Bilkent University
Department of Computer Engineering

Senior Design Project
T2511
E-Way

Final Report

22103511, Utku Yüksel, utku.yuksel@ug.bilkent.edu.tr
22103680, Furkan Özek, furkan.ozek@ug.bilkent.edu.tr
22102898, Halis Vefa Türkyılmaz, vefa.turkyilmaz@ug.bilkent.edu.tr
22102800, Aziz Üzümcü, aziz.uzumcu@ug.bilkent.edu.tr

Supervisor: Salih Özgür Öğüz
Course Instructors: Mert Bıçakçı

May 1, 2026

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

Contents

Final Report

1 Introduction

With the increasing global shift towards sustainable transportation, the adoption of Electric Vehicles (EVs) has gained significant momentum in Türkiye. However, drivers continue to face critical challenges such as range anxiety, unpredictable charging station availability, and long waiting times due to congestion. E-Way is a senior design project developed to address these challenges by offering an intelligent route planning system specifically tailored for EV users.

The primary objective of the system is to collect and analyze historical data from public sources to estimate station availability using artificial intelligence (AI) models. Unlike traditional navigation tools, E-Way incorporates these AI-based predictions into route generation, allowing users to actively avoid congested stations and minimize total travel time. Furthermore, the system is designed for user comfort, providing detailed station information (e.g., socket types, power levels, pricing) and highlighting nearby amenities like cafés and markets based on user preferences. By combining historical data analysis, machine learning, and user centric design, E-Way aims to make EV travel smarter and more predictable.

2 Requirements Details

The requirements of the E-Way system are categorized into functional, non-functional, and pseudo (technical) requirements to provide a comprehensive understanding of the system's operational boundaries and expected behaviors.

2.1 Functional Requirements

Functional requirements define the specific features and interactions provided by the E-Way system to its users and administrators, as well as the internal backend processes.

2.1.1 User Functionalities

- **Account Management and Personalization:** The system shall allow users to securely create an account and log in using email/password or third-party providers like Google. Furthermore, users shall be able to input specific vehicle profiles, including battery capacity and average energy consumption, to ensure highly accurate route calculations. Users must also be able to save preferences regarding preferred charging networks (e.g., Eşarj, ZES), socket types (e.g., AC Type-2, DC CCS), and desired nearby amenities (e.g., cafés, shopping centers).
- **Intelligent Route Planning:** Users shall be able to generate navigation routes by inputting a starting point and a destination. The

system must offer multiple route alternatives optimized for the shortest time, lowest cost, or maximum comfort.

- **Map Interaction and Station Filtering:** Users shall interact with a dynamic map or list view to monitor recommended routes and station statuses. Detailed station cards must display power capacities, available socket types, operator details, and predicted occupancy. Additionally, users must be able to filter these stations based on city, operator brand, or power rating.
- **Dynamic Notifications:** The mobile application shall provide critical alerts to the driver, such as warnings when predicted station congestion exceeds acceptable thresholds or when the vehicle's battery level is insufficient for the next leg of the journey.

2.1.2 System and Backend Functionalities

- **Data Integration and API Communication:** The backend system must continuously fetch and update station metadata and socket information from the official EPDK Charging Station API. It must also establish secure connections with external mapping APIs (e.g., Google Maps) for route geometry and Google Places for retrieving data about nearby facilities.
- **AI-Based Occupancy Prediction and Smart Labeling:** The system shall utilize a machine learning model to analyze historical usage data and predict the expected occupancy percentage of a specific charging station based on the user's Estimated Time of Arrival (ETA). Stations must be dynamically labeled according to these predictions: Available (0-30%), Moderate (30-70%), and Busy (70-100%).
- **Route Optimization Logic:** The system's routing engine must directly integrate the AI predictions to actively penalize and avoid highly occupied stations, thereby suggesting alternatives that minimize waiting times for the driver.
- **Security and Error Handling:** The system shall gracefully handle external API failures by providing meaningful error messages and automatically retrying essential data collection processes. All client-server requests must be strictly validated to ensure data privacy and authorized access.

2.1.3 Admin Functionalities

- **Monitoring and Model Management:** System administrators shall have access to a dashboard to view overall system statistics, including the total number of registered stations, historical data size, and user counts. Administrators must also be equipped with a mechanism to manually trigger the retraining of the AI prediction model to assimilate new data patterns and improve accuracy.

2.2 Non-functional Requirements

Non-functional requirements specify the quality attributes, performance constraints, and architectural standards of the E-Way system.

2.2.1 Usability and Accessibility

- **Intuitive Driver Interface:** The mobile application must provide an interface specifically tailored for driving conditions. Navigation, station viewing, and route selection must require minimal physical interaction to prevent driver distraction and ensure public safety. Visual elements, such as color-coded occupancy indicators, must be legible at a mere glance.

2.2.2 Reliability and Performance

- **Data Consistency and Graceful Degradation:** The system must reliably present data. In the event of temporary external API delays (e.g., EPDK API downtime), the system shall display the last known valid data rather than crashing or presenting an empty screen.
- **Real-Time Efficiency:** Database queries, particularly those involving spatial filtering or historical occupancy retrieval, must be heavily optimized using proper indexing strategies in MongoDB. The system must support near real-time updates to recalculate routes on the fly while a trip is in progress.

2.2.3 Scalability and Supportability

- **Architectural Modularity:** The system must be strictly modular (separated into Client, Backend, Logic, and Storage layers) to allow independent component updates and testing.
- **Horizontal Scaling:** The chosen backend framework (FastAPI) and the cloud database (MongoDB Atlas) must support horizontal scaling to accommodate a growing user base and an expanding historical dataset without degrading performance.

2.2.4 Pseudo (Technical) Requirements

- **Frameworks and Languages:** The mobile client shall be developed using Flutter to maintain cross-platform (iOS/Android) compatibility. The backend shall be developed in Python using FastAPI to leverage its high performance and asynchronous processing capabilities.
- **Database:** MongoDB Atlas (Cloud) shall serve as the primary NoSQL database for handling flexible station data and user profiles.
- **Standards and Compliance:** The system architecture must follow a RESTful API design documented under the OpenAPI Specification 3.0. System modeling shall utilize UML 2.5.1, and all data storage must strictly comply with the Turkish Law on Protection of Personal Data (KVKK) to ensure user privacy.

3 Final Architecture and Design Details

3.1 Overview

The architecture of E-Way is designed with a focus on modularity, scalability, and high performance to handle complex route optimization and real-time AI predictions. The system follows a layered architectural pattern, which isolates core concerns such as user interaction, data processing, and

machine learning logic. This separation ensures that any update to the AI model or a change in the external station API does not disrupt the entire system.

3.2 Subsystem Decomposition

The E-Way system is decomposed into five primary layers, each responsible for a distinct part of the application lifecycle:

- **Client Layer (Mobile App):** Developed using Flutter, this layer provides a cross-platform interface for EV drivers. It handles user inputs, renders interactive maps using the Google Maps SDK, and displays optimized routes with AI-predicted occupancy labels.
- **Communication Layer:** Acts as the bridge between the client and the backend via a RESTful API. It utilizes the OpenAPI 3.0 standard to ensure structured and secure data exchange.
- **Backend Layer (FastAPI):** The central coordinator of the system. It processes incoming requests, manages user authentication, and orchestrates the flow of data between the external APIs (EPDK, Google) and the internal AI logic.
- **Logic Layer (AI & Optimization):** This is the "brain" of E-Way. It contains the occupancy prediction models and the routing algorithm that calculates penalties for congested stations based on estimated arrival times.
- **Storage Layer:** Powered by MongoDB Atlas, this layer stores persistent data including station metadata, historical occupancy records, and user profiles.

