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

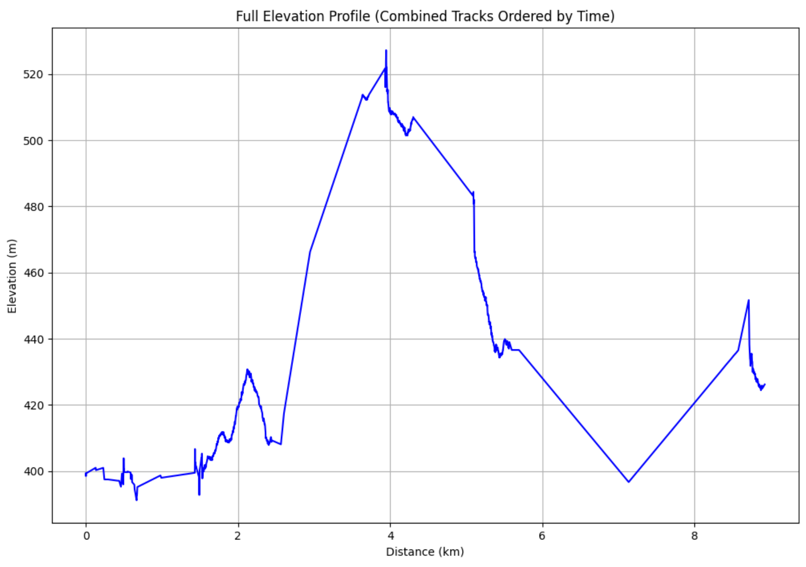


Figure 1: First GPX file recorded by our app

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 Andriod App	App's operational workflow
	T23	Modules of app	Modularization of 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 Andriod App	App's operational workflow
	T23	Modules of app	Modularization of app.
Isaac Ye	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
	Task	Andriod fixes	Android setup and changed a dependency that did not support Android
	Task	App Development	(Track Pages, UI Changes, Code Restructure)
	Task	App Development	(Notifications, Y/N Dialogue, Fix Android Import, Home Initial Redesign))
	Task	App Development	(Fixes, Convert Status to Notifications, Location Permission Denied Logic)
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.

Member	Task ID	Title	Description & Result
	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 App Development (More Fixes, Lots of Styling) Fixed display of statistics and convert UTC to local time, more styling to the home page, tracks page and track page.

App Development (Remove max points, remove pause on app exit, change GPS to stream, run in background (Android only), limit points rendered in track details) Allowed for infinite points and tracking while not in the app. Also changed GPS to stream instead of querying for the location. Changed GPS interval to be less frequent. Limited rendered point information (since we could have infinite points).

App Development (Framework Client, Account Page, Upload / Delete File, UI Fixes, Tidy) Added client for interacting with the 'framework' to upload GPX files. Added an account page (to login and interact with framework). Elementary attempt to map local files to uploaded files (to ensure we don't upload twice and can delete them). More UI fixes and cleaning up.

App Development (Better Local to Server File Mapping, Better UX + Concurrency when loading + parsing GPX files, Add Statistic Headers) Use hashing to compare contents of GPX files. Loading the tracks page now doesn't try to spawn too many threads that it hangs the render thread. Added headers to statistic 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.

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.

Project Documentation This document serves as the official project documentation.

Frontend iOS Testing iOS frontend was tested using a MacBook and Xcode. Test cases were created, and code coverage analysis was performed. Improvement areas were identified.

