Bilkent University

Department of Computer Engineering

CS 492 - Senior Design Project

SürDur

Spring 2025

Final Report

T2419

Bora Haliloğlu - 22101852

Burak Oruk - 22102443

Emir Tuğlu - 22003165

Mustafa Gökalp Gökdoğan - 22102936

Tevfik Emre Sungur - 22102377

Table of Contents

1. Introduction

SürDur is a mobile application designed to enhance the road trip experience by offering personalized stopovers along the route. After selecting a destination, users can explore a variety of recommended points of interest (POIs) drawn from various travel databases and blogs, providing a customized and enriching journey. The app also integrates a social media feature that allows users to vote on routes, follow other travelers, and share experiences. This community-driven aspect fosters engagement and facilitates personalized recommendations based on user interactions. SürDur has been developed as a cross-platform solution, supporting both Android and iOS devices to ensure accessibility for a wide range of users.

In the remainder of this report, we begin by introducing the SürDur application, outlining its main features and the core functionality it provides. We then discuss the design goals that guided the development, as well as the system's architecture, providing insights into the layered structure and components. Following this, we present the implementation details, including backend and frontend development, and how the application was built to meet both functional and non-functional requirements. Additionally, we cover the test cases we designed to validate the application's features and performance. We also examine the key constraints and considerations, such as technical, economic, and social factors, that shaped our decisions throughout the project. The report concludes with a detailed look at teamwork, highlighting the contributions of each team member, and offering reflections on the project's success and potential future improvements.

2. Requirements Details

2.1. Functional Requirements

This section describes the functional requirements that SürDur application follows.

2.1.1. Sign Up & Login

Application does:

- Allow users to log in to the application using their account information.
- Allow users to register manually by providing account details.
- Allow users to add their personal information, such as name, age, and preferences.
- Enable users to add place category preferences for road trips.

2.1.2. Destination Selection

Application does:

- Display the user's current location on an interactive map.
- Allow users to move around the interactive map.
- Provide a feature to return to the user's current position on the map.
- Allow users to select a destination location for route planning.

2.1.3. Point of Interest Suggestion

Application does:

- Display suggested points of interest (POIs) close to the given route.
- Allow users to zoom in or out to a particular region to get an adequate number of suggestions in that area.
- Allow users to filter POIs by category to display only the POIs with the selected categories.
- Display the preview information about the suggested places.
- Display detailed information about the selected place among the list of suggested places.

- Allow users to add places to the route from the given suggestions.
- Allow users to remove places from the route among all of the previously selected places.
- Reconstruct the route dynamically after each place addition and removal.
- Display the estimated time to reach the destination and the percentage of how off the new route is from the original route.
- Allow users to finalize the route with selected places and save it on the ‘Planned’ routes of the user.

2.1.4. Social Media

Application does:

- Display the shared route posts from other users on the application, with the components:
    - Starting and Destination Location
    - Interactive Map Overview of The Route
    - Title
    - Description
    - Author Username and Profile Picture
    - Upvote and downvote counts
- Allow users to display detailed description of the selected route among shared routes.
- Open the selected shared route by the user on the interactive map.
- Allow users to upvote or downvote the route post.
- Allow users to follow and unfollow other users
- Allow users to save the selected shared routes on the ‘Saved Routes’ folder of the user.
- Send users notifications when there are upvotes or downvotes on their posts.
- Send users notifications when the users that are followed share their routes.

2.1.5. Archives

Application does:

- Display all the personal routes of the user that are divided into four route categories:

    - In Draft: Routes that have not been totally completed on planning by the user and require further route planning completion from the user.
    - Planned: Routes that have been assigned as ‘completed planning’ by the user and are ready to be traveled afterward.
    - Completed: Routes that have been traveled and finished by the user in real life.
- Allow users to filter personal routes based on their category, destination location, and start location.
- Open the selected route from the archive on the interactive map for further route editing or to start the road trip.
- Allow users to delete present routes from the user archive.
- Allow users to share their routes on the social media part of the application.
- Allow users to flag certain routes for further distinction from other routes.

2.1.6. Live Route Navigation

Application does:

- Allow users to start live navigation throughout the selected route.
- Perform real-time navigation throughout the road trip.
- Display real-time road directions (next maneuver’s turn and distance etc.) on the screen.
- Display the total expected time remaining.
- Allow users to select emergency 'gas station' and 'electric vehicle charging station' stops.
- Add the nearest and most convenient gas station or electric vehicle charging station dynamically into the current route if chosen.
- Adjust live directions dynamically when the driver gets off the planned route.
- Allow users to exit the live navigation of the current route.

2.1.7. Profile Editing & Setting Adjustment

Application does:

- Allow users to edit their personal account information, such as username, password, etc.
- Allow users to adjust application experience settings, such as theme, distance and speed units, notification enablement, etc.

2.2. Nonfunctional Requirements

2.2.1. Usability

The usability of SürDur is evaluated based on both the duration of app usage and the satisfaction level of users with the personalized place suggestions. Therefore, SürDur provides a user-friendly interface that includes simple, easy-to-use, yet comprehensive components, which allow users to smoothly navigate from the main page to the successful completion of a planned route. This design aims to reduce confusion during usage and increase overall user engagement. Additionally, the application ensures that the number of place suggestions is balanced—neither overwhelming the user with too many options nor offering too few, which could result in a lack of interest. The current version of SürDur has been implemented to work efficiently across different platforms, including Android and iOS, ensuring accessibility for a broad user base.

2.2.2. Reliability

SürDur is designed to provide uninterrupted and stable navigation throughout the trip. In case of network issues, the application caches map data locally on the user's device and continues to operate without relying on a live internet connection. It is also equipped to handle connection failures by informing the driver clearly and safely, without causing distraction. These features ensure that navigation continues smoothly even in areas with limited connectivity. The application is capable of maintaining its functionality during planned maintenance or unexpected server downtimes. In addition to these features, the system regularly creates backups of the

database to prevent potential data loss due to crashes or critical failures, which increases the overall robustness of the system

2.2.3. Performance

SürDur includes a data retrieval system that is optimized to reduce latency and improve the responsiveness of route planning and recommendation features. The application makes use of caching strategies to store and quickly access frequently needed place information, which is selected based on user location, preferences, and relevance. These optimizations help ensure that users receive fast and relevant suggestions without noticeable delay. Moreover, live navigation is designed to run with minimal lag, providing users with a smooth and real-time experience during their trips. This is particularly important for safety, as even small delays in navigation feedback can negatively impact the driving experience.

2.2.4. Supportability

SürDur is implemented with a flexible and scalable system architecture that supports global compatibility and long-term maintenance. The development environment allows the backend structure to be updated through the addition or removal of microservices without affecting the core functionality of the application. To support smooth deployments and consistent environments, virtualization tools are used across development and production setups. This approach simplifies library updates and configuration management. Additionally, the system is equipped with detailed logging services that record system events, errors, and warnings. These logs are valuable for identifying and resolving issues quickly, improving system reliability and maintainability over time.

2.2.5. Scalability

SürDur is capable of handling large-scale usage scenarios that involve high volumes of user requests and vast amounts of place data. The backend is built with fast and asynchronous service logic to

support many users at the same time without major slowdowns. The application uses data storage and caching strategies to optimize performance under increasing load. Efficient use of database queries and memory also helps ensure that large amounts of place data can be managed and retrieved without causing delays or overloading the system. As the number of users and data entries grows, SürDur can adapt to demand by scaling its services without sacrificing performance or stability.

3. Final Architecture and Design Details

3.1. Overview

The following component diagram shows the architecture in terms of separate, self-contained components. When joined in a certain manner, they build the whole software, and this diagram includes those connections and components. It represents SürDur's layered architecture in a slightly more detailed way. SürDur’s layered architecture is designed to promote modularity and maintainability of the project. This ensures modern architectural principles like modularity and separation of concerns. It contains four main layers: Presentation Layer, Business Logic Layer, Data Access Layer, and Database Layer.