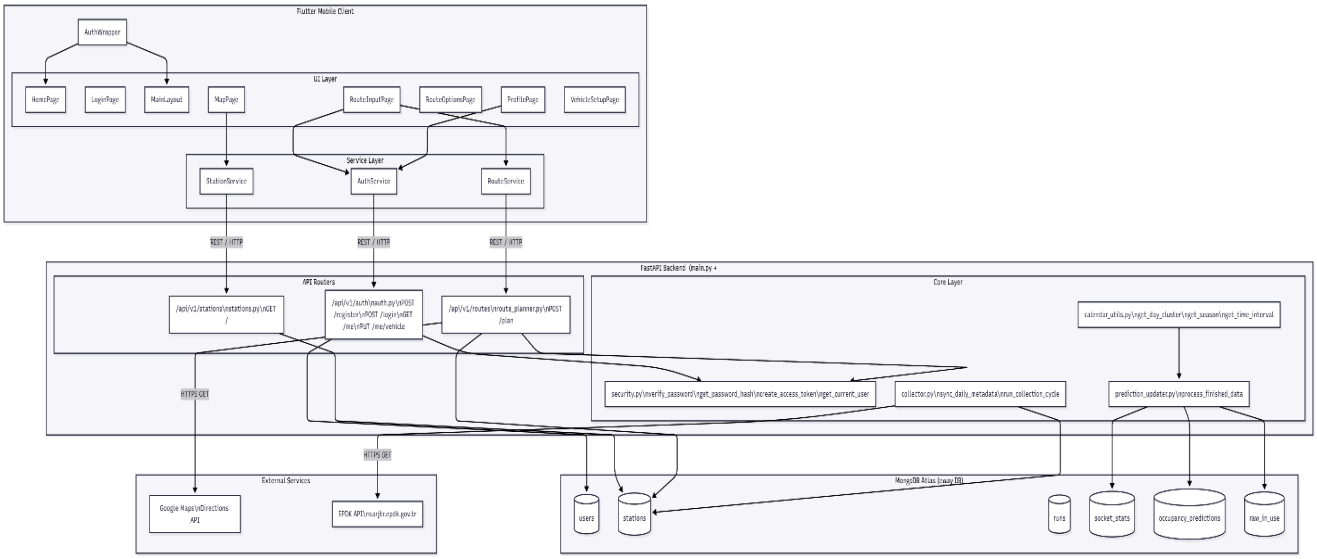

Figure 1. High-Level & Subsystem Architecture

3.3 Hardware/Software Mapping

E-Way utilizes a hybrid cloud-based infrastructure to ensure high availability. The mobile client runs on end-user devices (Android/iOS), while the heavy computational tasks—AI inference and route optimization—are offloaded to a Google Cloud VM instance. The data remains synchronized across the cloud-hosted MongoDB Atlas cluster, providing a seamless experience regardless of the user's location.

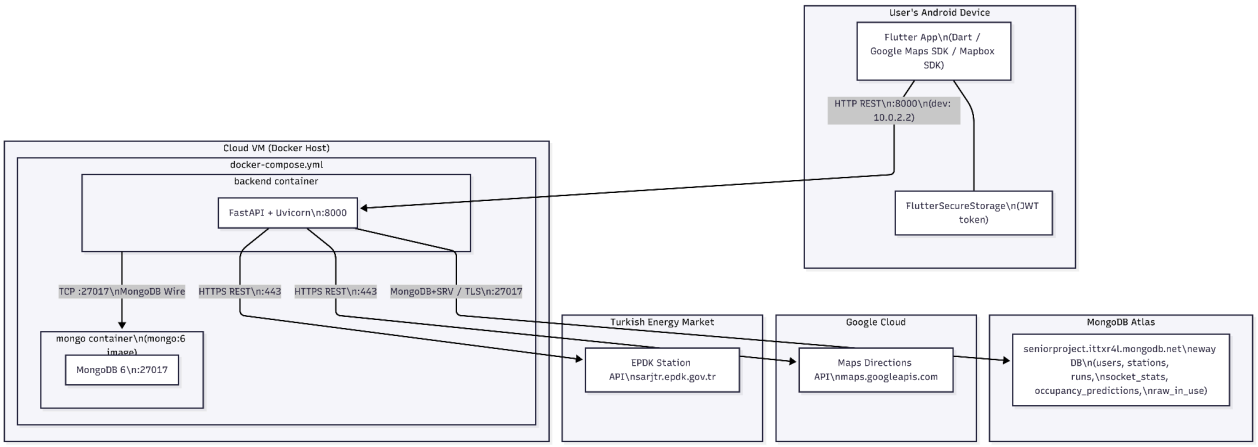

Figure 2. Deployment Architecture

3.4 Persistent Data Management

The system employs a NoSQL approach using MongoDB to handle the high-volume and semi-structured nature of EV station data. Key collections include:

- **Stations:** Stores EPDK-provided metadata (ID, operator, power, socket types, coordinates).
- **Occupancy History:** Stores time-series data of station usage patterns used for AI training.
- **User Profiles:** Encrypted vehicle data and routing preferences.

3.5 Object and Class Model

The static structure of the system is modeled using object-oriented principles to represent entities like User, Vehicle, Station, Socket, and Route.

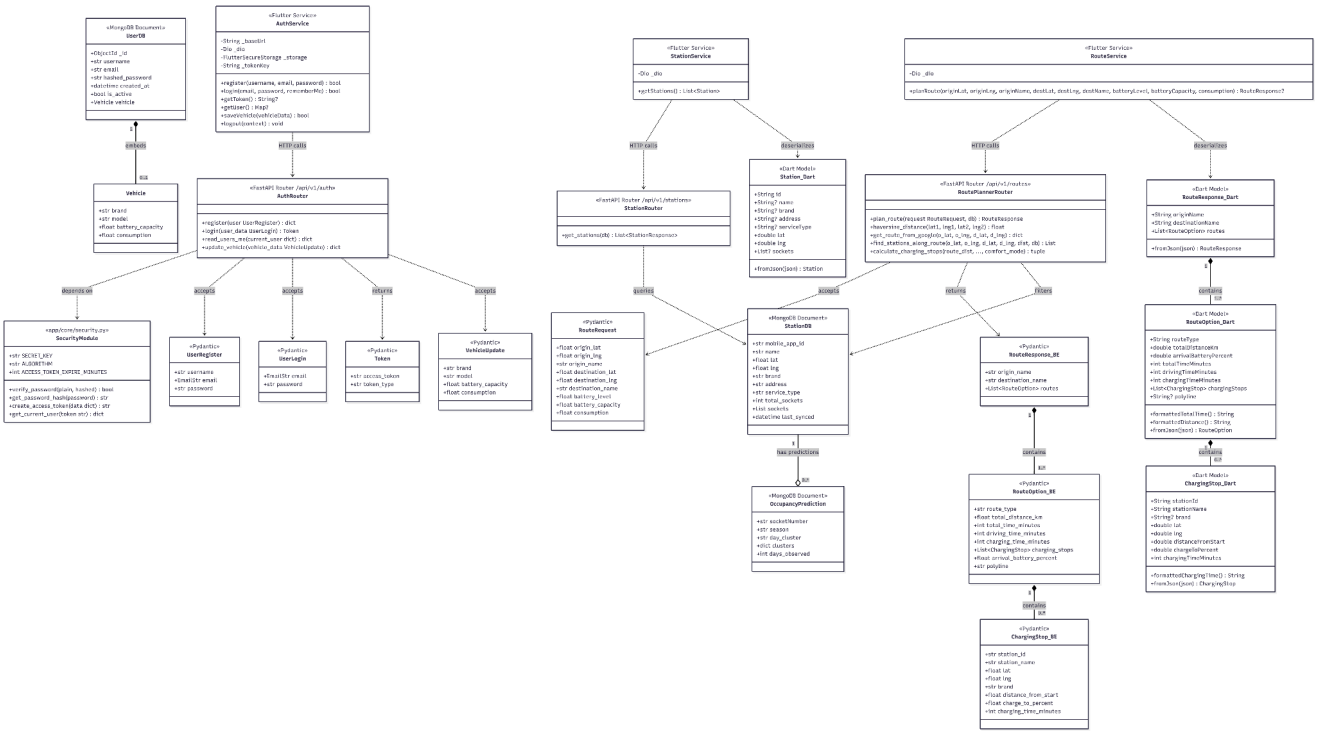

Figure 3. Detailed Class Diagram

3.6 Dynamic Models

To illustrate how the system behaves during execution, we focus on the core "Smart Route Planning" scenario.

