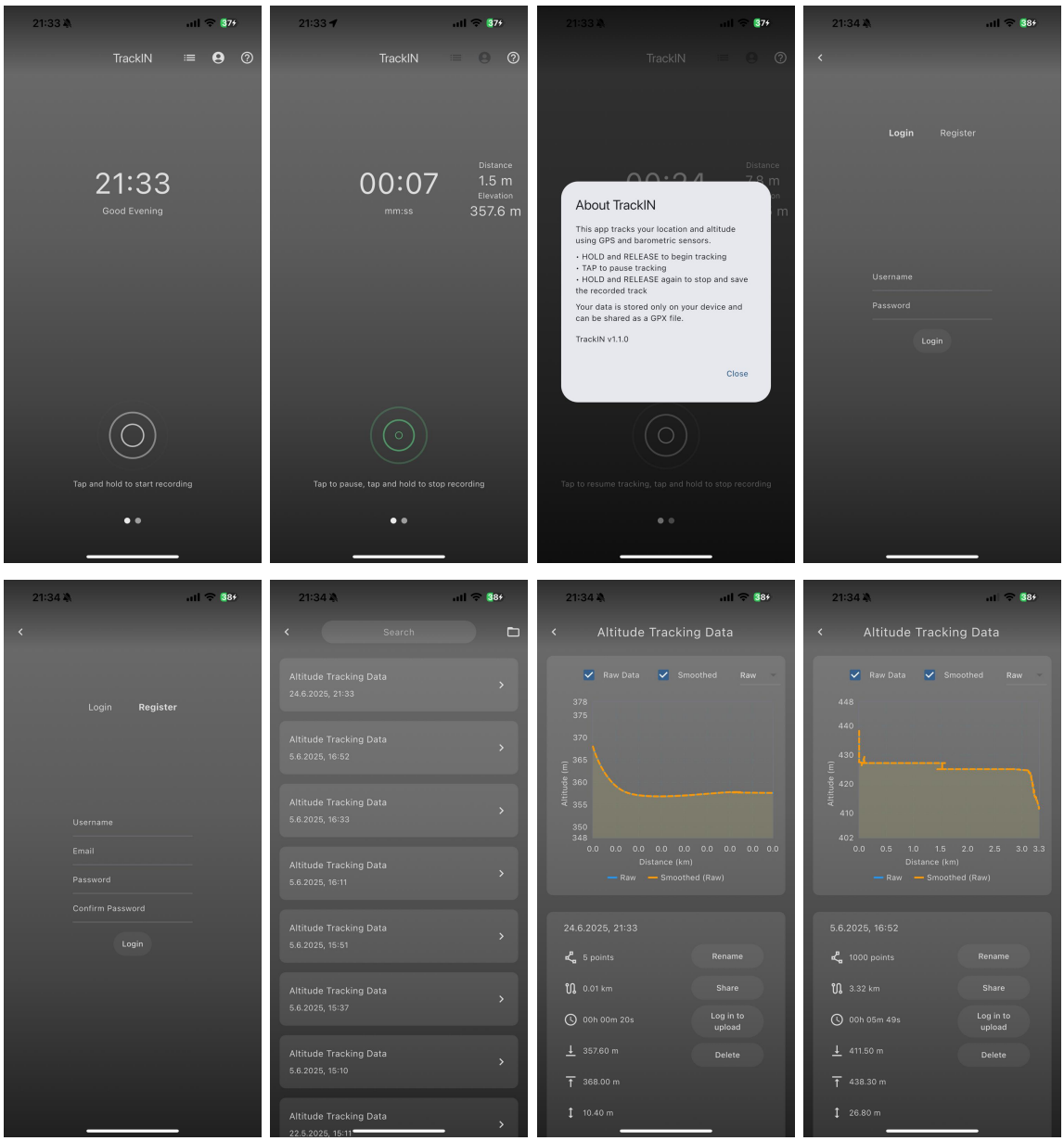

Figure 20: Flutter Application/ iOS and Andriod GUI

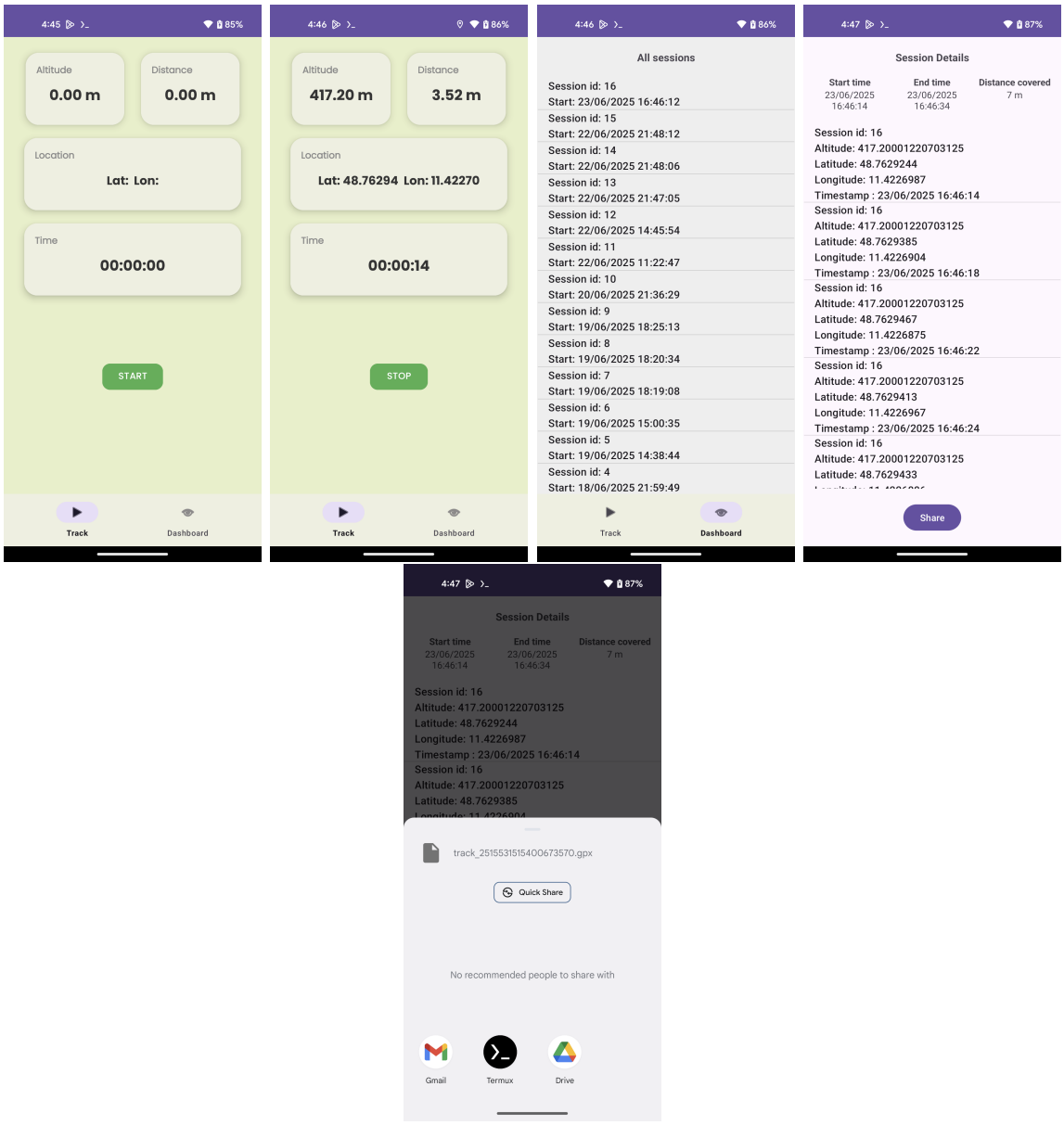

Figure 21: Andriod application GUI by Java

16 GitHub Repository Structure

TrackINJava

```
TrackINJava/
|
|-- .idea/
|-- app/
|-- gradle/
|
|-- .gitignore
|-- build.gradle.kts
|-- gradle.properties
|-- gradlew
|-- gradlew.bat
```

backend

```
backend/
|
|-- data/
|-- docs/
|-- src/
|-- test/
|
|-- .gitignore
|-- LICENSE
|-- README.md
|-- main.py
|-- poetry.lock
|-- pyproject.toml
|-- requirements.txt
```

frontend

```
Flutter Application/
|
|-- android/
|-- ios/
|-- lib/
|-- linux/
|-- macos/
|-- test/
|-- web/
|-- windows/
|
|-- .gitignore
|-- .metadata
```

```
|-- .README.md
|-- analysis_options.yaml
|-- main.py
|-- pubspec.lock
|-- pubspec.yaml
|-- trackin_icon_v1.png
```

Docs

```
Task documents/
|
|-- documents/
```

serverTesting

```
serverTesting/
|
|-- backend/
|-- main.py
|-- render.yaml
|-- requirement.txt
```

17 Conclusion

Throughout the project, we are successful in developing a cross platform tracking application with a detailed focus on altitude data analysis using GNSS-derived coordinates. Despite the time constraints and complexity of the integration of multiple components like frontend, GUI, backend processing, data synchronization and algorithmic comparison – we were able to achieve working prototype with functionalities.