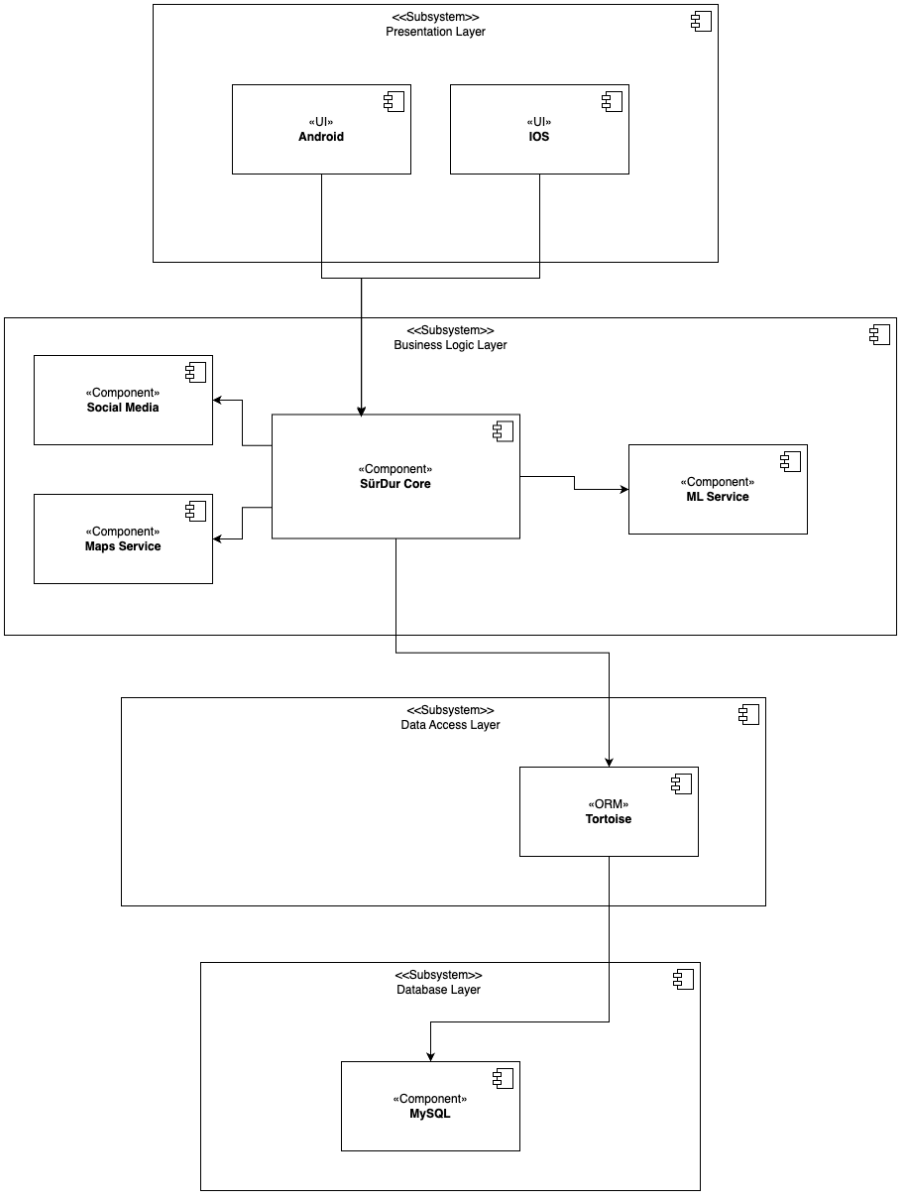

Figure 1: Component Diagram of SürDur Link SürDur Component Diagram.drawio.png

3.2. Subsystem Decomposition

The High Level System Architecture consists of several key subsystems, each of which plays a specific role within the overall architecture. By maintaining a layered structure, the system ensures modularity, maintainability, and separation of concerns, ultimately making the architecture more scalable and efficient. The subsystems are divided into four main layers:

3.2.1. Presentation Layer

Presentation Layer includes two different mobile UIs (Android and iOS). This layer essentially enables user interactions.

3.2.2. Business Logic Layer

The Business Logic Layer includes the core and supporting components like ML Service and Maps Service. Each supporting component is directly connected to SürDur's Core since the core acts as coordinator of other components. ML Service returns POI recommendations. Maps Service is responsible for navigation and pathfinding. Social Media is responsible for post and user interaction management. Therefore, this layer's main function is supporting the application's core business functions. It also acts as an intermediary between the presentation and data access layers.

3.2.3. Data Access Layer

The Data Access Layer manages data persistence between the Business Layer and the Database Layer with Tortoise ORM Library. It simplifies database operations.

3.2.4. Database Layer

The Database Layer uses a MySQL database which includes all the data about the users, places, posts, and route logs. It manages the data storage and retrieval. It provides this information through Tortoise ORM in the Data Access Layer.

3.3. Services

In the backend architecture of SürDur, the Business Logic Layer was designed as a modular, component-based structure that effectively decoupled the major functional domains of the system. This organization enhanced maintainability, scalability, and fault isolation throughout development and deployment. The core components of the Business Logic Layer are the SürDur Core, the ML Service, the Maps Service, and the Social Media Service, each of which was developed as an independent and cohesive module.

3.3.1. SürDur Core

The SürDur Core component served as the central orchestrator within the Business Logic Layer. It operated as the primary interface between the client-side Presentation Layer and the specialized backend services. SürDur Core managed authentication, input validation, routing, and response composition, ensuring that every client request was processed securely, efficiently, and consistently. It directed route planning requests to the Maps Service, personalized recommendation queries to the ML Service, and social interaction activities to the Social Media Service. The Core service also handled system-level responsibilities such as centralized error management, structured event logging for monitoring, and the enforcement of security and rate-limiting policies. By consolidating these critical control operations, SürDur Core ensured smooth cross-service workflows and upheld the system's high standards for reliability, responsiveness, and user experience.

3.3.2. ML / Recommendation Service

The ML Service was responsible for delivering SürDur's personalized point-of-interest (POI) recommendation functionality. It processed user-specific preference vectors that are derived from past interactions and choices and compared them against the embeddings of available POIs using a cosine similarity-based ranking algorithm. The service generated a ranked list of stopovers tailored to the user's preferences, route geometry, and context. To enhance recommendation diversity and avoid overly narrow suggestions, the ML Service applied controlled exploration strategies such as epsilon-greedy selection. The ML Service was a key enabler of SürDur's differentiation from traditional navigation platforms, providing intelligent, user-adaptive suggestions that transformed the travel experience into a more personal and engaging journey.

3.3.3. Maps Service

The Maps Service managed all geospatial functionalities within SürDur, including initial route generation, geocoding, and reverse geocoding. It integrated external providers such as OpenRouteService to calculate optimal routes. The service supported dynamic route adjustments by recalculating optimal paths whenever users added or removed stopovers during their trip. It also provided enriched navigation details, such as maneuver-by-maneuver instructions, estimated

times of arrival, and alternative route suggestions. By combining external data sources with local resilience mechanisms, the Maps Service significantly enhanced the reliability, flexibility, and user satisfaction of the SürDur application, ensuring consistent navigation quality across varying conditions and operational scenarios.

3.3.4. Social Media

The Social Media Service enabled all social interaction features within SürDur. It allowed users to create and publish route posts, upvote or downvote shared routes, follow or unfollow other travelers, and save interesting routes to personal collections. The service was designed to maintain strict separation between private and public user data, ensuring data integrity and privacy compliance. It utilized event sourcing to track all social interactions in an auditable and consistent manner, while its read models, optimized for performance, enabled rapid retrieval of feeds and posts. The Social Media Service successfully promoted community engagement, enriching the travel experience by fostering interaction, discovery, and knowledge-sharing among SürDur users.

3.4. Hardware/Software Mapping

SürDur does not require any additional software mapping other than the device’s GPS functionality and internet connectivity. The application is designed to operate without excessive computational requirements on personal devices. Users' personal devices are primarily used to store locally cached route information and user-preferred points of interest (POIs), ensuring that storage requirements remain minimal.

SürDur is developed for both Android and iOS platforms. Any mobile device that meets the requirements of an active GPS module and internet connection is suitable for running the application.

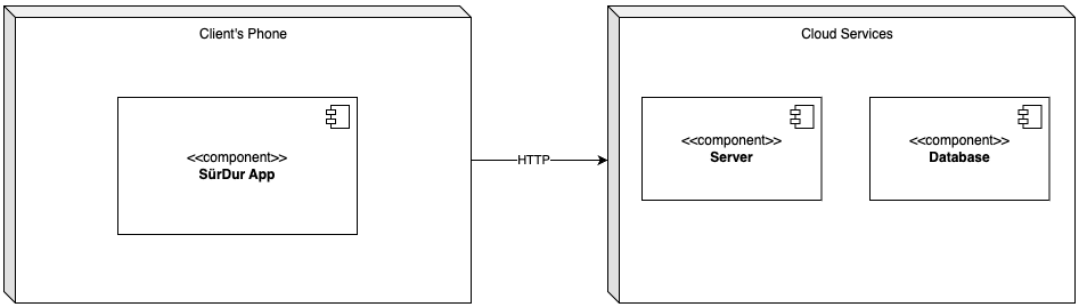

Figure 2: Hardware/Software Mapping

SürDur is developed using FastAPI for the backend. The frontend is built with React Native to ensure cross-platform compatibility. Google Maps and Apple Maps are used for map rendering and OpenRouteService API is used for route creation, while AWS servers host the backend and database.

The system architecture, the client's device (smartphone) runs the SürDur application and communicates with the server. Upon receiving a request from the client, the server retrieves relevant data, including personalized POI recommendations, route data, and user preferences. The server processes this data and returns a response to the presentation layer on the client device.

3.5. Persistent Data Management

Our project leverages AWS RDS with MySQL to store and manage data, including over 70,000 places sourced from the FourSquare API. Given the scale and diversity of this data, we implemented a strategy to optimize category management for more efficient querying and analysis. Initially, the raw place data included a vast number of categories, some of which were not directly related to our project's purposes.

To address this, we applied the k-means clustering algorithm to group similar categories, reducing redundancy and organizing the data into more coherent clusters. After this automated clustering step, we manually refined the results to correct vague or ambiguous classifications, ensuring a higher level of precision. This hybrid approach where combining machine learning with human judgment, helped us build a more reliable and manageable category structure, enhancing the overall accuracy and usability of the data for our application.

3.6. Access Control and Security

SürDur implements access control and security mechanisms to ensure the safety and usability of user data. The application uses FastAPI's integration with the OAuth2 authorization protocol to apply access control. Whenever a user logs in, a unique and distinctive access token is generated and sent to the user. This token is stored in the users' devices and appended to each subsequent request coming from that user. This way, the server can identify from which user a request originates from. Knowing the owner of a request, the business logic implemented in the server side determines whether the requested activity can be performed by the request owner. In this way, users can only access data and functionality they are entitled to.