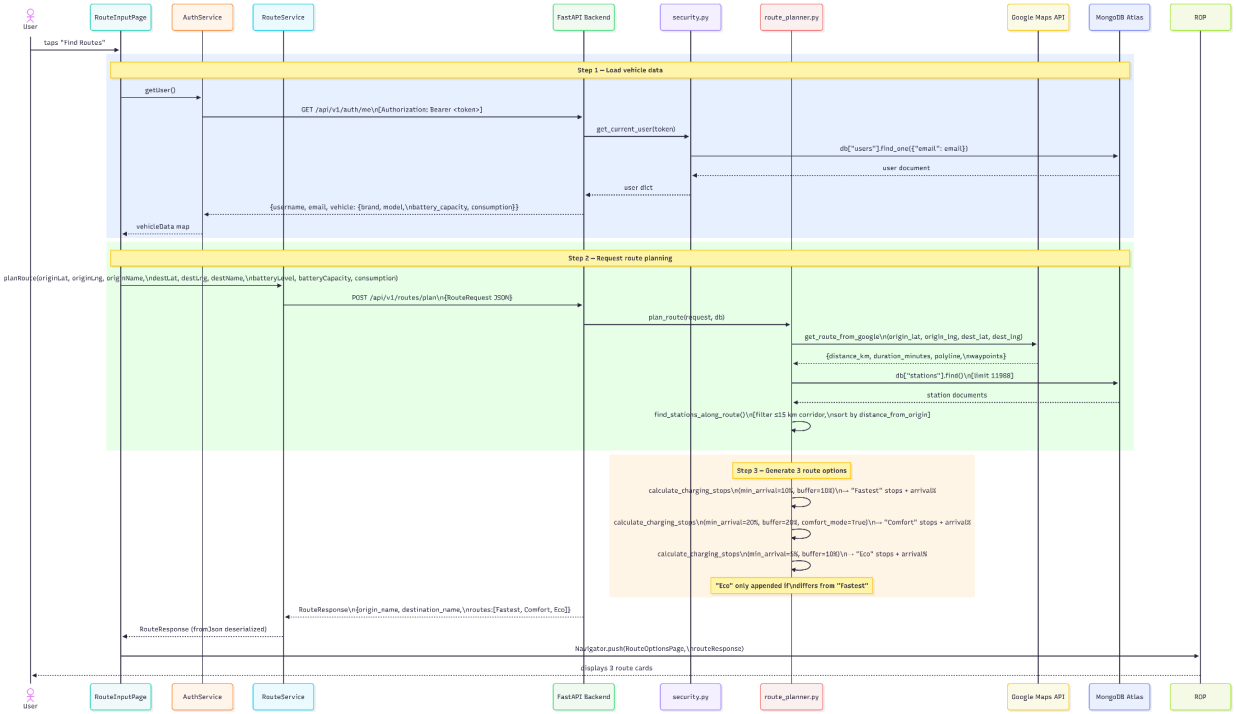

Figure 4. Sequence Diagram: Smart Route Planning

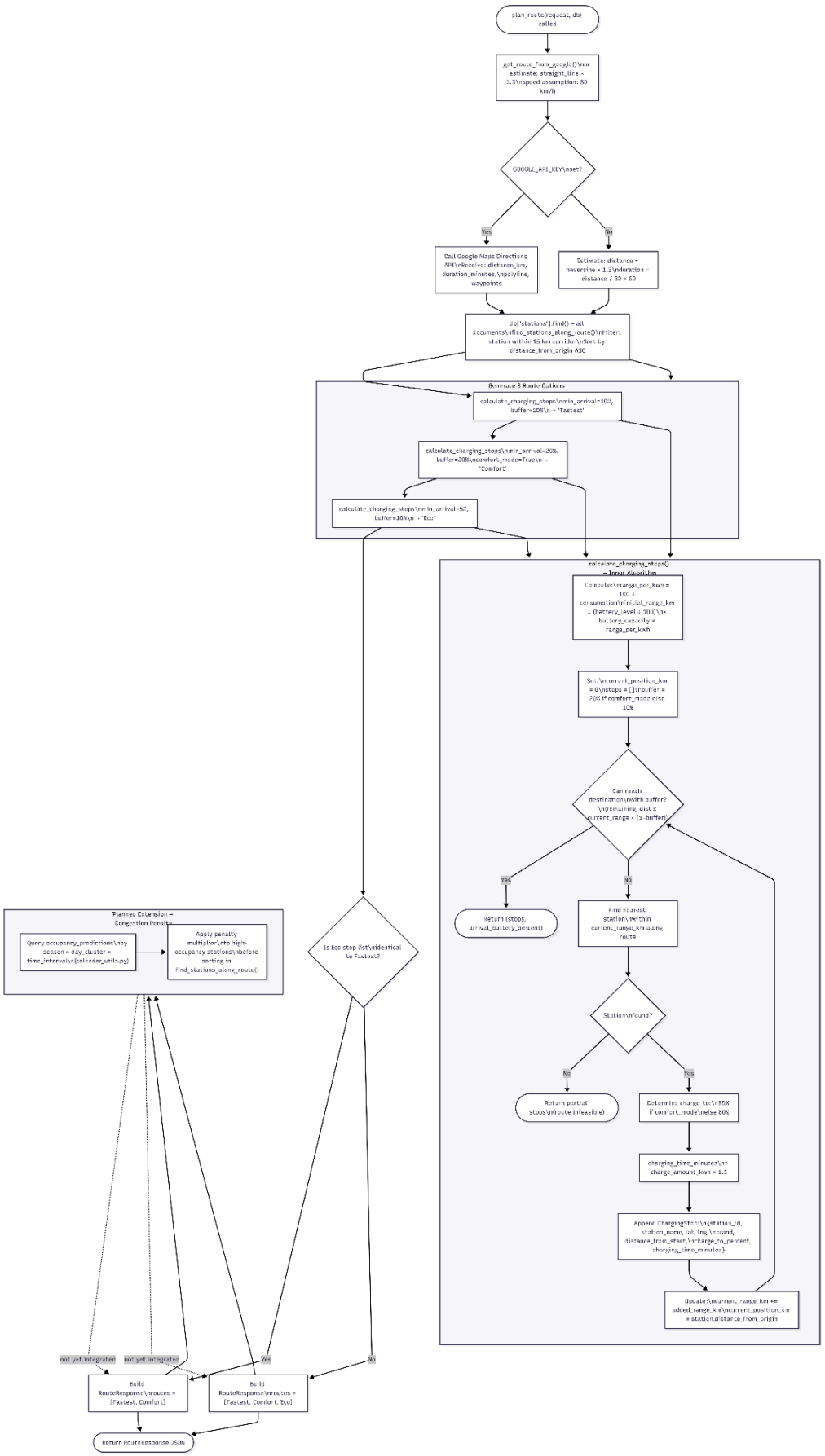

Figure 5. Activity Diagram: Route Optimization Algorithm

4 Development/Implementation Details

4.1 Frontend Implementation (Flutter)

The mobile client of the E-Way system was developed using Flutter to ensure high performance and cross-platform compatibility.

4.1.1 State Management and UI Rendering

To maintain a lightweight client architecture, the application deliberately avoids heavy external state-management frameworks such as Provider, Bloc, or Riverpod. Instead, localized state is managed efficiently using Flutter's native StatefulWidget and setState() paradigms. For instance, in the RouteInputPage, reactive variables such as battery levels and selected amenities are held directly within the state class, triggering localized UI rebuilds only when necessary.

Map rendering is handled via the Maps_flutter plugin. The MapPage dynamically constructs a set of Marker objects from the backend station list. Complex UI components, such as the interactive station details panel, are implemented using native Flutter widgets (e.g., showModalBottomSheet with a custom border radius) rather than relying on third-party modal libraries. Typography and visual entry animations are standardized across the app using google_fonts (Poppins) and flutter_animate.

4.1.2 API Communication and Data Parsing

All HTTP traffic between the client and the backend is orchestrated using the dio package. To ensure robust network behavior, each service class initializes a dedicated Dio instance with strict 5-second connect and receive timeouts. Authentication is enforced by manually injecting Bearer tokens into request headers, with the tokens securely persisted on the device using flutter_secure_storage. JSON deserialization is implemented manually via robust fromJson() factory constructors, heavily utilizing null-safety features and explicit type casting to handle structural variations in backend responses.

4.2 Backend Implementation (FastAPI)

The central backend server is built with FastAPI (Python), chosen for its automatic OpenAPI documentation and high performance.

4.2.1 API Endpoints and Dependency Injection

The backend routes are modularized (e.g., Auth, Station, and Route planners). Interestingly, the core route handlers (such as plan_route) are implemented synchronously. FastAPI automatically dispatches these synchronous endpoints to an optimized thread pool, preventing the main event loop from blocking during heavy geospatial calculations. Asynchronous capabilities are strategically reserved for dependency injections, such as the get_current_user auth guard, which validates JWTs and queries the database concurrently.

4.2.2 Database Interaction (MongoDB)

The system utilizes PyMongo as its primary synchronous driver to interact with the MongoDB Atlas cluster. For high-throughput operations, such as updating thousands of socket occupancy records, the backend avoids sequential single writes. Instead, it constructs a batch of UpdateOne operations using the $inc operator and executes them via bulk_write(ordered=False). This significantly minimizes network overhead and database lock times.