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

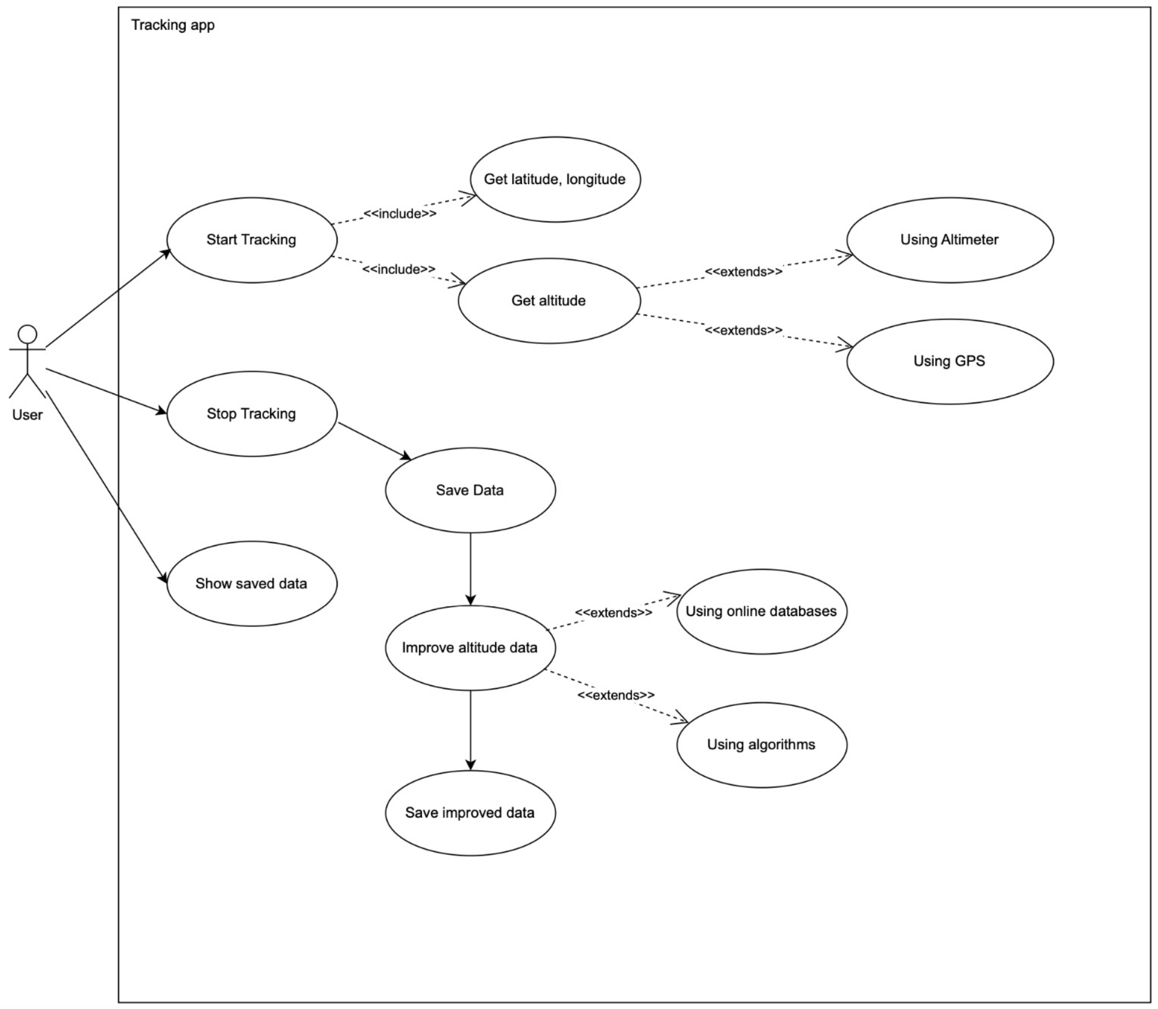


Figure 2: Use case diagram for TrackIN

9 Plotting

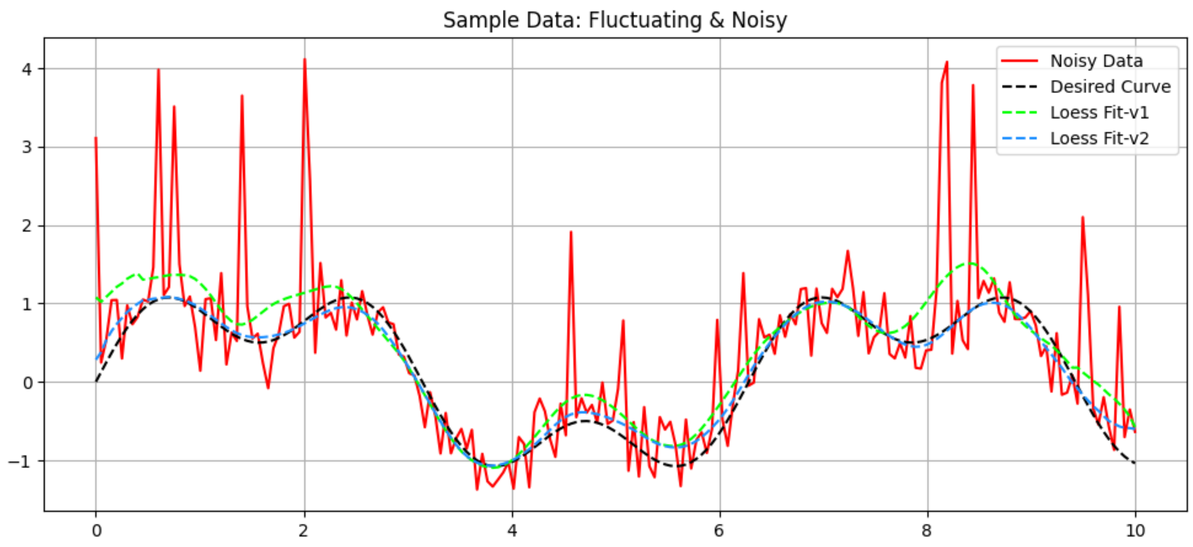


Figure 3: Curve smoothing algorithms

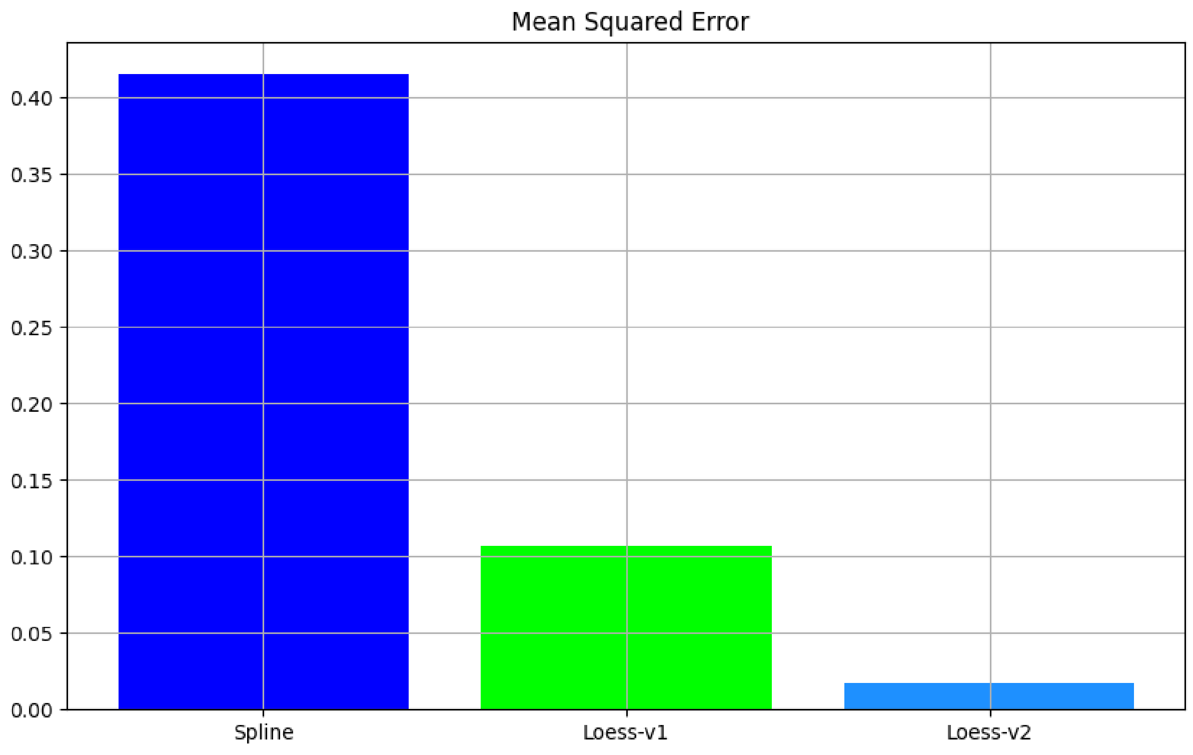