SürDur also has security mechanisms that ensure security of the user data. User passwords are encrypted using the bcrypt algorithm before being stored in the database. This prevents exposure of sensitive password data even if a data breach occurs. The application also utilizes the SQLAlchemy ORM library to query the database. This ensures that user inputs are sanitized before being passed into an SQL query, hence preventing SQL injection attacks. Moreover, in the client devices, authentication tokens are stored in secure storage rather than local storage to prevent XSS attacks.

SürDur also implements additional measures to protect the system against brute force attacks and API abuse. Using rate limiting in the server to restricting the number of requests a user can perform within a time interval. This way, SürDur prevents attacks like user enumeration, credential guessing, and API abuse. Moreover, the system has logging mechanisms to detect and respond to any kind of security threat.

4. Development/Implementation Details

The Subsystem Services section details the functional components within each subsystem of the SürDur application. These services define the core operations that enable various features of the system, ensuring seamless interaction between users and the application. The system is structured into three main layers: Presentation Layer, Server Layer and Data Access Layer. Each layer is responsible for specific tasks.

4.1. Presentation Layer Services

This layer contains services related to the user interface and interactions. The presentation layer of the SürDur mobile application is developed using React Native, that supports both iOS and Android environments. For UI rendering and smooth interaction with maps, the app integrates with built-in map display services of Apple and Android platforms configured in the Expo framework and utilizes OpenRouteService API for real-time route creation. Expo facilitates easier deployment and testing during development.

- **Interactive Map Display Service:** Handles rendering, animations, and displaying dynamic and interactive map elements for the mobile application (iOS, Android).
- **Route Generation Service**: Manages all user interactions, such as selecting a destination, filtering POIs, and adding places to routes.
- **Live Navigation Display Service**: Presents real-time route navigation and step-by-step directions, using OpenRouteService and handles offline routing by caching the route and directions.
- **Social Media Display Service**: Shows user-generated posts, upvotes, downvotes, posted images in the social media section of the app.
- **Profile & Settings Service**: Allows users to edit their profiles and adjust settings.

4.2. Server Layer Services

This layer contains services responsible for processing and logic execution. These services interact heavily with the main database, requiring the Data Access Layer for most operations. The server layer of the SürDur application is developed using Python and the FastAPI framework, taking advantage of its asynchronous capabilities to ensure scalable and efficient request handling. For data validation and serialization, the Pydantic library is used to define Data Transfer Objects (DTOs) and enforce strict schema constraints, ensuring that the data exchanged between the client and server remains consistent and type-safe. The backend services are deployed on AWS infrastructure—specifically, EC2 service for hosting the FastAPI server, RDS for managing the MySQL database, and S3 for storing media assets such as profile images. These services are integrated within the server layer to

ensure reliable performance, high availability, and seamless data flow across the application. There is a core decomposition of the server layer services below.

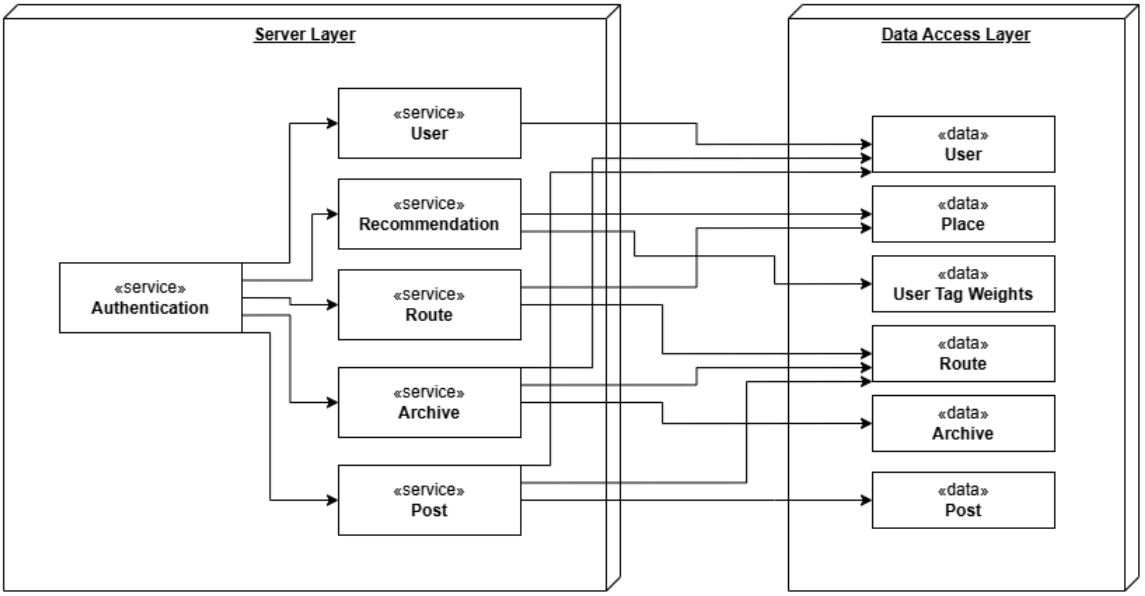

Figure 3: Decomposition of Server Layer Services

- **User Authentication Service**: Manages user login, registration, and session handling. This service has to be configured before other services to execute, meaning all the other server services are dependent on authentication service. This service interacts with the User Data Access layer to handle safe and robust database logic functions.

- **AI-Based Recommendation Service**: Interacts with our custom database that holds both the user preferences and all the place information across the Türkiye. Analyzes user preferences and suggests personalized point of interests (POI) along the route using destination information and word embedding-based cosine similarity approach between place tags that has been gathered from the largest place databases (e.g. FourSquare) and user tag weights. User preferences are updated after the route completion.

- **User Service:** Handles post-authentication user-related use-cases, such as profile editing, personal information retrieval. This service interacts with the User Data Access layer to handle safe and robust database logic functions.

- **Route Service**: Handles route-related use cases other than route recommendation; such as adding and removing places inside a completed route, deleting and editing route information. This service interacts with the Route Data Access layer to handle safe and robust database logic functions.

- **Post Service**: Handles social media related user interactions, including posting, upvoting, following users, and sharing routes. The posts are retrieved with batches to secure robustness and performance. This service interacts with the Post Data Access layer to handle safe and robust database logic functions.

- **Archive Service**: Manages personal route and post archive for each individual user, allowing personal archive organization, sharing completed routes as posts, and handling saved routes from other user's posts.

4.3. Data Access Layer Services

This layer ensures that data is retrieved and stored efficiently between the Business Logic Layer and the Database. The underlying database system is MySQL, a widely-used, reliable relational database that stores core data such as user profiles, route archives and point-of-interest (POI) metadata.

- **ORM Data Retrieval Service**: Uses SQLAlchemy ORM to fetch and update database entries efficiently.
- **User Data Management Service**: Fetches and updates user information, such as travel history (archives), personal preferences, and saved routes.
- **POI Data Fetching Service**: Retrieves place details, categories, and popularity rankings from the database layer filled with data from external sources (e.g. FourSquare).

4.4. Data Preprocessing

The data needed for SürDur to work with mostly consists of geolocations of the places (i.e. restaurants, historical sites). However, for the recommendation system to work, categories of those places should be present as well. Therefore, category metadata was requested alongside with fundamental details of places while fetching data from external sources (e.g. Foursquare). The data across Turkey

having 581 distinct categories, the question of “can this many categories be personalized accurately and precisely” has arised.

As a means of reducing the redundancy in the data, k means clustering was performed on the category data. The resulting clusters (not all of them, this will be explained later) were mapped as one category (i.e. “Adana Kebab” and “Shish Kebab” can both be treated as “Turkish Kebab”) since they are closely related in their meanings. Even though some precision within the data is lost, the reduction in number of categories helps the recommendation system to not overfit the data and give more relevant suggestions.

While performing the k means clustering, in order to reduce ineffective clusters, frequent phrase words such as “shop”, “store”, “place” were removed before clustering. The effect of removing such stop words can be seen in the following example:

- Baby Store
- Gun Store
- Cheese Store

In this set of examples, keeping the word “store” adds a bias to the word they are paired with for a phrase embedding is calculated as the average of separate words of it, resulting in phrases treated as close during k means clustering even if they are only related in being a store and should not be grouped together. When the word “store” is removed for calculation, Gun Store is grouped with Gun Range as it should happen. After contextual stop words were removed, k means clustering was performed with various k values. Then the clustering that gave the most relevant groupings was chosen with human judgement.

In order to reduce the negative effects of the clustering approach, some clusters were manually disbanded, or divided into smaller clusters to regain some of the lost precision. As an example, as the result of the machine clustering all instances of Turkish food is clustered under a group. However, for an app that has Türkiye as its target market, treating all Turkish food as one would not yield a fruitful personalization since it will not have the ability to differentiate any of the wildly popular dishes. At this stage, two things should be noted: One is that the app is produced for a Turkish audience, hence it is logical to tailor the data which app will operate on according to the audience. Second is that this process of arranging

clusters manually was conducted by human judgement and is prone to error. To reduce the human bias as much as possible, after the task was done by the responsible body, there was a review session held with all the members of the group in order to discuss and double-check the decisions that were made.

After clustering is done, in order to add a layered structure to the data, and provide a medium for the recommendation system to do exploration (details explained in section 4.5), a second round of clustering was operated upon the clustered data. Later the clusters arisen from the second clustering were grouped together meaningfully to construct a third layer of categories. Figure 4 represents the layers of categories to provide a more clearer understanding. It should be noted that Layer 0 is discarded in the process of writing the fetched data to the projects database, and that the app only operates on Layer 1 and above.