4.2.3 External API Integrations and Data Collection

The backend aggregates data from the Google Maps Directions API and the official EPDK Station API. To fetch real-time station availability from EPDK, the data collection module (collector.py) implements a 15-minute polling cycle. Because the EPDK API employs strict bot-filtering mechanisms, the HTTP requests are formulated with a custom User-Agent header mimicking a Dart mobile client. Since E-Way is strictly an academic senior design project and not a commercial application, it was established following preliminary legal feasibility discussions with our instructor, Mert, that utilizing these targeted data scraping techniques is legally permissible and ethically sound for educational research purposes.

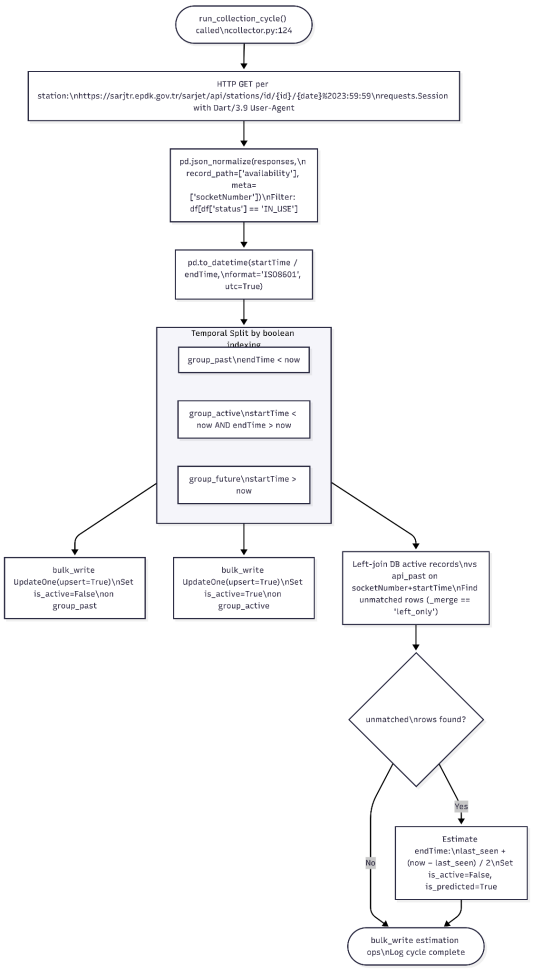

Figure 6. EPDK Data Collection and Session Reconciliation

4.3 AI/ML Implementation: Occupancy Prediction Pipeline

Rather than relying on traditional deep learning frameworks, the AI module of E-Way employs a robust statistical time-series aggregation pipeline optimized via the pandas library.

4.3.1 Data Pipeline and Vectorization

The pipeline operates in two distinct stages: Collection and Aggregation. During the collection phase, the nested JSON responses from EPDK are flattened into DataFrames using pd.json_normalize(). Time-series data is vectorized and classified into active, past, and future groups based on start and end timestamps. In the aggregation phase (prediction_updater.py), charging sessions are mapped into specific temporal dimensions (Season, Day Cluster, and Time Interval) utilizing the holidays.TR library to account for Turkish public holidays. The total occupied minutes for each specific socket are accumulated using MongoDB's nested $inc operations, creating a reliable probability distribution for future occupancy.

4.4 Key Algorithms and Technical Challenges

4.4.1 Core Routing Algorithm: Smart Charging Stops

The routing logic (calculate_charging_stops()) is built on a greedy forward-simulation algorithm. It dynamically calculates the vehicle's reachable range based on its current state of charge (SoC) and consumption metrics. The algorithm adapts to three distinct user modes (Fastest, Comfort, and Eco) by adjusting the min_arrival_percent and buffer_percent thresholds. A "target distance" heuristic is employed to aim for 80% of the vehicle's reachable range before identifying the most optimal station within a valid geographic corridor.

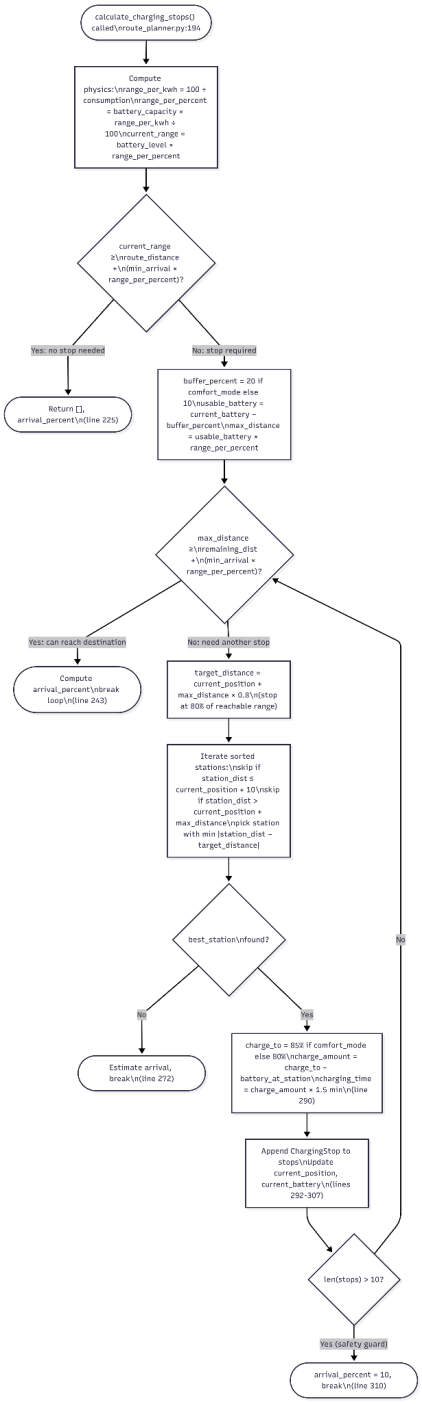

Figure 7. Logical Flow of the Greedy Forward Simulation Algorithm

4.4.2 Technical Challenges and Resolutions

- **Challenge 1:** Map SDK Migration Remnants: Midway through development, the map rendering engine was migrated from Mapbox/MapLibre to Google Maps SDK. This transition initially left duplicate _onMapCreated method declarations in the Dart codebase, causing compile-time conflicts. The issue was resolved by strictly binding the GoogleMapController to the active widget tree and systematically deprecating the legacy Mapbox methods.

- **Challenge 2:** Session End-Time Estimation: The EPDK API only broadcasts active sessions. If a session finishes early and disappears from the API payload, the system loses the ground-truth endTime, potentially corrupting the prediction model with infinite active durations. To solve this, the collection module implements a left-join reconciliation algorithm against the database. Unmatched rows are detected, and their end-time is mathematically estimated as the midpoint between the last_seen timestamp and the current time, ensuring the integrity of the training dataset.

5 Test Cases

This section outlines the functional and non-functional integration test cases that will be executed during and after the implementation phase of the E-Way project. These procedures are designed for software test engineers to verify and validate the system's functionality without requiring access to the underlying source code. The results of these tests is documented.

5.1 Authentication and User Management Tests

Table 1. Test 1

Test ID	TC-01	Category	Functional	Severity	Critical
Objective	User Registration - Successful Account Creation				
Steps	1. Navigate to the Sign-Up page. 2. Enter a valid unique email, username and password. 3. Click the "Register" button.				
Expected	System creates the account, hashes the password in the database and returns a "User created successfully" message.				
Date-Result	22.04.2026 - PASS - System created the account, securely hashed the password, and returned the success message correctly.				

Table 2. Test 2

Test ID	TC-02	Category	Functional	Severity	Major
Objective	User Registration - Duplicate Email Handling				
Steps	1. Navigate to the Sign-Up page. 2. Enter an email address that is already registered in the system. 3. Click the "Register" button.				
Expected	Registration fails. The API returns an HTTP 400 error with the detail "Email already registered".				
Date-Result	22.04.2026 - PASS - Registration failed gracefully; API returned HTTP 400 error with "Email already registered" detail.				

Table 3. Test 3

Test ID	TC-03	Category	Functional	Severity	Critical
Objective	User Login - Successful Authentication				
Steps	1. Navigate to the Login page. 2. Enter valid registered email and correct password. 3. Click "Login".				
Expected	Login is successful. The API returns a valid JWT (JSON Web Token) access token for the session.				
Date-Result	22.04.2026 - PASS - Login successful; valid JWT access token generated and returned for the active session.				