Figure 4: Curve smoothing algorithm evaluations

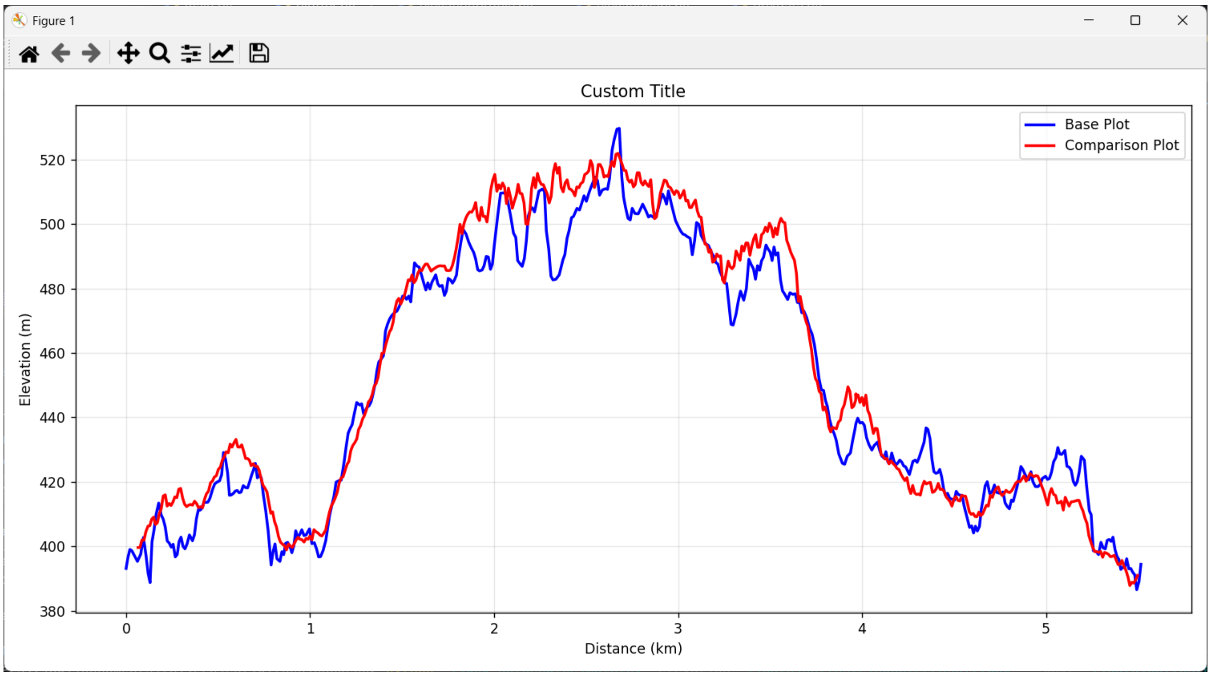


Figure 5: Elevation plot with Elevation sync

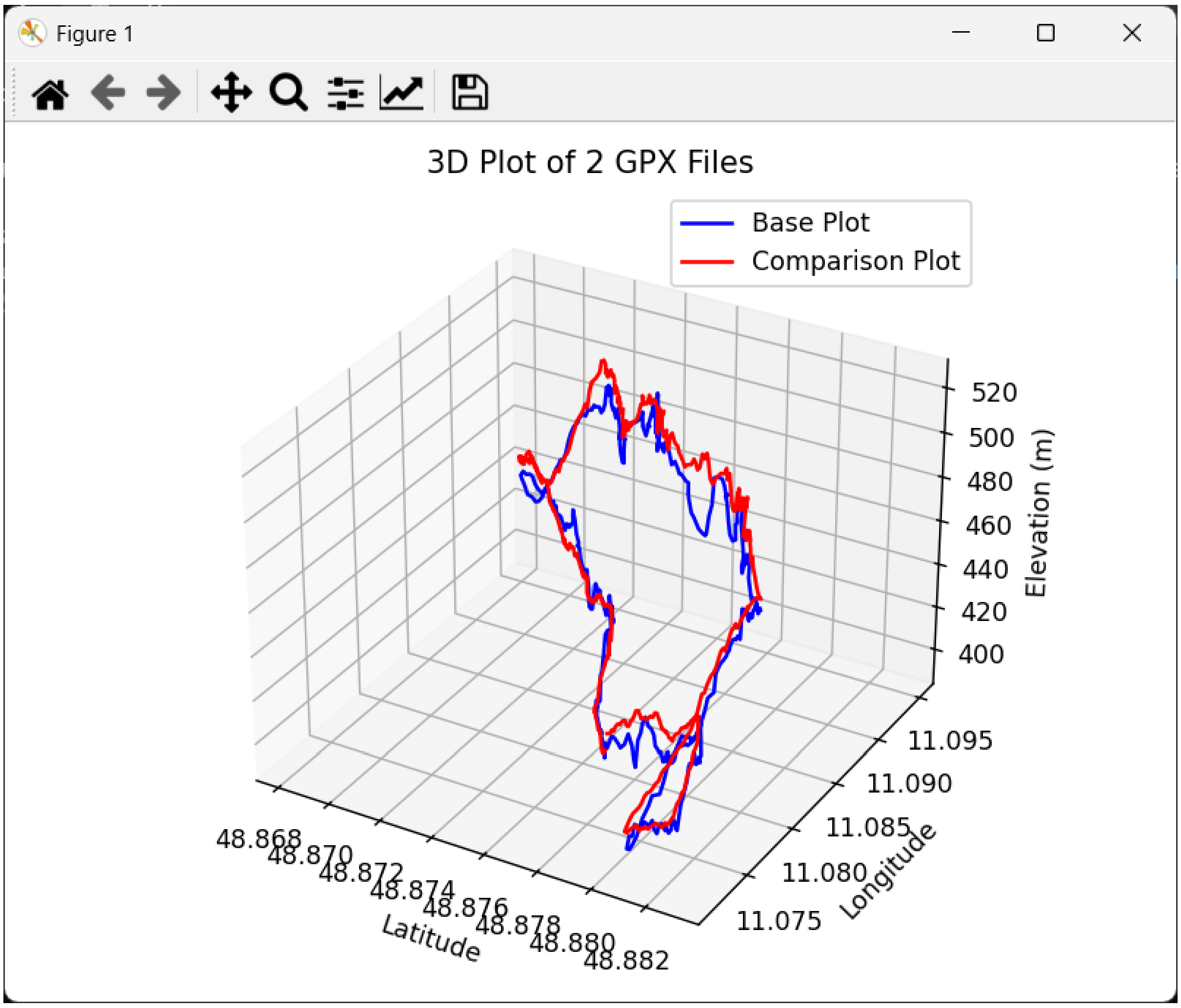


Figure 6: Comparison 3D plot

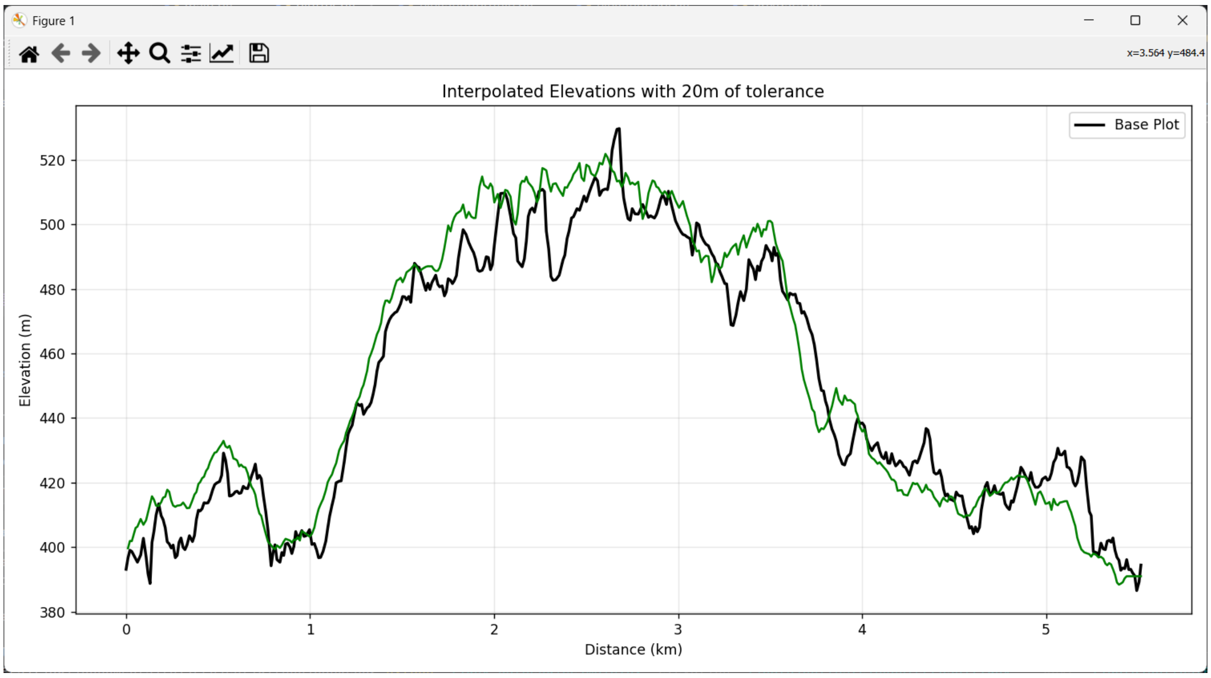