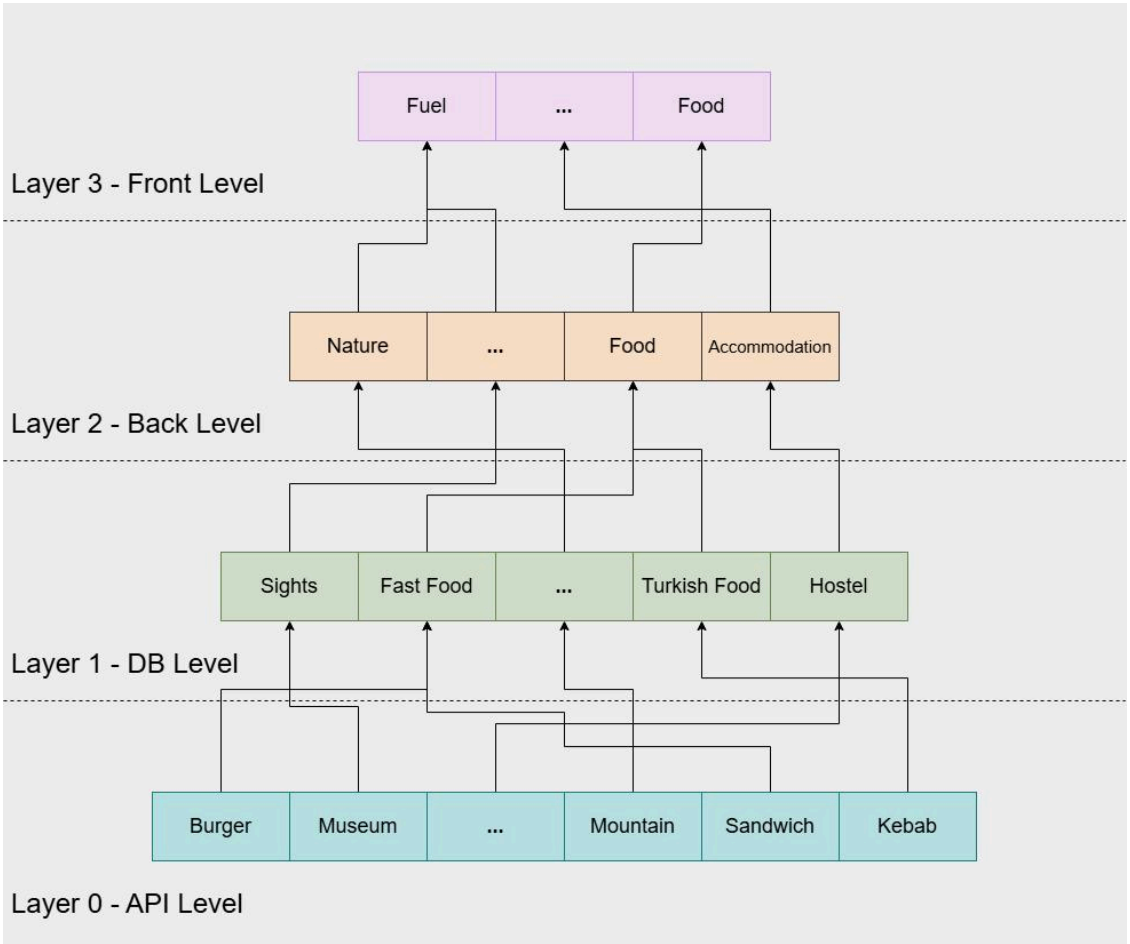

Figure 4: Representation of Categories' Layered Structure

4.5. Recommendation Mechanism

1. Configuration & JSON Logging

Each user's level-1 weights and complete embedding are written as JSON lines by a dedicated logger, which flushes instantly. Every run logs both raw and "delta" records, calculates L_2 distances and cosine deltas against the most recent values, and removes older entries to retain only the most recent.

2. Category Data & Hierarchy

With pre-calculated high-dimensional embeddings, top-level categories reside in a straightforward text list. A three-tiered hierarchy is defined by a JSON map (e.g. Ramen → Japanese Cuisine → Food).

3. Initial User Profile Creation

Sent for embedding, declared preferences (e.g., “Food”:3) are converted into a weighted sentence and stored. Similarities to the user's embedding are scaled by defined strengths, aggregated, normalized (max = 1.0), and persisted. Each category begins with a default weight.

4. Embedding & Similarity Utilities

A lazy initialization of an embedding client is made. With each request, a saved embedding is retrieved (or a random fallback is used), dimensions are checked, and similarity is calculated by a cosine-similarity helper.

5. Recommendation Pipeline

It loads the embedding client and category data at startup, retrieves the user's embedding and weights for each request, logs any modifications, and then ranks the top K categories based on embedding similarity.

- Ranks top K categories by embedding similarity.
- Propagates 60% of each level-1 weight to level 2 and 30% to level 3.
- Adds ε-greedy exploration from level 2/3.
- Computes the route’s length and selects 10–25 slots based on distance tiers.
- Filters places via a lat/lon bounding box, indexes them with a k–d tree, ensures each category appears once, then fills remaining slots by rating-and-detour buckets (≈70% high, 30% medium, rest low).
- Returns a list of place records (ID, name, category, coords, rating, image).

6. Continuous Profile Updates

When users click or select, the corresponding embeddings (weighted modestly) are added to their stored embedding and level-1 weights are reinforced and saved.

7. Core Algorithms & Design Principles

Uses cosine similarity for weighting and ranking; ε-greedy for exploration; hierarchical propagation to generalize interests; Haversine + k–d tree for efficient spatial filtering; rating-based diversification; and a modular pipeline for scalability.

5. Test Cases and Results

5.1. Test Cases for Functional Requirements

Test ID	1.1	Category	System Test	Severity	Major
Objective	Verify that an embedding vector is correctly generated for each category using OpenAI's text-embedding-ada-002 model.				
Steps	1. Check if each category is generated successfully. 2. Check whether the semantically similar categories also have close vector embeddings.				
Expected	The vector correctly symbolizes each category, even if the category contains Turkish words.				
Date-Result	24.04.2025 - Pass				

Test ID	1.2	Category	Security	Severity	Major

Objective	Verify that the Recommendation Engine is not prone to attacks such as DDoS
Steps	1. Check if the server is usable under a heavy load of requests. 2. Check if one user is able to damage the system by sending multiple queries.
Expected	The server will recognize that the receiving queries are overwhelming the server and initiate a control mechanism.
Date-Result	N/A

Test ID	1.3	Category	System Test	Severity	Critical
Objective	When a user selects multiple categories, verify that their profile vector is correctly computed using the chosen aggregation method				
Steps	1. Check if the backend can handle multiple categories in the recommendation system. 2. Check if the UX is smooth when multiple categories are selected.				

Expected	The recommendation profile should be updated with respect to all the selected categories.
Date-Result	24.04.2025 - Pass

Test ID	1.4	Category	Functional Test	Severity	Critical
Objective	Given the same set of selected categories, ensure that the generated profile vector is always the same (deterministic behavior)				
Steps	1. Check if the system has deterministic behavior when categories are selected for recommendation. 2. Check if the profile vector has the same value for the same categories.				
Expected	The two profiles that were saved after the calculation should be the same. In other words, there should be a deterministic behaviour.				
Date-Result	24.04.2025 - Pass				

Test ID	1.5	Category	System Test	Severity	Major
Objective	When exploration is enabled, ensure that the recommendations include some diverse or less similar places, not just the top matches.				
Steps	1. Check if when exploration is enabled in the recommendation system the exploration results include diversity.				
Expected	The results should include diverse recommendations.				
Date-Result	24.04.2025 - Pass				

Test ID	1.6	Category	Integration Test	Severity	Major
Objective	Tests whether data is correctly passed between microservices and stored in the database.				
Steps	1. Check whether the data is lost between microservices. 2. Check whether the data is stored without any loss in the database. 3. Check if the microservices correctly modify the data 4. Check if the modified data is sent without any loss over the network.				
Expected	There should not be any loss at any point in the transactions. The modifications done to the data should be as expected.				

Date-Result	24.04.2025 - Pass

Test ID	1.7	Category	System Test	Severity	Major
Objective	Verifies that a user receives a confirmation email after successful registration.				
Steps	1. Check if the user recieves a mail after successful registration. 2. Check if the mail received has the correct content and is sent to the correct recipient.				
Expected	The mail should be sent to the correct recipient and the content should be as expected meaning that there should not be any distortion in the content.				
Date-Result	24.04.2025 - Fail				

Test ID	1.8	Category	Security Test	Severity	Major
Objective	Verify that users can successfully register with valid credentials.				
Steps	1. Register with valid credentials and check if it succeeds. 2. Try registering with missing or incorrect fields and verify error messages. 3. Attempt to register with an existing username or email. 4. Test with special characters and non-ASCII inputs to check system stability.				

Expected	The system should allow valid registrations while preventing invalid inputs and duplicate accounts. It should handle all inputs without crashing and enforce security measures effectively.
Date-Result	24.04.2025 - Pass

Test ID	1.9	Category	Functionality Test	Severity	Minor
Objective	Verify that users can filter POIs based on categories.				
Steps	1. Navigate to the main screen for route recommendation. 2. Select a destination and view the suggested POIs. 3. Apply a filter by selecting a specific category (e.g., restaurants, gas stations). 4. Verify that only POIs from the selected category are displayed. 5. Remove or change the filter and ensure that results update accordingly.				
Expected	The system should correctly display POIs based on the selected category, updating the suggestions dynamically. Unfiltered results should restore when no category is selected.				
Date-Result	29.04.2025 - Pass				

Test ID	1.10	Category	Functionality Test	Severity	Major
Objective	Verify that POI details (ratings, descriptions) are correctly displayed on the recommendation stage.				