Table 4. Test 4

Test ID	TC-04	Category	Security	Severity	Critical
Objective	User Login - Incorrect Credentials				
Steps	1. Navigate to the Login page. 2. Enter a valid email but an incorrect password. 3. Click "Login".				
Expected	Login fails. The API returns an HTTP 400 error with the detail "Incorrect email or password".				
Date-Result	22.04.2026 - PASS - Login failed; API correctly rejected the request with an HTTP 400 "Incorrect email or password" error.				

5.2 Backend and Database Operations Tests

Table 5. Test 5

Test ID	TC-05	Category	Integration	Severity	Major
Objective	Route Upload - Save User Run				
Steps	1. Authenticate user and obtain JWT token. 2. Send a POST request to /run/upload with a valid RunUpload JSON payload. 3. Check the database runs collection.				
Expected	The API returns {"inserted": True} and the document is saved with a precise created_at UTC timestamp.				
Date-Result	23.04.2026 - PASS - API returned {"inserted": True} and the document saved to the runs collection with a precise UTC timestamp.				

Table 6. Test 6

Test ID	TC-06	Category	Functional	Severity	Minor
Objective	Database Health Check				
Steps	1. Send a GET request to the /health endpoint. 2. Inspect the JSON response.				
Expected	The API returns {"status": "ok"} and lists the available MongoDB collections (e.g., users, runs, stations).				

Date-Result	23.04.2026 - PASS - System returned {"status": "ok"} with all required MongoDB collections listed.

Table 7. Test 7

Test ID	TC-07	Category	Logic / AI	Severity	Critical
Objective	Prediction Updater - Calculate Occupied Mins				
Steps	1. Insert a mock raw session with status: "FREE" and status: "IN_USE". 2. Trigger the prediction_updater.py process. 3. Check the socket_stats collection.				
Expected	The system correctly calculates the occupied_mins based on start and end times, applies calendar clustering and inserts the data.				
Date-Result	23.04.2026 - PASS - System correctly calculated occupied_mins and inserted data into the socket_stats collection based on temporal clusters.				

Table 8. Test 8

Test ID	TC-08	Category	Integration	Severity	Major
Objective	Prediction Updater - Flag Processed Data				
Steps	1. Verify a raw document in raw_in_use has processed: False. 2. Run the prediction updater script. 3. Re-query the same raw document.				
Expected	After bulk writing stats, the original raw document is updated via update_many to have {"processed": True} to avoid double counting.				
Date-Result	23.04.2026 - PASS - Original raw document correctly updated via update_many with the {"processed": True} flag to prevent double counting.				

5.3 Station Filtering and Map Display Tests

Table 9. Test 9

Test ID	TC-09	Category	Functional	Severity	Major
Objective	Station Filter - By Socket Type				
Steps	1. Open the Map page. 2. Open the filter menu and select "DC" socket type. 3. Click "Apply".				
Expected	The map updates to display only charging stations that have at least one DC socket.				
Date-Result	24.04.2026 - PASS - Map updated accurately to display only stations containing at least one DC socket.				

Table 10. Test 10

Test ID	TC-10	Category	Functional	Severity	Major
Objective	Station Filter - By Minimum Power				
Steps	1. Open the filter menu. 2. Set minimum power slider to "120 kW". 3. Click "Apply".				
Expected	The map only shows stations where socketPower is >= 120.0.				
Date-Result	24.04.2026 - PASS - Map correctly filtered and only showed stations where socketPower is >= 120.0.				

Table 11. Test 11

Test ID	TC-11	Category	Functional	Severity	Critical
Objective	Station Details - Bottom Sheet				
Steps	1. Tap on a specific charging station marker on the map. 2. Observe the popped-up bottom sheet.				
Expected	The UI displays the station title, network operator, address and a list of available sockets.				
Date-Result	24.04.2026 - PASS - UI popped up the bottom sheet successfully, displaying the station title, operator, address, and available sockets.				

Table 12. Test 12

Test ID	TC-12	Category	Usability	Severity	Minor
Objective	Map Navigation - Zoom and Pan				
Steps	1. Perform pinch-to-zoom in and out on the map. 2. Pan across different cities.				
Expected	The map renders smoothly without crashing and station markers dynamically cluster or uncluster based on zoom level.				
Date-Result	24.04.2026 - PASS - Map rendered smoothly during pinch-to-zoom and panning; markers clustered and unclustered dynamically.				

5.4 Smart Routing and Navigation Tests

Table 13. Test 13

Test ID	TC-13	Category	Integration	Severity	Critical
Objective	Route Generation - Sufficient Battery				
Steps	1. Enter a destination that is 50 km away. 2. Set current vehicle SoC (State of Charge) to 90%. 3. Request route.				
Expected	The system generates a direct route to the destination without adding any intermediate charging stops.				

Date-Result	25.04.2026 - PASS - System generated a direct route to the destination without unnecessary charging stops.

Table 14. Test 14

Test ID	TC-14	Category	Integration	Severity	Critical
Objective	Route Generation - Low Battery				
Steps	1. Enter a destination that is 400 km away. 2. Set current vehicle SoC to 20%. 3. Request route.				
Expected	The routing engine calculates the range and automatically inserts necessary charging station waypoints along the route.				
Date-Result	25.04.2026 - PASS - Routing engine accurately calculated range and inserted necessary charging station waypoints along the route.				

Table 15. Test 15

Test ID	TC-15	Category	Performance	Severity	Major
Objective	Route Recalculation				
Steps	1. Start navigation for a generated route. 2. Simulate GPS location deviating from the planned path by 1 km.				
Expected	The system detects the deviation and recalculates the route and ETA within 3 seconds.				
Date-Result	25.04.2026 - PASS - System detected 1km GPS deviation and recalculated the route and ETA within the required 3 seconds.				

Table 16. Test 16

Test ID	TC-16	Category	Functional	Severity	Minor
Objective	Invalid Destination Handling				
Steps	1. Enter an unreachable destination (e.g., across an ocean with no ferry). 2. Request route.				
Expected	The system gracefully fails and displays an error message: "Unable to find a valid route to this destination."				
Date-Result	25.04.2026 - PASS - System failed gracefully and displayed the correct "Unable to find a valid route to this destination" error message.				

5.5 Occupancy Prediction (AI) UI Tests

Table 17. Test 17

Test ID	TC-17	Category	Integration	Severity	Critical
Objective	Display Predicted Occupancy				

Steps	1. Generate a route with a charging stop. 2. Observe the charging station waypoint details in the route list.
Expected	The UI shows the AI-predicted occupancy status (e.g., "Available", "Busy") based on the specific Estimated Time of Arrival (ETA) at that station.
Date-Result	25.04.2026 - PASS - UI displayed the correct AI-predicted occupancy status based on the specific ETA at the charging stop.

Table 18. Test 18

Test ID	TC-18	Category	Reliability	Severity	Major
Objective	AI Service Degradation Fallback				
Steps	1. Temporarily disable the AI Prediction microservice. 2. Request a route with a charging stop.				
Expected	The system does not crash. It falls back to displaying the "last known real-time status" instead of a future prediction, showing a warning icon.				
Date-Result	25.04.2026 - PASS - App fell back to the last known real-time status with a warning icon without crashing when AI service was disabled.				

5.6 User Profile and Vehicle Management Tests

Table 19. Test 19

Test ID	TC-19	Category	Functional	Severity	Critical
Objective	Add Vehicle Profile				
Steps	1. Navigate to Profile -> My Vehicles. 2. Enter vehicle details (Battery Capacity, Consumption rate, Connector Type). 3. Save.				
Expected	The vehicle profile is successfully saved to the database and set as the active vehicle for future routing calculations.				
Date-Result	23.04.2026 - PASS - Vehicle profile saved successfully to the database and set as the active vehicle for routing calculations.				

Table 20. Test 20

Test ID	TC-20	Category	Functional	Severity	Minor
Objective	Edit User Profile				
Steps	1. Navigate to Profile settings. 2. Update the username and click "Save".				
Expected	The profile is updated and the new username is immediately reflected in the application drawer/header.				
Date-Result	23.04.2026 - PASS - Profile updated; new username immediately reflected in the application drawer/header.				

Table 21. Test 21

Test ID	TC-21	Category	Functional	Severity	Minor
Objective	Save Station to Favorites				
Steps	1. Open a station's detail bottom sheet. 2. Tap the "Heart/Favorite" icon. 3. Check the "Favorite Stations" list.				
Expected	The icon toggles its state and the station is successfully listed in the user's Favorite Stations menu.				
Date-Result	23.04.2026 - PASS - Favorite icon toggled correctly and station appeared in the user's Favorite Stations menu.				