Figure 7: Elevation Plot with Interpolated Elevations and Tolerance

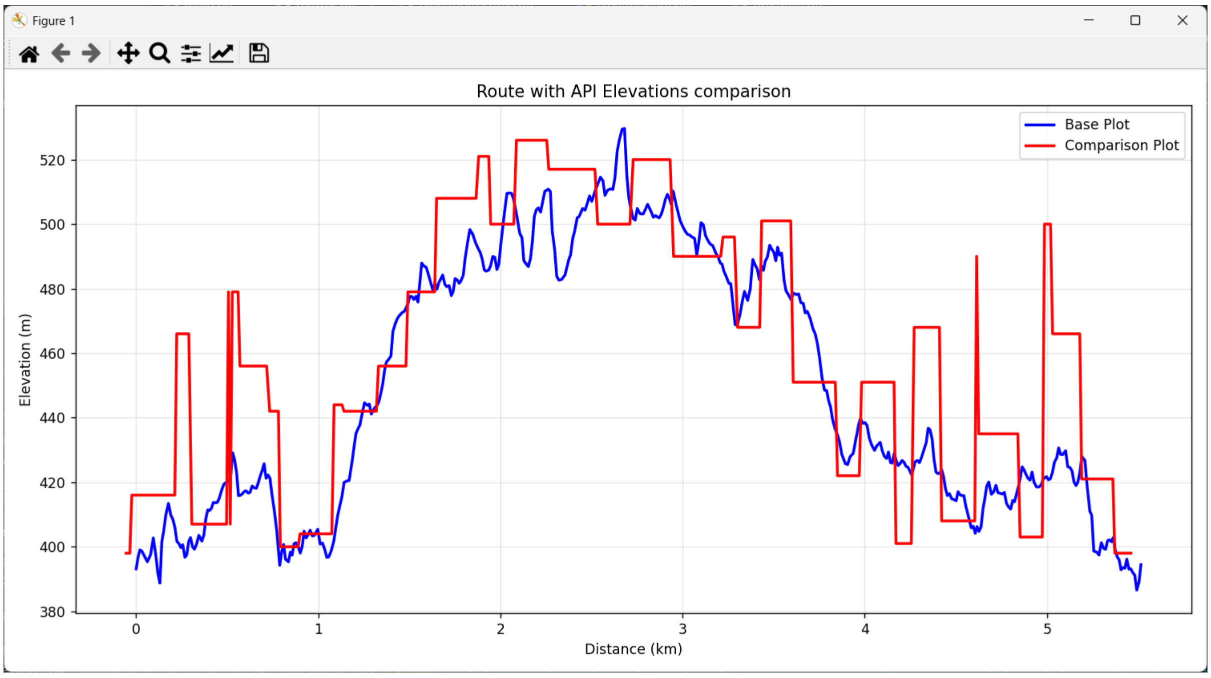


Figure 8: Elevation PLotusing API plot

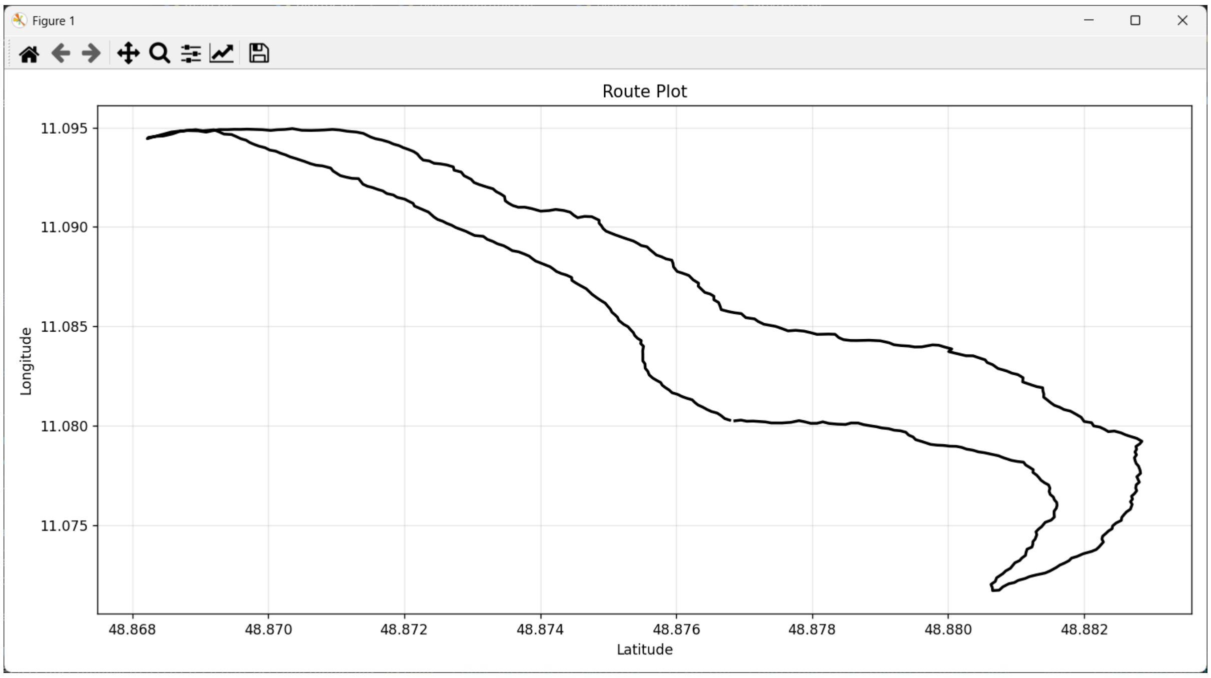


Figure 9: Single surface plot

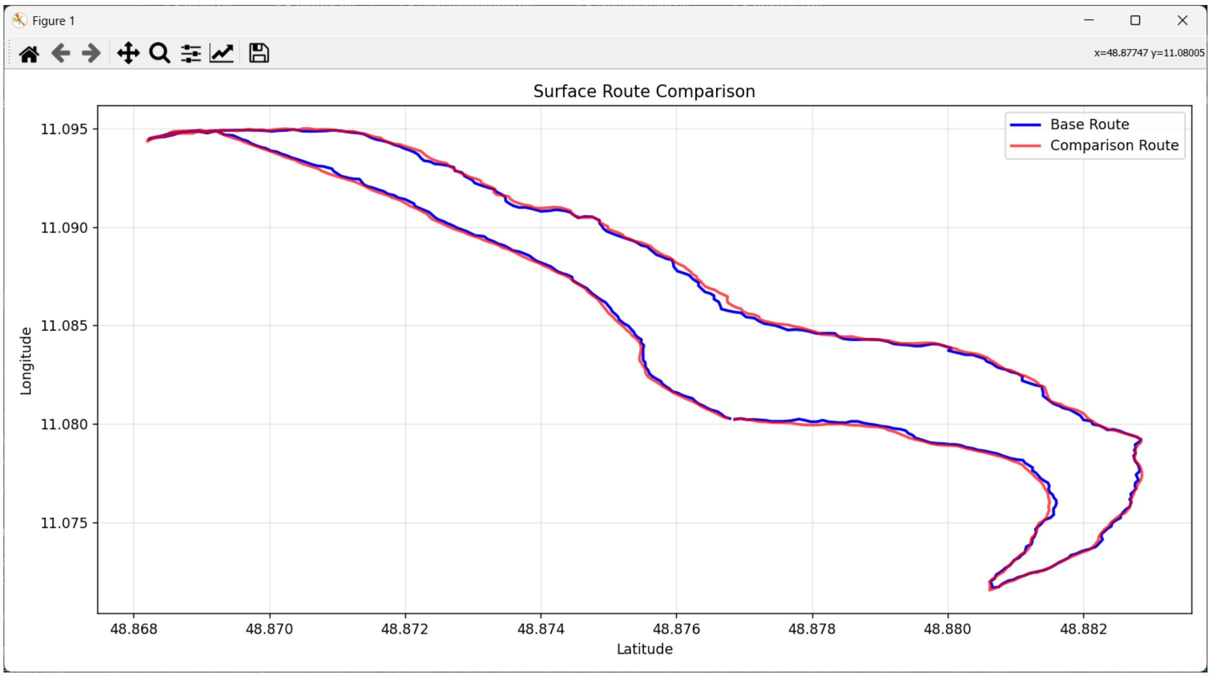