Steps	1. Navigate to the main screen for route recommendation. 2. Select a destination and view the suggested POIs. 3. Click on a recommended POI to open its details. 4. Check if the POI details such as name, description, rating, and category are displayed correctly. 5. Compare displayed details with the expected values from the database or API.
Expected	The system should accurately display the correct POI name, description, ratings, and other relevant details. Any missing or incorrect information should not be shown.
Date-Result	26.04.2025 - Pass

Test ID	1.11	Category	Functionality Test	Severity	Major
Objective	Verify that POI suggestions are displayed on the map correctly based on the destination route.				
Steps	1. Select a starting point and a destination on the map. 2. Allow the system to generate a route between the two points. 3. Check if suggested POIs appear along the route. 4. Click on a suggested POI and verify its location matches the expected coordinates. 5. Compare displayed POIs with actual relevant locations from the database or API.				
Expected	POI suggestions should be accurately placed along the selected route on the map. Locations should correspond to real-world data, and misplaced or missing POIs should not occur.				
Date-Result	26.04.2025 - Pass				

Test ID	1.12	Category	Functionality Test	Severity	Moderate
Objective	Verify that users can create posts with their previously saved routes.				
Steps	1. Navigate to the ‘Archive’ section. 2. Select a previously saved route from the personal archive. 3. Click on the "Create Post" button. 4. Enter a title, description, and optionally add images. 5. Publish the post and verify that it appears in the social page.				
Expected	The system should allow users to create posts using their saved routes, and the post should be visible in the social feed with the correct details.				
Date-Result	26.04.2025 - Pass				

Test ID	1.13	Category	Functionality Test	Severity	Minor
Objective	Verify that users can follow/unfollow other users.				
Steps	1. Navigate to the social page to display posts. 2. Click on a user's profile to open their details. 3. Click the "Follow" button and verify that the status changes to "Following." 4. Refresh the page and ensure the following status persists. 5. Click "Unfollow" and confirm that the status updates correctly.				
Expected	Users should be able to follow and unfollow others seamlessly, with the				

	status updating correctly and persisting after refresh.
Date-Result	26.04.2025 - Pass

Test ID	1.14	Category	Functionality Test	Severity	Moderate
Objective	Verify that users can upvote/downvote and retract their votes on posts correctly.				
Steps	1. Navigate to the social page and find a post. 2. Click the "Upvote" button and verify that the vote count increases. 3. Click the "Downvote" button and check if the vote count adjusts accordingly. 4. Click the same vote button again to retract the choice and ensure the count updates properly. 5. Refresh the page and confirm that the vote status remains consistent.				
Expected	Users should be able to upvote, downvote, and retract their choices without errors. The vote count should update correctly and persist after a refresh.				
Date-Result	26.04.2025 - Pass				

Test ID	1.15	Category	System Test	Severity	Critical
Objective	Users would select any place from the search engine.				

Steps	1. Enter search. 2. Search for any place and select it as the destination.
Expected	A route should be created with the correct locations and should perform routing operations.
Date-Result	26.04.2025 - Pass

Test ID	1.16	Category	System Test	Severity	Critical
Objective	When selecting a destination from microservices, the unknown place should be available within the application regarding the operations related to that route.				
Steps	1. Enter search. 2. Select detailed search. 3. Search and select a destination that is not in the SürDur DB.				
Expected	Routing operations should continue as usual.				
Date-Result	27.04.2025 - Pass				

Test ID	1.17	Category	Functionality Test	Severity	Major
Objective	When a place not from the SürDur database is selected, it should be held effectively in the backend. When all the references to the temporary				

	destination are deleted, the object should be garbage collected.
Steps	1. Search and select a destination that is not in the SürDur DB. 2. Save the route. 3. Verify the destination is stored in the backend. 4. Delete all references to the temporary destination (e.g. the saved route).
Expected	The temporary place is no longer stored in the backend.
Date-Result	27.04.2025 - Pass

Test ID	1.18	Category	Unit Test	Severity	Minor
Objective	When the recommendation engine selects places from the database, if a place belonging to several classes is selected, it should not be served several times but once.				
Steps	1. Enter search. 2. Select destination. 3. Wait until recommendations are provided.				
Expected	Recommendations are given as unique subsets of places.				
Date-Result	27.04.2025 - Pass				

Test ID	1.19	Category	Unit Test	Severity	Minor

Objective	Fetched places should be within a constant radius of the road.
Steps	1. Enter search. 2. Select destination. 3. Select radius *r* for the recommendations to be searched within. 4. Wait until recommendations are provided.
Expected	Recommendations are within the *r* kilometers radius from the route.
Date-Result	27.04.2025 - Pass

Test ID	1.20	Category	Regression Test	Severity	Minor
Objective	Ensures that users can upload a profile picture, and existing functionality remains unaffected.				

Steps	1. Log in to the application. 2. Navigate to the profile settings page. 3. Click on the "Upload Profile Picture" button. 4. Select an image from the device. 5. Confirm and save changes.
Expected	Profile picture is uploaded and displayed correctly without affecting other functionalities.
Date-Result	27.04.2025 - Pass

Test ID	1.21	Category	Integration Test	Severity	Major
Objective	Ensures that the recommendation engine fetches relevant POIs based on user preferences.				
Steps	1. The user logs in and selects a destination. 2. The system retrieves POIs from various sources. 3. Apply user-specific filtering based on preferences. 4. Display recommended POIs along the route.				
Expected	The user sees relevant POIs based on previous selections and interactions.				
Date-Result	28.04.2025 - Pass				

Test ID	1.22	Category	Acceptance Test	Severity	Major
Objective	Confirms that users can create, edit, and share posts successfully.				
Steps	1. Login and navigate to the social feed. 2. Click on "Create Post" and add route details. 3. Attach images and a description. 4. Click "Share" and verify the post appears in the feed. 5. Edit the post and save changes.				
Expected	Users can create, modify, and share posts without errors.				
Date-Result	28.04.2025 - Pass.				

Test ID	1.23	Category	Functional Test	Severity	Moderate
Objective	Ensures that search queries return accurate and relevant results without knowing the internal implementation.				
Steps	1. Open the app and navigate to the search bar. 2. Type a location or POI category. 3. Press enter and observe results. 4. Verify that search results are relevant to the query.				
Expected	The system displays correct results matching the entered keywords.				

Date-Result	28.04.2025 - Pass

Test ID	1.24	Category	Security Test	Severity	Major
Objective	Ensures that users cannot access or modify data they are not authorized.				
Steps	1. Attempt to access another user's profile settings via URL manipulation. 2. Try sending a request to modify someone else's data. 3. Analyze server response codes. 4. Ensure database restrictions prevent unauthorized access.				
Expected	Unauthorized attempts are blocked, and security logs record suspicious activities.				
Date-Result	28.04.2025 - Pass				

Test ID	1.25	Category	Functionality Test	Severity	Moderate
Objective	Verify that users can add/remove POIs to the route in between the starting and destination locations.				
Steps	1. Select a destination by clicking a location on the map. 2. Click on the “Create Route” button. 3. Add two recommended POIs to the route by clicking the “+” icon next to them..				

	4. Remove a POI from the route by clicking the “-” icon next to it..
Expected	One POI remains in the route.
Date-Result	28.04.2025 - Pass

Test ID	1.26	Category	Functionality Test	Severity	Moderate
Objective	Verify that users can change the order of POIs in their routes.				
Steps	1. Select a destination by clicking a location on the map. 2. Click on the “Create Route” button. 3. Add POI A to the route by clicking the “+” icon next to it. 4. Add POI B to the route by clicking the “+” icon next to it. 5. Drag POI B in front of POI A in the route overview bar.				
Expected	POI B precedes POI A in the route.				
Date-Result	28.04.2025 - Pass				

Test ID	1.27	Category	Security Test	Severity	Major
Objective	Verify that users cannot inject and execute malicious code as a text input while creating posts.				

Steps	1. Navigate to the “Archive” page. 2. Select a previously saved route from the archive. 3. Click on the "Create Post" button. 4. Put malicious code into the “description” section. 5. Click on the “Create” button.
Expected	The inserted malicious code is not executed.
Date-Result	28.04.2025 - Pass

Test ID	1.28	Category	UI Test	Severity	Major
Objective	Verify that users can enter data only in the correct format to email, date, and phone number fields in register and edit profile pages				
Steps	1. Go to the “Sign Up” page. 2. Enter a non-email text to email field. 3. Enter an invalid date. 4. Enter an invalid phone number. 5. Click on the “Sign Up” button.				
Expected	For each field, an error message pops up after the user enters the invalid input.				
Date-Result	28.04.2025 - Pass				

Test ID	1.29	Category	System Test	Severity	Major

Objective	Verify that API endpoints validate data before storing in the database
Steps	1. Send a request containing invalid sign up credentials to the /register endpoint of the server.
Expected	The server returns an error message indicating which field contains invalid input.
Date-Result	28.04.2025 - Pass

Test ID	1.30	Category	Functionality Test	Severity	Major
Objective	Verify that users can select their current location as a starting point for a route.				
Steps	1. Select a destination by clicking a location on the map. 2. Click on the “Select starting point” button 3. Click on the “Use my current location” button. 4. Click on the “Create” button.				
Expected	The route containing the user location as the starting point is created.				
Date-Result	28.04.2025 - Pass				