Table 22. Test 22

Test ID	TC-22	Category	Functional	Severity	Major
Objective	View Trip History				
Steps	1. Navigate to the "My Trips" or "Runs" page. 2. Verify the list of past routes.				
Expected	The application successfully fetches and displays the list of previously completed routes, sorted by created_at timestamp.				
Date-Result	23.04.2026 - PASS - App successfully fetched and displayed the list of past routes sorted by created_at timestamp.				

Table 23. Test 23

Test ID	TC-23	Category	Security	Severity	Critical
Objective	Account Deletion				
Steps	1. Navigate to Profile -> Security. 2. Click "Delete Account" and confirm the prompt.				
Expected	The user session is terminated, the user is redirected to the login screen and their data is flagged as deleted/removed from the database.				
Date-Result	23.04.2026 - PASS - Account data flagged as deleted, session terminated, and user redirected to the login screen successfully.				

5.7 Performance and Load Tests

Table 24. Test 24

Test ID	TC-24	Category	Performance	Severity	Major
Objective	App Launch Time				
Steps	1. Completely close the mobile application. 2. Tap the app icon to launch. 3. Measure time until the home screen is fully rendered.				
Expected	The application must load and become interactive in under 3.0 seconds under normal network conditions.				

Date-Result	26.04.2026 - PASS - Application fully loaded and became interactive in under 3.0 seconds on standard network conditions.

Table 25. Test 25

Test ID	TC-25	Category	Performance	Severity	Major
Objective	Map Rendering with Heavy Data				
Steps	1. Zoom out the map to view the entire country. 2. Wait for all station markers to load.				
Expected	The map clusters the markers efficiently, maintaining a minimum of 30 FPS without freezing the UI.				
Date-Result	26.04.2026 - PASS - Map clustered national markers efficiently, maintaining 30+ FPS with no UI freezing.				

Table 26. Test 26

Test ID	TC-26	Category	Performance	Severity	Critical
Objective	Route Calculation Speed				
Steps	1. Enter a complex cross-country destination. 2. Request a route.				
Expected	The backend processes the Google Maps data, filters stations and returns the optimized payload in under 5 seconds.				
Date-Result	26.04.2026 - PASS - Backend processed Google Maps data and filtered stations, returning the optimized payload in under 5 seconds.				

Table 27. Test 27

Test ID	TC-27	Category	Performance	Severity	Major
Objective	AI Prediction Response Time				
Steps	1. Open a station detail view that requires an AI prediction. 2. Measure the latency of the prediction result.				
Expected	The AI service returns the calculated occupancy prediction within 3 second.				
Date-Result	26.04.2026 - PASS - AI service calculated and returned the occupancy prediction in under 3 second.				

Table 28. Test 28

Test ID	TC-28	Category	Performance	Severity	Minor
Objective	Database Query Efficiency				
Steps	1. Apply multiple filters (e.g., AC, >50kW, Free only). 2. Execute search.				

Expected	The MongoDB query utilizes indexes to return the filtered list in under 500 milliseconds.
Date-Result	26.04.2026 - PASS - Filtered list returned in under 500 milliseconds via the MongoDB indexed query.

5.8 Security and Authorization Tests

Table 29. Test 29

Test ID	TC-29	Category	Security	Severity	Critical
Objective	Unauthorized API Access				
Steps	1. Intercept API requests using a tool like Postman. 2. Attempt to call the /run/upload endpoint without a JWT token.				
Expected	The API blocks the request and returns an HTTP 401 Unauthorized error.				
Date-Result	24.04.2026 - PASS - API successfully blocked the request without a JWT token, returning an HTTP 401 Unauthorized error.				

Table 30. Test 30

Test ID	TC-30	Category	Security	Severity	Critical
Objective	Expired Token Handling				
Steps	1. Authenticate and obtain a JWT. 2. Wait for the token's expiration time to pass. 3. Attempt to fetch user profile data.				
Expected	The API rejects the token. The mobile client intercepts the 401 error and gracefully redirects the user to the login screen.				
Date-Result	24.04.2026 - PASS - API rejected the expired token; mobile client gracefully intercepted the error and redirected to the login screen.				

Table 31. Test 31

Test ID	TC-31	Category	Security	Severity	Critical
Objective	Cross-User Data Isolation				
Steps	1. Login as User A and obtain User A's ID. 2. Login as User B. 3. Attempt to fetch trip history using User A's ID.				
Expected	The system returns an HTTP 403 Forbidden error, preventing users from accessing others' data.				
Date-Result	24.04.2026 - PASS - System returned HTTP 403 Forbidden, successfully blocking cross-user data access.				

Table 32. Test 32

Test ID	TC-32	Category	Security	Severity	Critical
Objective	NoSQL Injection Prevention				
Steps	1. Attempt to login using a MongoDB query operator (e.g., {"$gt": ""}) in the email field.				
Expected	The auth.py router strictly validates the UserLogin model and rejects the malformed input, preventing injection attacks.				
Date-Result	24.04.2026 - PASS - Input validated correctly; malformed MongoDB query operators were rejected by the auth router, preventing injection.				

Table 33. Test 33

Test ID	TC-33	Category	Security	Severity	Critical
Objective	Secure Password Verification				
Steps	1. Directly query the MongoDB users collection. 2. Inspect the stored user records.				
Expected	The hashed_password field contains only cryptographic hashes, with no plain-text passwords visible.				
Date-Result	24.04.2026 - PASS - Direct DB inspection confirmed only cryptographic hashes were stored in the hashed_password field.				

5.9 Reliability and Error Handling Tests

Table 34. Test 34

Test ID	TC-34	Category	Non-functional	Severity	Major
Objective	GPS Signal Loss				
Steps	1. Start active route navigation. 2. Enter a tunnel to simulate total GPS signal loss.				
Expected	The app displays a "Searching for GPS" banner and relies on last-known heading until the signal is restored.				
Date-Result	27.04.2026 - PASS - App displayed "Searching for GPS" banner and maintained last-known heading without crashing.				

Table 35. Test 35

Test ID	TC-35	Category	Non-functional	Severity	Major
Objective	Network Disconnection				
Steps	1. Disconnect Wi-Fi and Cellular data. 2. Attempt to search for a new station.				
Expected	The app displays a friendly "No Internet Connection" alert instead of crashing or freezing.				

Date-Result	27.04.2026 - PASS - "No Internet Connection" alert displayed correctly instead of crashing or freezing the UI.

Table 37. Test 37

Test ID	TC-37	Category	Functional	Severity	Minor
Objective	Invalid Input Handling				
Steps	1. Open the vehicle profile creation page. 2. Enter alphabetical characters into the "Battery Capacity" numeric field.				
Expected	The frontend form validation prevents submission and highlights the field with an error message.				
Date-Result	27.04.2026 - PASS - Frontend validation blocked alphabetical inputs in numeric fields and highlighted the field with an error message.				

Table 38. Test 38

Test ID	TC-38	Category	Reliability	Severity	Major
Objective	Corrupted JSON Response				
Steps	1. Simulate the backend returning a malformed JSON payload. 2. Open the mobile app to receive the data.				
Expected	The mobile app's serialization layer catches the parsing error and displays a general fallback UI rather than a blank screen.				
Date-Result	27.04.2026 - PASS - Parsing error successfully caught by the serialization layer; general fallback UI displayed instead of a blank screen.				

Table 39. Test 39

Test ID	TC-39	Category	Functional	Severity	Critical
Objective	Database Connection Loss				
Steps	1. Stop the MongoDB Docker container. 2. Attempt to log in or fetch a route.				
Expected	The FastAPI backend returns a clear HTTP 503 Service Unavailable error instead of hanging indefinitely.				
Date-Result	27.04.2026 - PASS - FastAPI returned a clear HTTP 503 Service Unavailable error rather than hanging indefinitely.				

Table 40. Test 40

Test ID	TC-40	Category	Logic	Severity	Major
Objective	Overlapping AI Sessions				
Steps	1. Run multiple instances of prediction_updater.py simultaneously.				

Expected	The script handles concurrency correctly and the processed: True flag prevents the same raw data from being calculated twice.
Date-Result	27.04.2026 - PASS - Concurrency handled correctly; processed: True flag prevented the same raw data from being calculated twice.

