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 in combination with the flutter_foreground_task 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.

To facilitate **comparative analysis between recordings**, we developed two core backend algorithms:

1. **Track Alignment Function:**
   We implemented align_track_endpoints, 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 20% of 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 interpolate_to_match_points, 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 path_provider 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 altitude-focused GPS analysis and provide a solid foundation for the integration of rule-based or AI-assisted correction techniques in future work.