Test ID	1.31	Category	Functionality Test	Severity	Major
Objective	Verify that users can edit their profile.				
Steps	1. Log in to a user account. 2. Click on the profile picture icon on the top right corner. 3. Click on the edit icon. 4. Change data in each field. 5. Click on the “Edit” button.				
Expected	Profile of the user is updated with the entered information.				
Date-Result	28.04.2025 - Pass				

Test ID	1.32	Category	UI Test	Severity	Major
Objective	Recommendations are given as brief information cards at the bottom bar.				
Steps	1. Select destionation and create a route. 2. Wait for the engine to give recommendations.				
Expected	A slide bar with selectable recommendations are displayed at the bottom of the map area.				
Date-Result	28.04.2025 - Pass				

Test ID	1.33	Category	UI Test	Severity	Major
Objective	Recommendation selection bar should have a detailed page when dragged up.				
Steps	1. Select a destination and create a route. 2. Wait for the engine to give recommendations. 3. Pull the recommendations tab up by holding and swiping.				
Expected	The screen shows the recommendations given by the engine in detailed form.				
Date-Result	28.04.2025 - Fail				

5.2. Test Cases for Non-functional Requirements

Test ID	2.1	Category	System Test	Severity	Minor
Objective	Verify that embeddings are stored efficiently (e.g., SSD storage is properly used and memory limits are respected).				
Steps	1. Check quantization to store the embeddings and save the memory usage. 2. Check Huffman coding to store the embeddings and save the memory usage. 3. Check HDF5 to store the embeddings and save the memory				

	usage. 4. Check which one is best.
Expected	The result should be within the memory limits of the server.
Date-Result	30.04.2025 - Pass

Test ID	2.2	Category	Usability	Severity	Minor
Objective	Ensure that the POI recommendation page is not getting stuck when the recommendation engine is running				
Steps	1. Check if the application is still responsive when the backend is processing. 2. Check if the UI is informative and provides a smooth UX experience.				
Expected	The UI should be fully functional throughout the response waiting.				
Date-Result	30.04.2025 - Pass				

Test ID	2.3	Category	Usability	Severity	Minor
Objective	If for some reason the Backend API times out, the correct message should be displayed				

Steps	1. Check if the application can handle time-out error from the server. 2. Check if the application can handle 5xx errors.
Expected	The UI does not get stuck when there is an error in the system either from the frontend or the backend.
Date-Result	30.04.2025 - Pass

Test ID	2.4	Category	Functionality Test	Severity	Critical
Objective	When selecting places, the recommendations should not be far off from the user's preferences.				
Steps	1. Set user preferences. 2. Request place recommendations from the system. 3. Plot the distribution of the given recommendation tags. 4. Verify the distribution aligns with the user's preferences. 5. Check that a *k* percentage of recommendations explore new but relevant categories.				
Expected	Users receive recommendations that are related to their preferences.				
Date-Result	30.04.2025 - Pass				

Test ID	2.5	Category	Resilience Test	Severity	Major

Objective	When a fault in the backend occurs, the out-of-database places should not cease to exist.
Steps	1. Add an out-of-database place to the system. 2. Simulate a backend fault (e.g., crash or service restart). 3. Restore the backend and query for the place. 4. Verify the out-of-database place still exists and is accessible.
Expected	All place instances should remain intact.
Date-Result	30.04.2025 - Pass

Test ID	2.6	Category	Performance Testing	Severity	Major
Objective	Measures how quickly search results are returned after a place search query.				
Steps	1. Execute a search request for a popular location. 2. Record the response time. 3. Repeat with different queries. 4. Measure average response time across multiple attempts.				
Expected	Search results should return within an acceptable threshold, < 2 seconds.				
Date-Result	30.04.2025 - Pass				

Test ID	2.7	Category	Performance Testing	Severity	Major
Objective	Measure how quickly place recommendations are generated based on personal category preferences.				
Steps	1. Simulate personal category preferences for one test user. 2. Execute a place recommendation request based on a selected route. 3. Record the response time for recommendations to appear. 4. Repeat the test with different test users with different preferences. 5. Measure and analyze the average response time across multiple attempts.				
Expected	Search results should return within an acceptable threshold, < 2 seconds.				
Date-Result	30.04.2025 - Pass				

Test ID	2.8	Category	Load Testing	Severity	Major
Objective	Simulates 100 concurrent users making route searches.				
Steps	1. Deploy a test scenario with 100 simulated users. 2. Monitor server performance metrics. 3. Check if any delays or crashes occur. 4. Evaluate system scalability.				
Expected	The system should handle the load without much performance degradation.				

Date-Result	N/A

Test ID	2.9	Category	Stress Testing	Severity	Major
Objective	Tests system stability under maximum load conditions.				
Steps	1. Simulate extreme traffic with thousands of requests per second. 2. Observe system behavior and log response times. 3. Check for bottlenecks in backend processing. 4. Determine the breaking point of the system.				
Expected	The system should degrade gracefully, not crash, and maintain partial functionality under high stress.				
Date-Result	*To be filled after execution.*				

Test ID	2.10	Category	Security Testing	Severity	Major
Objective	Verify that sensitive user data, such as passwords, are encrypted when stored in the database.				

Steps	1. Register a new user with a password. 2. Access the database and retrieve stored user credentials. 3. Verify that the password is stored in a hashed format of **bcrypt**. 4. Attempt to decrypt or retrieve the original password from the database. 5. Confirm that encryption is applied correctly and that plaintext passwords are not stored.
Expected	Passwords should be securely hashed and stored using an industry-standard encryption method. Plaintext passwords must never be visible in the database.
Date-Result	30.04.2025 - Pass

Test ID	2.11	Category	Performance Testing	Severity	Major
Objective	Verify that caching mechanisms improve loading speed for frequently accessed routes, such as archived routes and social posts.				
Steps	1. Load a saved route from the archive and record the loading time. 2. Navigate to a social post containing a route and record the loading time. 3. Repeat the actions multiple times to check if caching reduces loading times. 4. Clear the cache and compare the loading times with cached results. 5. Analyze the difference in response times before and after caching.				
Expected	Loading times should decrease for repeated actions due to caching, improving overall system performance.				

Date-Result	30.04.2025 - Fail

Test ID	2.12	Category	Security Test	Severity	Major
Objective	Ensure that the authentication token of inactive users will be deactivated after 1 hour.				
Steps	1. Log in to a user account. 2. Save the received authentication token. 3. Wait for 1 hour. 4. Send a POST request to /post endpoint with the saved token.				
Expected	Server returns an error message indicating the user is not authorized to perform that action.				
Date-Result	30.04.2025 - Pass				

Test ID	2.13	Category	Security Test	Severity	Major
Objective	Ensure that a large number of requests coming from the same IP address will be throttled down.				
Steps	1. Log in to a user account. 2. Save the received authentication token. 3. Write a script that continuously sends GET requests to the /recommend endpoint with the saved authentication token.				

Expected	After the limit is reached, the server returns an error message indicating that the user exceeded the request limit.
Date-Result	30.04.2025 - Pass

Test ID	2.14	Category	Performance Test	Severity	Major
Objective	Ensure that the time required to start the application from scratch is within a reasonable interval.				
Steps	1. Clear the local cache of the test device. 2. Start the timer. 3. Start the “SürDur” application. 4. Stop the timer when the application is loaded and ready to use.				
Expected	The measured time is less than 2 seconds.				
Date-Result	30.04.2025 - Pass				

Test ID	2.15	Category	Documentation Test	Severity	Moderate

Objective	Ensure that the system documentation is complete, accurate, and aligns with the implemented features.
Steps	1. Open the latest version of the project documentation. 2. Compare system architecture details with the actual implementation in the codebase. 3. Verify that API endpoints documented match the actual backend API specifications. 4. Check if all features listed in the documentation exist and function correctly in the application. 5. Ensure that user guides, installation instructions, and troubleshooting steps are clear and up to date. 6. Identify any outdated, missing, or inconsistent information.
Expected	Documentation should accurately reflect the current system architecture, features, and API endpoints. User guides should provide clear and correct instructions. No missing, outdated, or conflicting details should be in the documentation.
Date-Result	30.04.2025 - Pass

6. Maintenance Plan and Details

6.1. Agile-Based Maintenance Approach

Just like the pre-launch maintenance, SürDur's post-launch maintenance will follow the Agile methodology, continuing the sprint-based work model used during development. After release, the development team will organize two-week maintenance sprints focusing on bug fixes, performance improvements, and small feature updates. The backlog will be continuously updated based on system monitoring outputs and user feedback, ensuring iterative enhancements without requiring major overhauls.

Emergency issues such as server downtime or API failure will be handled outside the regular sprint cycle as critical hotfix tasks, while planned improvements and optimizations will be scheduled into upcoming sprints.

6.2. User Feedback Collection and Processing

User feedback is gathered primarily from in-app feedback forms, App Store and Google Play Store reviews, and social media engagement. Feedback is regularly categorized into usability issues, feature requests, and bug reports, and logged into the team's issue tracker on “Github Projects” that we are already managing the issue organization.

Prioritization of user-reported issues are considered issue topic, frequency and user impact. Critical feedback affecting navigation accuracy or application stability were immediately assigned to the active sprint, while less urgent requests are scheduled for later iterations.

6.3. Version Control and Release Management

SürDur utilizes the Git and GitHub portal for version control, following a structured Git flow strategy. Development work occurs on side feature branches, which are merged into a staging branch for integration testing before final releases are merged into the main branch.

GitHub Actions have been integrated into the private Backend repository to automate deployment pipelines to enhance release efficiency and security. Hotfixes are branched directly from the production branch to allow rapid deployment without disrupting ongoing development. For future enhancements, each production release follows semantic versioning, like ‘Major’, ‘Minor’ and ‘Patch’.