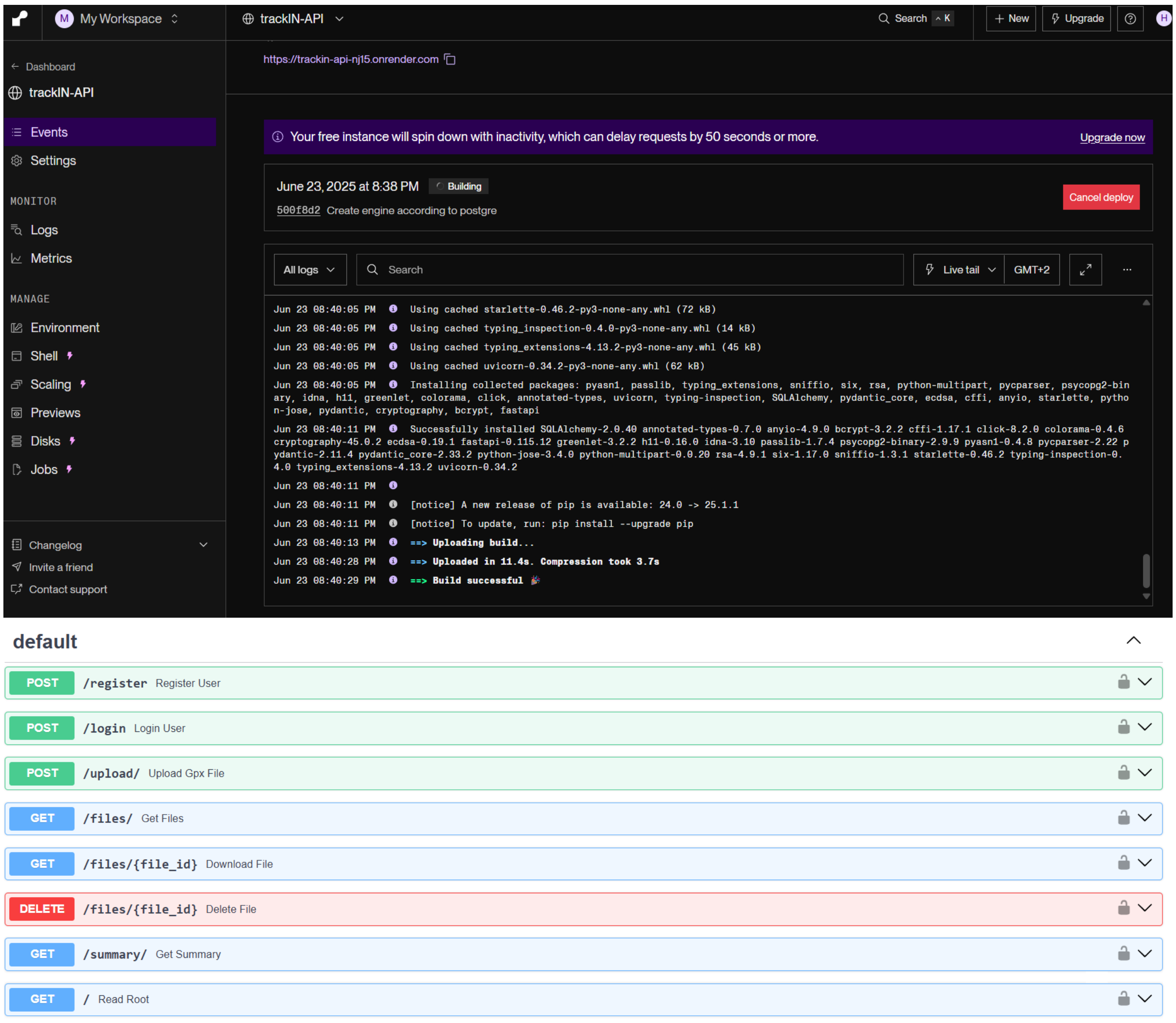


Figure 10: Surface comparison plot

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.

10 APIs



Your free instance will spin down with inactivity, which can delay requests by 50 seconds or more.