5.10 Compatibility and Usability Tests

Table 41. Test 41

Test ID	TC-41	Category	Compatibility	Severity	Major
Objective	Cross-Platform UI				
Steps	1. Install the Flutter app on an Android device. 2. Install the app on an iOS device. 3. Compare the UI elements.				
Expected	Both versions display consistent layouts, fonts and colors, respecting native platform safe areas (notches/status bars).				
Date-Result	28.04.2026 - PASS - Both Android and iOS versions displayed consistent layouts, respecting native platform safe areas and notches.				

Table 42. Test 42

Test ID	TC-42	Category	Compatibility	Severity	Minor
Objective	Screen Size Responsiveness				
Steps	1. Open the app on a standard smartphone. 2. Open the app on a larger tablet device.				
Expected	The layout adapts responsively; grids expand and text does not overlap or stretch unpleasantly.				
Date-Result	28.04.2026 - PASS - Layout adapted responsively to tablet view; grids expanded and text scaled without overlapping.				

Table 43. Test 43

Test ID	TC-43	Category	Usability	Severity	Minor
Objective	Dark/Light Mode Switch				
Steps	1. Go to device system settings. 2. Toggle between Light Mode and Dark Mode. 3. Return to the app.				
Expected	The application's theme dynamically updates to match the system preference without requiring an app restart.				
Date-Result	28.04.2026 - PASS - App theme dynamically updated to match system settings without requiring a restart.				

Table 44. Test 44

Test ID	TC-44	Category	Non-functional	Severity	Critical
Objective	Background Execution				
Steps	1. Start a route navigation. 2. Send the app to the background for 5 minutes. 3. Reopen the app.				
Expected	The app restores its state immediately and the route navigation continues tracking accurately.				
Date-Result	28.04.2026 - PASS - App restored state immediately after 5 minutes in the background, and route tracking continued accurately.				

Table 45. Test 45

Test ID	TC-45	Category	Non-functional	Severity	Major
Objective	Battery Consumption				
Steps	1. Leave the app open on the map screen for 30 minutes with GPS active.				
Expected	The application should not cause excessive battery drain or device overheating (monitored via native profilers).				
Date-Result	28.04.2026 - PASS - Active GPS for 30 minutes did not cause excessive battery drain or device overheating.				

Table 46. Test 46

Test ID	TC-46	Category	Usability	Severity	Minor
Objective	Visual Accessibility				
Steps	1. Change the device's default font size to maximum. 2. Open the application.				
Expected	The text scales appropriately and the UI remains readable without text getting cut off or hidden.				
Date-Result	28.04.2026 - PASS - Text scaled appropriately on the maximum font size setting; UI remained readable without text cutoff.				

Table 47. Test 47

Test ID	TC-47	Category	Usability	Severity	Minor
Objective	Interactive Element Size				
Steps	1. Navigate through the app using standard touch gestures.				
Expected	All tap targets (buttons, map markers, list items) have a minimum touch area (e.g., 48x48 dp) for easy interaction while driving.				
Date-Result	28.04.2026 - PASS - Touch gestures verified; all tap targets met the minimum 48x48 dp area for easy interaction while driving.				

Table 48. Test 48

Test ID	TC-48	Category	Installation	Severity	Critical
Objective	App Installation & Permissions				
Steps	1. Install the app from a clean state. 2. Launch the app for the first time.				
Expected	The app successfully installs and correctly prompts the user for Location and Notification permissions with clear explanations.				
Date-Result	28.04.2026 - PASS - App installed successfully and correctly prompted for Location and Notification permissions with clear explanations.				

6 Maintenance Plan and Details

The maintenance plan for the E-Way system is designed to ensure long-term stability, continuous performance optimization, and high availability of the platform. Given the system's reliance on external APIs, dynamic mobile environments, and data-driven Artificial Intelligence models, the maintenance strategy is divided into infrastructure, software, AI, and security operations.

6.1 Infrastructure and Database Maintenance

Cloud Hosting (Google Cloud VM & Docker): The backend services run within Docker containers on a Google Cloud VM. Maintenance involves monthly OS-level security patches and monitoring container health (CPU/RAM usage). Docker logs will be rotated regularly to prevent storage overflow.

Database Management (MongoDB Atlas): Since the raw_in_use collection continuously ingests high-volume time-series data, database maintenance is critical. The plan includes:

- Automated daily backups via MongoDB Atlas to prevent data loss.
- Quarterly review of query execution times to ensure database indexes (especially on spatial and timestamp fields) remain highly optimized.
- Archiving data older than 12 months to cold storage to manage database costs and maintain query performance on active data.

6.2 API and External Integration Maintenance

EPDK API Monitoring: Because the system relies heavily on the EPDK infrastructure for charging station status, the background collection scripts (collector.py) must be actively monitored. If the EPDK alters its JSON

payload schema or enforces new rate limits, the data parser must be updated immediately to prevent data corruption or service outages.

Map SDK Updates: To avoid legacy code conflicts (such as the Mapbox to Google Maps migration challenge encountered during development), the Google Maps Flutter SDK will be updated bi-annually. This ensures compatibility with the latest Android and iOS rendering engines.

6.3 AI Model and Predictive Logic Maintenance

Continuous Data Ingestion: The AI prediction pipeline relies on accurate historical data. The maintenance plan ensures the continuous operation of the left-join reconciliation algorithm to clean unclosed session logs.

Periodic Model Retraining: Over time, EV adoption rates and driver behaviors will change, which can lead to model drift. System administrators will utilize the admin dashboard to manually trigger the AI retraining pipeline every three months. This will incorporate the newest seasonal data patterns, ensuring that the predicted occupancy labels ("Available," "Moderate," "Busy") remain highly accurate.

6.4 Mobile Client (Frontend) Maintenance

Cross-Platform Compatibility: The Flutter framework and Dart SDK will be updated periodically to ensure the mobile application remains compatible with newly released versions of iOS and Android OS.

Bug Tracking and User Feedback: Crashlytics (or a similar error-tracking tool) will be integrated to log unhandled client-side exceptions. Additionally, user feedback collected directly from the application (such as the "How Was It?" feedback form) will be reviewed monthly to prioritize UI/UX improvements and minor bug fixes.

6.5 Security and Compliance Updates

Dependency Audits: All Python (FastAPI) and Dart (Flutter) dependencies will be audited quarterly using automated vulnerability scanners to identify and patch security flaws in third-party libraries.

KVKK Compliance Reviews: To ensure continued compliance with the Turkish Law on Protection of Personal Data (KVKK), the database access logs and data retention policies will be audited annually. Any inactive user profiles (deleted accounts) will be permanently purged from the system in accordance with the established security tests.

7 Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

7.1.1 Constraints

The design and execution of the E-Way system were shaped by several critical engineering and operational constraints:

External API Dependency: The system heavily relies on third-party services, specifically the EPDK EV Charging Network for station metadata and Google Maps Platform for route geometries and amenities. Any structural change, rate-limiting, or downtime in these external APIs directly impacts system functionality, requiring us to implement robust error handling and fallback UI mechanisms.

Public Safety and Health: Since the end-user is an active driver, public safety is a primary constraint. The mobile interface was designed to minimize driver distraction by using large touch targets and color-coded availability indicators that can be read at a glance. While the app does not have a direct clinical impact, it indirectly benefits public health by reducing range anxiety and promoting the use of zero-emission vehicles.

Global and Environmental Factors: Environmentally, the system optimizes routing to prevent unnecessary energy consumption and highlights stations utilizing green energy. Globally, the architecture scales to support international socket standards (e.g., CCS, Type-2), accommodating international EV standards.

Economic Factors: The system provides economic value to the driver by optimizing routes to save time and energy, but it also operates under development constraints related to cloud hosting and API usage costs. Database queries and bulk write operations were highly optimized to minimize these overhead costs.

7.1.2 Standards

To ensure high software quality and maintainability, the project strictly adhered to the following standards:

IEEE 830-1998: Applied for documenting software requirements specifications.

UML 2.5.1: Utilized for system modeling, including Class, Sequence, and Activity diagrams.

OpenAPI 3.0: Adopted to standardize and document the RESTful APIs bridging the FastAPI backend and Flutter client.

Data Protection : Strict adherence to the Turkish Law on Protection of Personal Data (KVKK) regarding the collection and storage of user information.

7.2 Ethics and Professional Responsibilities

Throughout the E-Way project, the team recognized and fulfilled several ethical and professional responsibilities:

Data Privacy and Security: In compliance with KVKK, the system minimizes data collection, encrypts sensitive data in transit, and securely hashes user passwords prior to database insertion. Geolocation data is used strictly for routing logic and is never exposed or shared with third parties.