6.4. Monitoring Strategy

Backend server health, API response times, database performance, and application crash reports are monitored using AWS CloudWatch and in the future, mobile crash reporting services such as Firebase Crashlytics.

In addition to real-time alerts for major problems, weekly system performance summaries are reviewed to detect trends such as memory leaks, slow queries, or increased API call failures. Monitoring results are feeded directly into maintenance sprints to ensure early mitigation of potential issues.

6.5. Deployment and Update Strategy

Application updates are deployed to Google Play Store and App Store through standard release pipelines, with submission following respective platform guidelines. Minor updates are planned to occur monthly, while major updates introducing new features or design changes occur quarterly.

Each release is accompanied by detailed changelogs and version numbers to maintain transparency. Pre-deployment testing on staging environments ensures compatibility across Android, iOS, and CarPlay devices, safeguarding user experience during updates.

6.6. Backup and Disaster Recovery Plan

The backend service maintains daily automated backups of both application data and the main place database. Backup snapshots are retained for a minimum of 30 days to allow quick recovery from accidental deletions or server failures.

In case of catastrophic system failure, the team restores services by spinning up redundant AWS instances and restoring the latest database snapshot. A documented disaster recovery playbook guides the recovery process, minimizing downtime and user impact.

6.7. Third-Party Services Maintenance

Since SürDur relies heavily on third-party APIs such as OpenRouteService for route formation, route editing and fetching relevant route information. Regular audits are conducted to check API endpoint change notices and service reliability. Even though OpenRouteService is an open-source service, possible pricing model changes are regularly audited as well.

If an external service shows signs of instability or change, alternative services will be researched, tested, and integrated where necessary. Dependency libraries and SDKs related to these APIs will be updated quarterly to ensure compatibility and security compliance.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

7.1.1. Constraints

This section discusses the SürDur project's constraints in detail on aspects of development, economic, technological, social, safety, and sustainability. Also, the effects of global, cultural, social, environmental and economic factors on the app is given in the following table:

	Effect Level	Effect
Global	9	● Take different roads, places into consideration. ● Limitations of sources, passes from one country to another may affect the whole design.
Cultural	9	● Cultural preferences should be taken into consideration in place recommendation. ● Local cultural point of interests should be taken into consideration.
Social	9	● Social post interactions between users should be considered on personal preference loggings. ● Follower-following system should be considered on social content display.
Environmental	5	● Less fuel consumption should be taken into consideration in route generation.
Economic	7	● Budget availability by the user should be taken into consideration in place suggestions. ● Server maintenance fees should be evaluated to decide how large amount of place data the application can hold.

Table 2: Factors that can affect analysis and design

7.1.1.1. Development Constraints

- The project app is available for both iOS and Android.
- The mobile side of the project has been developed using React Native as it's compatible with both IOS and Android.
- The application has been developed with Python and FastAPI for the back end and React-Native for the front end.
- In categorizing POIs, NLP methodologies and tools have been used to help mapping many categories into predetermined categories.
- Git and Github have been used as our version control system.
- MySQL have been used to store and access database components related to both the application engine and the user data.
- The backend services and the database are kept on AWS servers.
- Github Projects have been used to keep track of the development process.

7.1.1.2. Economic Constraints

- The database is kept on AWS servers. Annual payment for reserving a micro-sized server space (1 GB of data space) from Stockholm servers requires $94 [1].
- Publishing the app on mobile platforms has two economic constraints. One is the $25 one-time registration fee on the Google Play Store (Android), and the other is the annual $99 fee on the App Store (IOS) [2].
- Frameworks and libraries that are used to implement the project such as FastAPI, React Native, and Expo, are free to use.

7.1.1.3. Technological Constraints

- The application needs an internet connection for all the functionalities, such as route creation, navigation, social functions, and profile operations.
- The application has to access the user's location on route creation, and navigation functions.

7.1.1.4. Social Constraints

- The application allows the sharing of previously followed routes.
- The public route posts include a title and a header which allows a detailed explanation of the route. However, there is no further text-based communication allowed on those posts.

- The posts have a voting system that shows the public appreciation of users' posts.

7.1.1.5. Safety Constraints

- Mobile app decisions are made to minimize user interaction during a car ride.
- The users are asked to make choices before starting the ride.
- The live navigation service of SürDur is designed primarily around driver safety. Therefore, the effect level of safety is **10** out of 10.

7.1.1.6. Sustainability Constraints

- Server and publishing services need to be paid annually.
- An increase in the number of users may result in database enlargement, which may result in higher server space costs.

7.1.2. Standards

7.1.2.1. IEEE 1471

Purpose:

IEEE 1471, a software-intensive system architecture documentation standard, is ISO/IEC/IEEE 42010. It outlines developing an architecture description that offers a shared comprehension of the system's structure, functionality, and essential characteristics [8]. We can more easily comprehend system components and their relationships thanks to IEEE 1471's assistance in clarifying the architecture. This entails outlining the architecture's background, perspectives, interested parties, and the reasoning behind essential choices [3].

Key Elements:

We document architectural choices in development; this outlines essential decisions made during the design process, supporting information, and other factors. Also, documenting architectural views in the project enables us to represent the project's physical, process, development, and logical aspects.

7.1.2.2. UML 2.5.1

Purpose

A widely used modeling language for describing, building, visualizing, and recording the structure and behavior of software systems is UML 2.5.1 [9]. It provides

a consistent method for drawing diagrams that explain various system components. We can understandably display the system's structural and functional elements using UML. This standard facilitates the creation of models for different views (such as class, sequence, and activity diagrams), which helps with system design and communication [4].

Key Aspects

Class, Component, and Deployment aspects define the system's static structure. Use Case, Sequence, and Activity represent dynamic aspects of the system, including interactions and workflows.

7.1.2.3. IEEE 830

Purpose

Writing Software Requirements Specifications (SRS) is standardized by IEEE 830. It establishes a thorough framework for recording functional and non-functional requirements, guaranteeing accuracy, consistency, and comprehensiveness [10]. IEEE 830 offers a systematic style for specs reports that assists teams in organizing requirements for easy understanding and verification by stakeholders, developers, and testers. Project objectives, scope, requirements, assumptions, and restrictions are all covered in this standard [5].

Key Aspects

This standard addresses the project's background, goal, and extent. Additionally, it gives a summary of the operating environment, user attributes, and product capabilities.

7.1.2.4. ISO 31000

Purpose

One standard that offers recommendations for efficient risk management is ISO 31000. It aids businesses in recognizing, evaluating, and reducing risks, which enhances decision-making and reduces uncertainty [11]. This standard exemplifies proactive risk management by addressing potential project risks (technical, operational, and financial) and mitigation techniques. Risk assessment frameworks, prioritization, and controls are a few examples [6].

Key Aspects

Identifying potential risks that have an impact on the project. Assessing the impact and probability of hazards that have been discovered. establishing measures to reduce or eliminate risks. We can identify possible problems and dangers by implementing risk management.

7.1.2.5. IEEE Citation Style

Purpose

IEEE Citation Style is a widely standardized approach for citing sources from engineering, information technology, and allied fields. It increases the traceability and dependability of the information by ensuring that sources are consistently mentioned. IEEE Citation Style provides a uniform method of referring to external sources (including research papers, technical publications, and standards) that ensures accuracy and lucidity. When citations are appropriately formatted, readers may locate sources for further context and proof [7].

Key Aspects

References match the list of references and are numbered in brackets (e.g., [1], [2])[12]. provides comprehensive information for every source and arranges citations in numerical order.

7.2. Ethics and Professional Responsibilities

7.2.1. Professional Responsibilities

- To ensure transparency and inclusivity, end users were kept informed throughout the project's lifecycle. This was achieved through surveys and notifications regarding any design changes or finalizations. Their valuable feedback was actively incorporated into the development process.
- A demo version of the application is presented to end users to collect their opinions and tailor the app to align with market expectations.
- The construction of the database prioritized equal representation across all cities and regions of Turkey. However, due to the natural non-uniform distribution of human-made stops such as restaurants, the density of solutions may vary depending on the region.

7.2.2. Ethical Responsibilities

- User geolocation data is strictly used for generating routes and navigation purposes. This data is neither stored nor shared under any circumstances.
- Sensitive personal information, such as recommendation details and past trip data, remains confidential and inaccessible to third parties.
- Publicly visible information, including name, surname, profile photo, and shared routes, can only be displayed within the app interface. Outside of the app, this information is safeguarded to ensure user privacy.

7.3. Teamwork Details

7.3.1. Contributing and Functioning Effectively on the Team

7.3.1.1. Bora Haliloğlu

- Recommendation Service Design and Implementation
- Created Frontend components

7.3.1.2. Burak Oruk

- Helped to design the flow and handling of data to be used in the recommendation service.
- Evaluated and processed the data to be inserted into database by the following methods:
    - Using K-Means algorithm, grouped similar place categories.
    - Unbundled some of the clusters and made some hand-picking to increase precision of the category groupings to be used in the recommendation service.
- Helped implement some minor backend services.

7.3.1.3. Emir Tuğlu

- Deployed the backend services to the cloud using AWS infrastructure, including EC2, RDS, and S3.
- Implemented various backend services and integrated them with the rest of the system.
- Contributed to the development of several frontend pages and components, and integrated them with the backend APIs.

7.3.1.4. Mustafa Gökalp Gökdoğan

- Implemented route related backend services and DAOs.
- Created the frontend structure and implemented many of the pages.
- Wrote the API scripts to fetch our places. Also, visualized these data.