June 23, 2025 at 8:38 PM

500f8d2 Create engine according to postgre

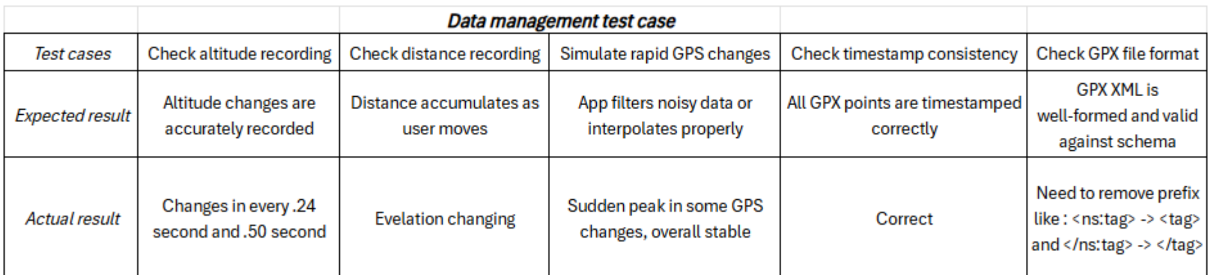
```
Jun 23 08:40:05 PM  Using cached starlette-0.46.2-py3-none-any.whl (72 kB)
Jun 23 08:40:05 PM  Using cached typing_inspection-0.4.0-py3-none-any.whl (14 kB)
Jun 23 08:40:05 PM  Using cached typing_extensions-4.13.2-py3-none-any.whl (45 kB)
Jun 23 08:40:05 PM  Using cached uvicorn-0.34.2-py3-none-any.whl (62 kB)
Jun 23 08:40:05 PM  Installing collected packages: pyasn1, passlib, typing_extensions, sniffio, six, rsa, python-multipart, pycparser, psycopg2-bin
ary, idna, h11, greenlet, colorama, click, annotated-types, uvicorn, typing-inspection, SQLAlchemy, pydantic_core, ecdsa, cffi, anyio, starlette, pytho
n-jose, pydantic, cryptography, bcrypt, fastapi
Jun 23 08:40:11 PM  Successfully installed SQLAlchemy-2.0.40 annotated-types-0.7.0 anyio-4.9.0 bcrypt-3.2.2 cffi-1.17.1 click-8.2.0 colorama-0.4.6
cryptography-45.0.2 ecdsa-0.19.1 fastapi-0.115.12 greenlet-3.2.2 h11-0.16.0 idna-3.10 passlib-1.7.4 psycopg2-binary-2.9.9 pyasn1-0.4.8 pycparser-2.22 p
ydantic-2.11.4 pydantic_core-2.33.2 python-jose-3.4.0 python-multipart-0.0.20 rsa-4.9.1 six-1.17.0 sniffio-1.3.1 starlette-0.46.2 typing-inspection-0.
4.0 typing_extensions-4.13.2 uvicorn-0.34.2
Jun 23 08:40:11 PM
Jun 23 08:40:11 PM  [notice] A new release of pip is available: 24.0 -> 25.1.1
Jun 23 08:40:11 PM  [notice] To update, run: pip install --upgrade pip
Jun 23 08:40:13 PM  ==> Uploading build...
Jun 23 08:40:28 PM  ==> Uploaded in 11.4s. Compression took 3.7s
Jun 23 08:40:29 PM  ==> Build successful 🎉
```

```
POST    /register  Register User
POST    /login  Login User
POST    /upload/  Upload Gpx File
GET     /files/  Get Files
GET     /files/{file_id}  Download File
DELETE  /files/{file_id}  Delete File
GET     /summary/  Get Summary
GET     /  Read Root
```

Figure 11: API and framework server

11 Testing

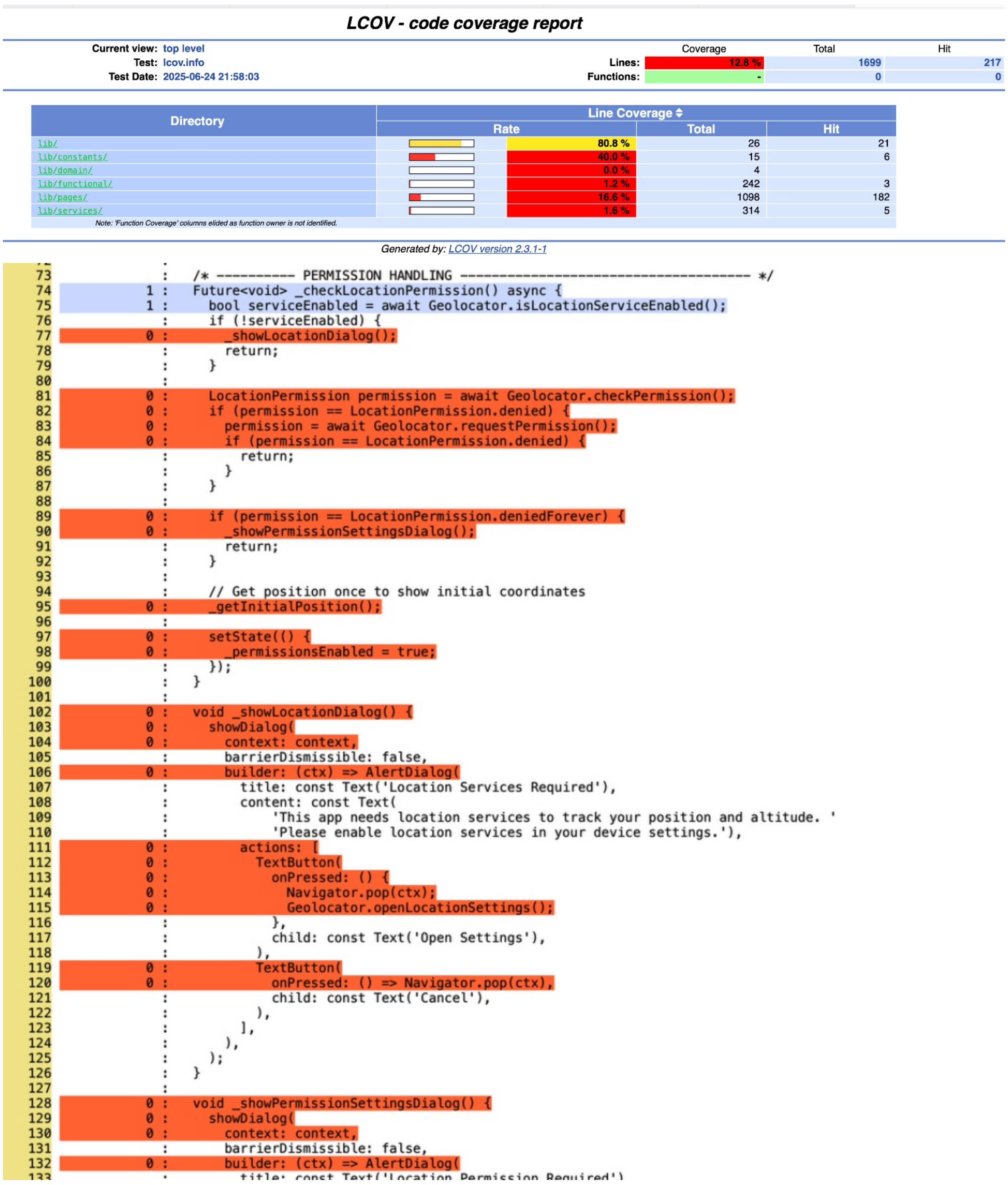
Test cases has been done in frontend testing with iOS, by downloading the TrackIN app in iPhone with XCode from Macbook. For the simplicity, only data management test cases has been shown, rest can be shown in Docs repository.



		Data management test case			
Test cases	Check altitude recording	Check distance recording	Simulate rapid GPS changes	Check timestamp consistency	Check GPX file format
Expected result	Altitude changes are accurately recorded	Distance accumulates as user moves	App filters noisy data or interpolates properly	All GPX points are timestamped correctly	GPX XML is well-formed and valid against schema
Actual result	Changes in every .24 second and .50 second	Evelation changing	Sudden peak in some GPS changes, overall stable	Correct	Need to remove prefix like : <ns:tag> -> <tag> and </ns:tag> -> </tag>