Key accomplishments include the implementation of persistent background tracking, alignment and interpolation of GPX tracks, curve smoothing algorithms, and an extensible framework for visualizing and evaluating geospatial data. All components were structured with modularity in mind to allow for easier future extension.

However, due to limited development time and technical requirements, complexity of flutter, some features remain needing improvement. Advanced elevation correction (e.g., AI-assisted smoothing), multi-user data management, and comprehensive UI polishing are among the areas identified for further improvement.

Future work could focus on:

- Enhancing real-time data synchronization between devices or with cloud storage.
- Integrating elevation correction using machine learning models or external APIs (e.g., Google Elevation, Mapbox Terrain).
- Expanding testing across diverse terrain types and recording conditions.
- Allow complete customisation of plots in the backend by providing settings in a JSON file

In summary, the current version of the app serves as a solid foundation for further development and experimentation. With additional time and resources, it has strong potential to become a robust tool for elevation-aware route tracking and analysis.

Acknowledgements: We would like to thank Prof. Dr. Robert Gold for his guidance and support throughout the course of this project. Special credit goes to all team members—Baatarbileg Erkhembayar, Syed Aayan Ahmed, Robin Roßnagel, Tom Williams, Himanka Ashan, Susheel Kumar, Prashil Rupapara, Isaac Ye, Pamirbek Almazbekov—for their contributions in development, testing, and documentation.