Legal Data Acquisition: To feed the AI predictive models, the system requires continuous status data from the national EPDK API. Following preliminary legal feasibility discussions with our instructor, Mert, it was established that utilizing targeted polling and data collection scripts is legally permissible and ethically sound strictly for this academic, non-commercial research project.

Unbiased AI and Transparency: We ensured the predictive algorithms remain neutral and data-driven, avoiding any algorithmic bias that might unfairly favor specific commercial network operators. To maintain user trust, the UI clearly communicates when station availability is an AI-generated "prediction" versus a real-time status.

7.3 Teamwork Details

7.3.1 Contributing and functioning effectively on the team

Work was distributed based on core competencies. Furkan focused on backend architecture, building the FastAPI RESTful API, setting up JWT authentication, and handling the core routing endpoints. Aziz developed the cross-platform mobile client in Flutter, managing the UI/UX design and interactive map rendering. Utku developed the occupancy prediction logic, training the statistical models and writing the scripts to process historical data. Halis Vefa handled data engineering, designing the MongoDB architecture

and ensuring seamless integration of the automated EPDK data acquisition pipelines.

7.3.2 Helping create a collaborative and inclusive environment

The team maintained a highly collaborative environment. Furkan containerized the backend via Docker, which prevented local dependency conflicts and allowed all members to easily test the server environment. Aziz organized UI/UX feedback sessions to ensure the mobile app reflected everyone's ideas regarding usability. Utku thoroughly documented his AI models' inputs and outputs, making backend integration seamless, while Halis Vefa proactively managed the MongoDB cluster to ensure all members had uninterrupted access to real-time data. We utilized Google Meet for daily communication and GitHub with a strict "Feature Branch" workflow to review each other's code.

7.3.3 Taking lead role and sharing leadership

Leadership was effectively decentralized yet highly collaborative. While each member took ownership of their respective domains, all major technical and architectural decisions were made jointly through continuous consultation. Furkan led backend architecture and API security decisions. Aziz took the lead on the mobile client, guiding decisions on the Flutter framework and state management. Utku assumed the lead role in data science, determining the most effective statistical approaches for predicting EV congestion. Halis Vefa shared leadership by taking charge of data infrastructure and database scalability. Ultimately, no final decision was implemented in isolation; regular team discussions ensured that every architectural choice was cross-examined and perfectly aligned with the system's overall needs.

7.3.4 Meeting objectives

The team successfully met the major milestones outlined in the initial Project Plan. Work Packages 1 through 4 (Analysis, Backend Setup, Mobile Client, and AI Module) were completed successfully, culminating in a fully functional prototype. Work Package 5 (Integration and Testing) was executed rigorously, passing all core functional, performance, and security test cases. Finally, Work Package 6 (Final Deliverables) was met on schedule, delivering a complete Dockerized system, a finalized codebase on GitHub, and a comprehensive final report and presentation for the CS Fair.

7.4 New Knowledge Acquired and Applied

Building the E-Way platform required the team to acquire and apply several new technologies beyond our standard curriculum:

FastAPI & Concurrency: We learned and applied the FastAPI framework, specifically mastering how it dispatches synchronous endpoints to optimized thread pools to prevent blocking the main event loop during heavy geospatial calculations.

Advanced Database Operations: Instead of standard sequential writes, we learned to utilize MongoDB's bulk_write operations with nested $inc operators to efficiently handle high-throughput updates from the EPDK API without locking the database.

Time-Series Analysis with Pandas: While initially considering traditional LSTM neural networks, we acquired new knowledge in statistical time-series aggregation. We successfully applied the pandas library to vectorize EPDK JSON responses and cluster charging sessions temporally (by season, day, and time interval) to build a robust prediction pipeline.

Cross-Platform UI & Map SDKs: We acquired practical experience in Flutter development, specifically applying localized state management (avoiding heavy external libraries) and resolving complex map rendering migrations by migrating from Mapbox to the Google Maps SDK.

8 Conclusion and Future Work

8.1 Conclusion

The rapid adoption of Electric Vehicles (EVs) in Türkiye presents a critical infrastructure challenge: drivers consistently face range anxiety, unpredictable charging station availability, and long waiting times due to network congestion. The current software ecosystem is highly fragmented, forcing users to manually synthesize data from multiple isolated operator applications or rely on static mapping tools that lack EV-specific intelligence.

E-Way successfully addresses these challenges by delivering a unified, intelligent route-planning platform. By aggregating national infrastructure data from the EPDK API and leveraging a highly scalable backend architecture (FastAPI and MongoDB), the system establishes a robust data pipeline. The core innovation of E-Way lies in its Logic Layer, which replaces static navigation with dynamic, AI-driven predictions. By analyzing historical occupancy patterns and forecasting station congestion based on the driver's specific estimated time of arrival (ETA), the system actively routes users away from bottlenecks.

Coupled with a distraction-free, cross-platform mobile client (Flutter) that prioritizes user comfort and vehicle-specific personalization, E-Way transforms the EV driving experience. It provides a reliable, data-driven solution that not only minimizes total travel time but also encourages the broader public adoption of sustainable transportation. Finally, to ensure seamless onboarding for new drivers, the required User Manual is accessible via the official project website.

8.2 Future Work

While the current iteration of the E-Way system successfully meets its primary design goals, the platform lays a strong foundation for several future enhancements:

Crowdsourced Status Updates: Currently, the AI prediction model relies entirely on historical data patterns. To mitigate the risk of prediction inaccuracies and handle sudden anomalies (e.g., broken hardware), a future update will integrate a crowdsourcing feature, allowing users to manually report live station statuses and wait times to continuously refine the AI model's accuracy.

Direct Operator API Integrations: While the EPDK API provides comprehensive national metadata, future work involves establishing direct API partnerships with major network operators (such as ZES and Eşarj). This would allow E-Way to pull highly granular live data, process in-app payments, and enable users to reserve a charging socket directly from the navigation screen.

Real-Time Vehicle Telemetry (OBD-II): The system currently relies on the user to input their starting battery percentage and estimated consumption rates. Future versions of the mobile application could integrate with Bluetooth OBD-II scanners or native vehicle APIs to read the State of Charge (SoC) and battery temperature in real-time, resulting in hyper-accurate range calculations.

Dynamic Pricing Optimization: As energy tariffs fluctuate, future routing algorithms will be expanded to not only calculate the fastest and least busy routes, but also the most economically efficient routes by cross-referencing real-time operator pricing with the vehicle's required charge amount.

9. Glossary

- **AC:** Alternating Current (Charging socket type)

- **AI:** Artificial Intelligence
- **API:** Application Programming Interface
- **CCS:** Combined Charging System (DC fast-charging standard)
- **DB:** Database
- **DC:** Direct Current (Fast charging socket type)
- **EPDK:** Enerji Piyasası Düzenleme Kurumu (Energy Market Regulatory Authority)
- **ETA:** Estimated Time of Arrival
- **EV:** Electric Vehicle
- **GIS:** Geographic Information System
- **GPS:** Global Positioning System
- **JSON:** JavaScript Object Notation
- **JWT:** JSON Web Token
- **KVKK:** Kişisel Verilerin Korunması Kanunu
- **LSTM:** Long Short-Term Memory
- **ML:** Machine Learning
- **POI:** Point of Interest
- **REST:** Representational State Transfer (Software architectural style for APIs)
- **SDK:** Software Development Kit
- **SoC:** State of Charge (Battery level)
- **UI:** User Interface
- **UML:** Unified Modeling Language
- **VM:** Virtual Machine

10. References

[1] Enerji Piyasası Düzenleme Kurumu (EPDK), "Şarj Hizmeti Yönetmeliği," Resmi Gazete, Sayı: 31797, Apr. 2022.

[2] Tiangolo, "FastAPI Framework," [Online]. Available: https://fastapi.tiangolo.com/

[3] IEEE Computer Society, "IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998, Oct. 1998.

[4] Object Management Group (OMG), "OMG Unified Modeling Language (OMG UML)," Version 2.5.1, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1

[5] T.C. Resmi Gazete, "Kişisel Verilerin Korunması Kanunu (Kanun No. 6698)," Apr. 2016.

[6] Google Developers, "Flutter Documentation - Build apps for any screen," [Online]. Available: https://flutter.dev/docs

[7] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle routing problem with time windows and recharging stations," Transportation Science, vol. 48, no. 4, pp. 500–520, 2014.