Figure 12: Test case for iOS/ 1 section

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				

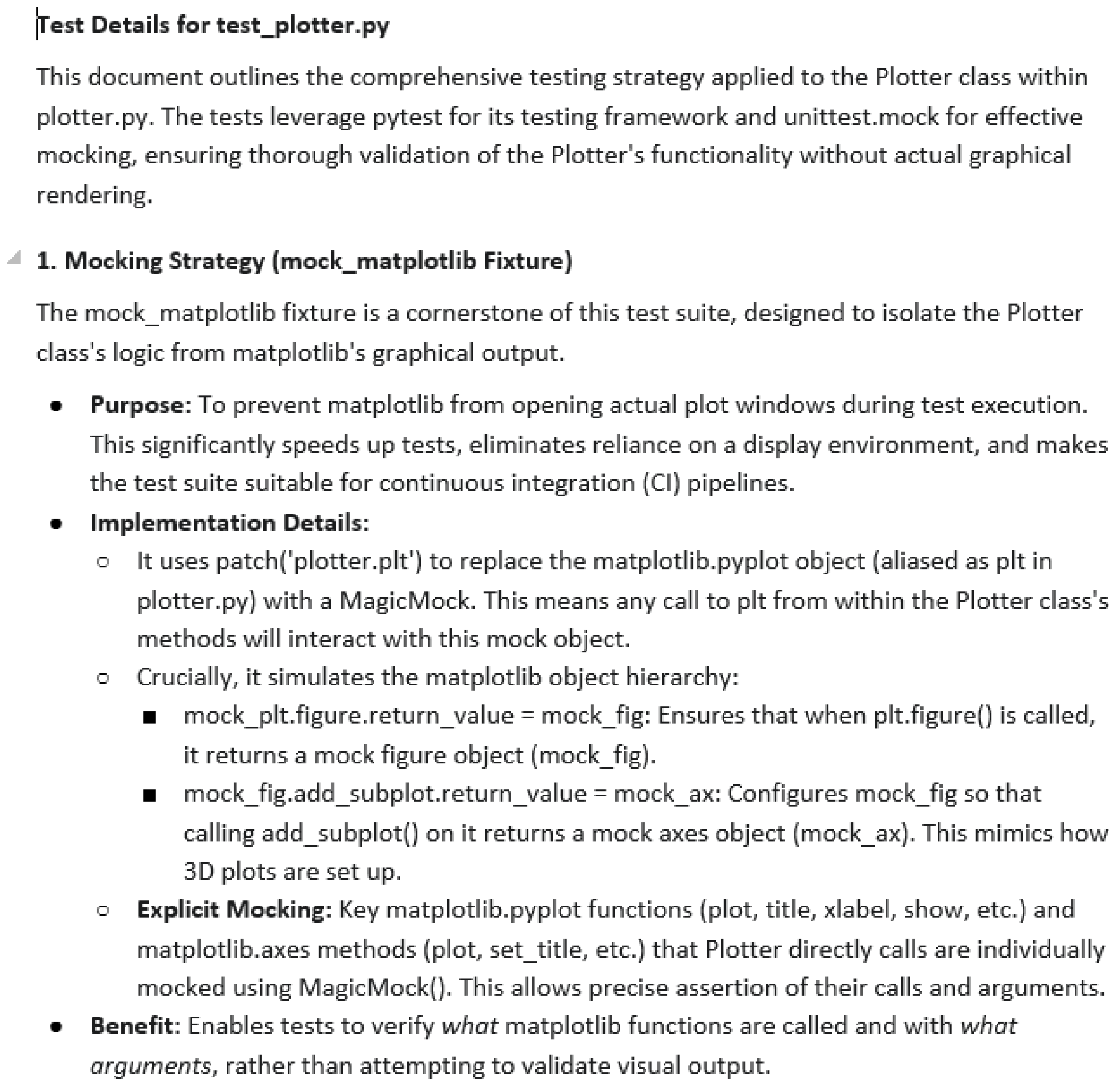
Directory	Line Coverage		
	Rate	Total	Hit
lib/	80.8 %	26	21
lib/constants/	40.0 %	15	6
lib/domain/	0.0 %	4	
lib/functional/	1.2 %	242	3
lib/pages/	16.6 %	1098	182
lib/services/	1.6 %	314	5

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 13: Coverage test for iOS



Test Details for 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.

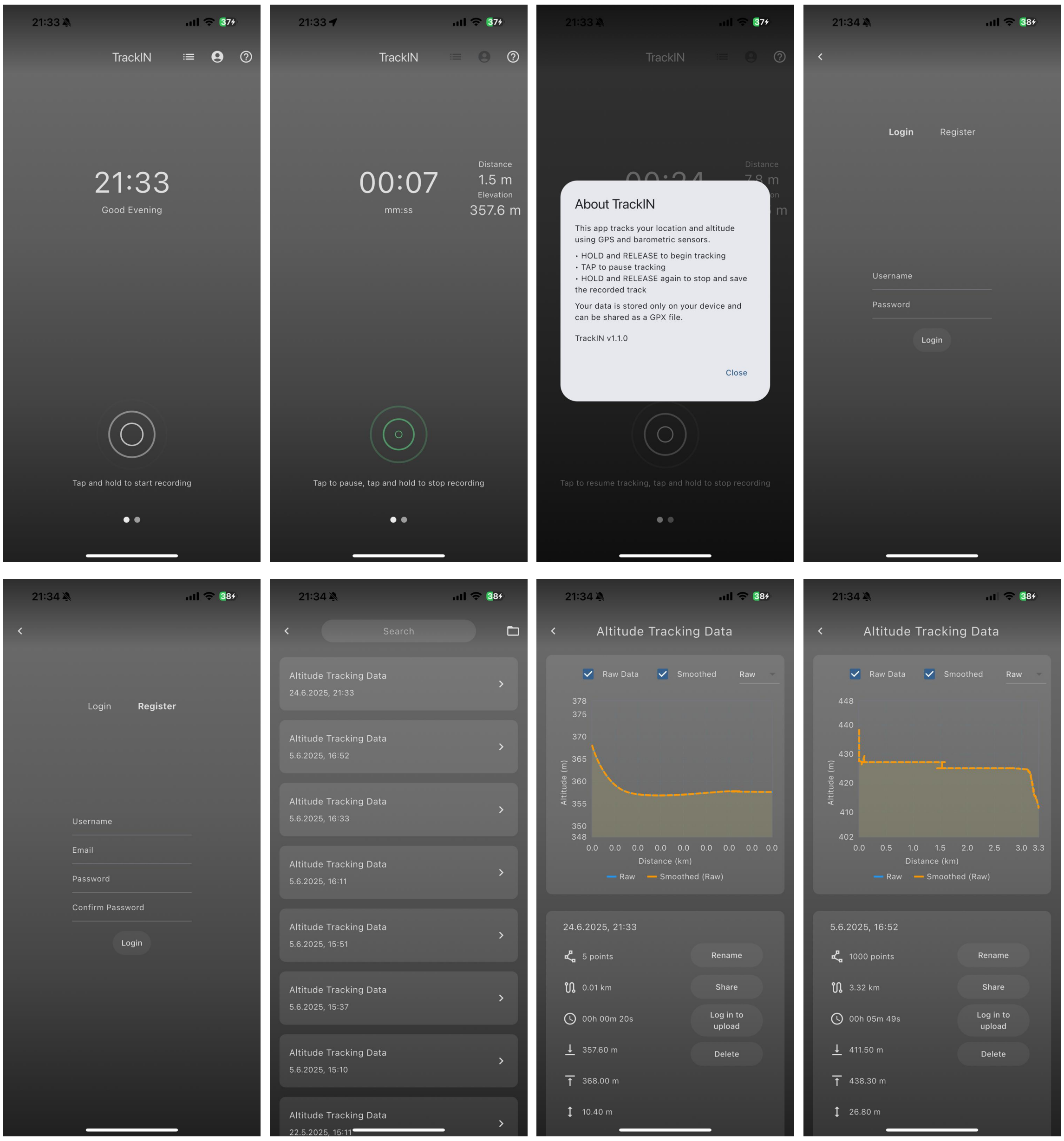
1. Mocking Strategy (mock_matplotlib Fixture)

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. This means any call to plt from within the Plotter class's methods will interact with this mock object.
  - 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). This mimics how 3D plots are set up.
  - **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(). This allows precise assertion of their calls and arguments.
- **Benefit:** Enables tests to verify *what* matplotlib functions are called and with *what arguments*, rather than attempting to validate visual output.

Figure 14: First slide of backend testing documentation

12 GUI



About TrackIN

This app tracks your location and altitude using GPS and barometric sensors.

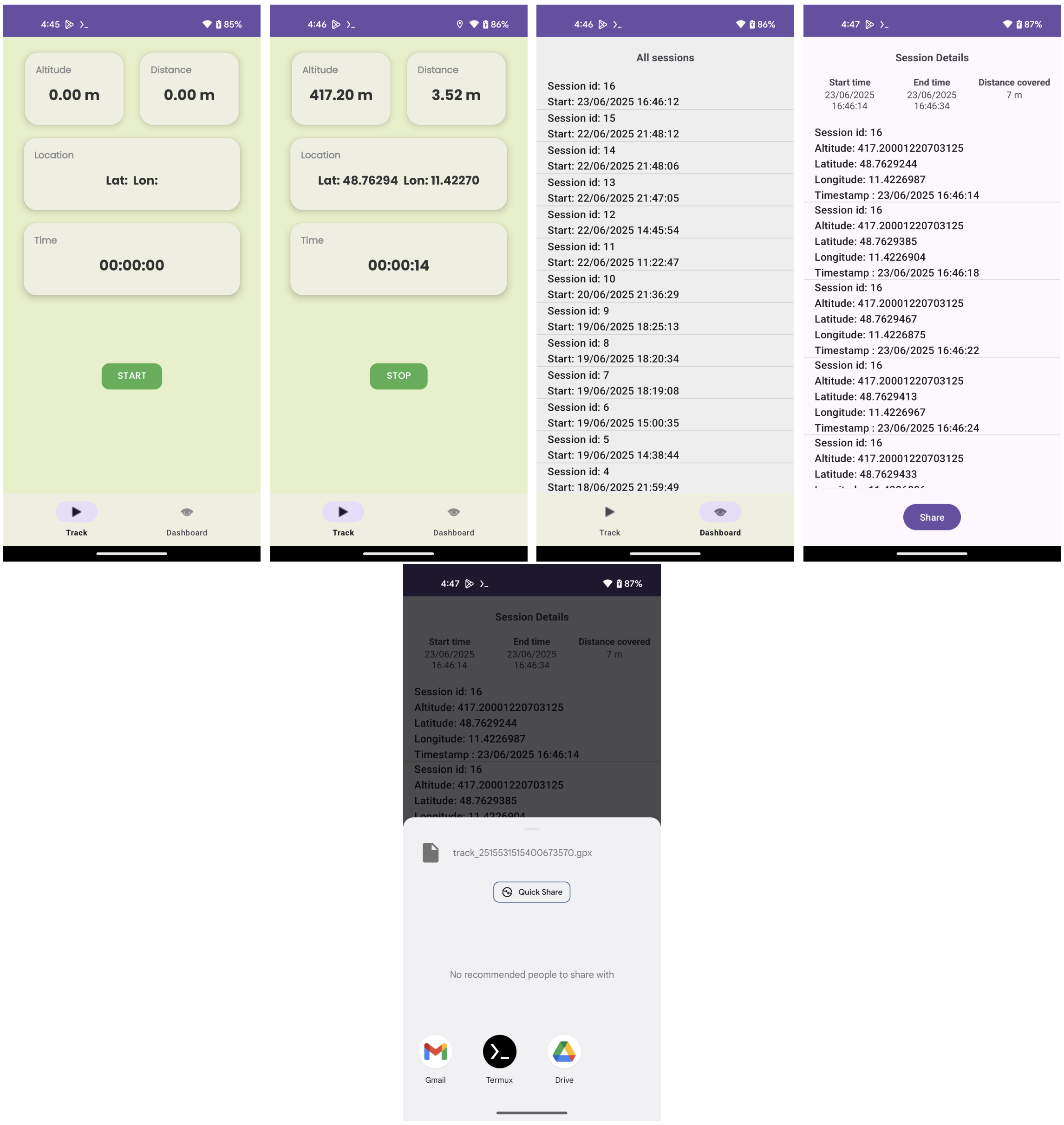
- HOLD and RELEASE to begin tracking
- TAP to pause tracking
- HOLD and RELEASE again to stop and save the recorded track

Your data is stored only on your device and can be shared as a GPX file.

TrackIN v1.1.0

Close

Figure 15: Flutter Application/ iOS and Andriod GUI



All sessions

Session id: 16
Start: 23/06/2025 16:46:12
Session id: 15
Start: 22/06/2025 21:48:12
Session id: 14
Start: 22/06/2025 21:48:06
Session id: 13
Start: 22/06/2025 21:47:05
Session id: 12
Start: 22/06/2025 14:45:54
Session id: 11
Start: 22/06/2025 11:22:47
Session id: 10
Start: 20/06/2025 21:36:29
Session id: 9
Start: 19/06/2025 18:25:13
Session id: 8
Start: 19/06/2025 18:20:34
Session id: 7
Start: 19/06/2025 18:19:08
Session id: 6
Start: 19/06/2025 15:00:35
Session id: 5
Start: 19/06/2025 14:38:44
Session id: 4
Start: 18/06/2025 21:59:49

Session Details

Start time	End time	Distance covered
23/06/2025 16:46:14	23/06/2025 16:46:34	7 m

Session id: 16
Altitude: 417.20001220703125
Latitude: 48.7629244
Longitude: 11.4226987
Timestamp : 23/06/2025 16:46:14
Session id: 16
Altitude: 417.20001220703125
Latitude: 48.7629385
Longitude: 11.4226904
Timestamp : 23/06/2025 16:46:18
Session id: 16
Altitude: 417.20001220703125
Latitude: 48.7629467
Longitude: 11.4226875
Timestamp : 23/06/2025 16:46:22
Session id: 16
Altitude: 417.20001220703125
Latitude: 48.7629413
Longitude: 11.4226967
Timestamp : 23/06/2025 16:46:24
Session id: 16
Altitude: 417.20001220703125
Latitude: 48.7629433

Share
4:47
87%
track_2515531515400673570.gpx
Quick Share
No recommended people to share with
Gmail
Termux
Drive

Figure 16: Andriod application GUI by Java

13 GitHub Repository Structure

TrackINJava

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

backend

```
backend/
|
|-- data/
|-- notebooks/
|-- src/
|-- test/
|
|-- .gitignore
|-- LICENSE
‘-- README.md
```

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
```

14 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.

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.