7.3.1.5. Tevfik Emre Sungur

- Constructed the Data Access Layer structure on the backend services, mostly on social page posts, and its implicit relationships (e.g. upvoting / downvoting).
- Formed the MySQL database table structure and relationships.
- Integrated front and back social page services together, executed their relative unit tests.

7.3.2. Helping to Create a Collaborative and Inclusive Environment

To create a friendly and collaborative environment, we have organized small groups to work on certain tasks, called work packages. Because team members are already familiar with one another's abilities from previous interactions, tasks are assigned based on individual capabilities. This strategy guarantees that each person makes an equitable and significant contribution to the project.

To ensure that no one feels left behind, team members can also ask any work package assignee for help if they run into problems. By successfully putting this structure into practice, we hope to create a helpful, collaborative environment where everyone feels involved and included.

7.3.3. Taking a Lead Role and Sharing Leadership on the Team

“Projects benefit greatly from having a leader because there is just one person for the team to look up to, which makes progress easier and faster. We separated our tasks into work packages and designated one person as the leader of each package so that no one person would be overburdened with the leadership responsibilities. Below is comprehensive information on the work packages.

WP#	Work package title	Leader

WP1	Project Specification Document	Gökalp Gökdoğan
WP2	Analysis and Requirement Report	Tevfik Emre Sungur
WP3	Frontend Development	Gökalp Gökdoğan
WP4	Backend Development	Burak Oruk
WP5	Setting up the Database	Tevfik Emre Sungur
WP6	Recommendation System Development	Emir Tuğlu
WP7	Demo	Bora Haliloğlu
WP8	Detailed Design Report	Emir Tuğlu
WP9	Design Project Final Report	Bora Haliloğlu
WP10	App Launch	Burak Oruk
WP11	Final Demo	Gökalp Gökdoğan

WP1: Project Specification Document
Start Date: 12 November 2024 End Date: 22 November 2024

Leader	Gökalp Gökdoğan	Members Involved	All Members
Objectives: Prepare and deliver the Project Specification Document.			
Tasks: **Task 1.1 Writing an Introduction:** Describe the project in detail. Describe the type of innovation that is being sought. Identify the limitations and ethical and professional concerns. **Task 1.2 Writing Requirements:** Describe the functional and non-functional requirements in your writing. **Task 1.3 Writing Ongoing Discussions:** Provide information on any ambiguities in the project's specifics and outline potential future directions. **Task 1.4 Writing References:** Use the proper citation formats and include references for all sources used in the report.			
Deliverables: **D1.1:** Project Specification Document			

WP2: Analysis and Requirement Report			
Start Date: 3 December 2024 End Date: 16 December 2024			
Leader	Tevfik Emre Sungur	Members Involved	All Members
Objectives: Prepare and deliver the Analysis and Requirement Report.			
Tasks:			

Task 2.1 Scenarios

Task 2.2 Creation of Use-Case Diagram

Task 2.3 Creation of Object and Class Model

Task 2.4 Creation of Dynamic Models: Create Activity, Sequence, and State Diagrams.

Task 2.5 Creation of UI Designs

Task 2.6 Other Analysis Elements: Identify the options and hazards, and describe the elements that influenced the design. Additionally, describe the professional and ethical obligations. Provide a thorough project strategy as well as a road map for gaining the technical know-how required for the future.

Task 2.7 References: Use the proper citation formats and include references for all sources used in the report.

Deliverables:

D2.1: Analysis and Requirement Report

WP3. Frontend Development			
Start Date: 29 November 2024 End Date: May 2025			
Leader	Gökalp Gökdoğan	Members Involved	Bora Haliloğlu Gökalp Gökdoğan Emir Tuğlu

Objectives: Implementation of the front-end of the application according to the UI Designs created for the Analysis Report.
Tasks: **Task 3.1 Implementation of Log-in & Sign-up pages** **Task 3.2 Implementation of the Onboarding Page** **Task 3.3 Implementation of the Main Page** **Task 3.4 Implementation of Suggestion Page** **Task 3.5 Implementation of the Navigation Page** **Task 3.6 Implementation of the Search Page** **Task 3.7 Connect front-end to back-end** **Task 3.8 Optimizing performance of application**
Deliverables: **D3.1:** The Frontend of the app.

WP4: Backend Development			
Start Date:16 November 2024 End Date: May 2025			
Leader	Burak Oruk	Members Involved	Tevfik Emre Sungur Burak Oruk Emir Tuğlu Gökalp Gökdoğan
Objectives:Implementation of the back-end of the application according to the			

design proposed in the Analysis Document.
Tasks: **Task 2.1: Initializing FastAPI project with correct dependencies** **Task 2.2: Implementation of basic classes according to class diagram** **Task 2.3: Implementing service layer** **Task 2.4: Connecting the external services to service classes** **Task 2.5: Testing controller endpoints via postman** **Task 2.6: Connecting the back-end with front-end** **Task 2.7: Deployment to AWS**
Deliverables: **D2.1:** The back-end application

WP5: Setting up the Database			
Start Date:16 November 2024 End Date: February 2025			
Leader	Tevfik Emre Sungur	Members Involved	Tevfik Emre Sungur
Objectives:Designing and creating a database that can store high volume of location data and allows low latency data retrieval.			

Tasks:

Task 5.1: Design the database schema

Task 5.2: Create tables according to the design

Task 5.3: Collect and standardize data from different APIs and blogs: Write scripts to collect data from the POI APIs and blogs. Then, standardize this data to the same format and eliminate duplicate data before storing in the database.

Task 5.4: Populate tables with the retrieved data: Save collected and standardized data into the database.

Deliverables:

D5.1: The database that contains POI information.

WP6: Recommendation System Development			
Start Date: January 2025 End Date: May 2025			
Leader	Emir Tuğlu	Members Involved	All Members
Objectives:Developing a recommendation system algorithm to provide users personalized POI recommendations.			

Tasks:

Task 6.1 Create Embeddings: Create the embeddings for the categories and store them.

Task 6.2 Create Embedding Logic: Create the logic for the recommendation for the embeddings.

Task 6.3 Create the Exploration System: Create the exploration logic.

Deliverables:

D6.1: The recommendation system that provides personalized recommendations according to users' preferences.

WP7: Demo			
Start Date: 16 December 2024 End Date: 20 December 2024			
Leader	Bora Haliloğlu	Members Involved	All Members
Objectives: Prepare and deliver the Demo			

Tasks:

Task 7.1 Prepare Slides: Prepare slides about the project, the problem that project solves, market and competitor analysis, business model, etc.

Task 7.2 Prepare Demo: Prepare a demo to display implemented functionality of the system.

Task 7.3 Present:

Deliverables:

D7.1: Demo

WP8: Detailed Design Report
Start Date: February 2025 End Date: March 2025

Leader	Emir Tuğlu	Members Involved	All Members
Objectives: Prepare and deliver the Detailed Design Report			
Tasks: **Task 8.1 Determine design goals:** Usability, performance, reliability, marketability, etc. **Task 8.2 Sketch the architecture of the system** **Task 8.3 Explain subsystem services** **Task 8.4 Define functional and non-functional test cases** **Task 8.5 Discuss teamwork details**			
Deliverables:D8.1: Detailed Design Report			

WP9: Design Project Final Report			
Start Date: April 2025 End Date:May 2025			
Leader	Bora Haliloğlu	Members Involved	All Members
Objectives: Prepare and deliver the Design Project Final Report			
Tasks: **Task 9.1 Write down requirements details:** **Task 9.2 Sketch the final architecture** **Task 9.3 Provide development and implementation details**			

Task 9.4 Give information about test cases and results **Task 9.5 Discuss maintenance plan** **Task 9.6 Discuss other project elements:** Constraints, standards, ethics and professional responsibilities, teamwork details etc.
Deliverables: **D9.1:** Design Project Final Report

WP10: App Launch			
Start Date: May 2025 End Date: May 2025			
Leader	Burak Oruk	Members Involved	All Members
Objectives: Launch the app			
Tasks: **Task 10.1 Test the app:** Ensure each functionality is working as expected **Task 10.2 Launch the app on the App Store** **Task 10.3 Launch the app on the Play Store**			
Deliverables: **D10.1:** The app that can be downloaded by iOS and Android devices			

WP11: Final Demo			
Start Date: May 2025 End Date: May 2025			
Leader	Gökalp Gökdoğan	Members Involved	All Members
Objectives: Prepare and deliver the Final Demo			
Tasks: **Task 11.1 Prepare Slides:** Prepare slides about the project, the problem that project solves, market and competitor analysis, business model, etc. **Task 11.2 Prepare Demo:** Prepare a demo in which functionalities of the app are displayed. **Task 11.3 Present**			
Deliverables: **D1.1:** Final Demo			

7.3.4. Meeting Objectives

At the beginning of the project, we set out to build a personalized travel assistant application that not only provides standard navigation features but also suggests meaningful stopovers based on user preferences. Our initial objectives included designing a modular system architecture, implementing a functional recommendation engine, enabling route creation and editing, and supporting social interaction features like route sharing and following other users. From the early planning stages, these objectives shaped our decisions on system design, technology stack, and task distribution within the team. We made sure to document our goals and reflect them in our GitHub Project board and internal documentation to maintain clarity.

Throughout the semester, we consistently monitored our progress with respect to the objectives. We held weekly meetings to review completed tasks, prioritize upcoming items, and address any blockers that could prevent us from staying on track. Whenever we needed to adjust the scope—for example, due to time or infrastructure limitations—we made sure that the changes still aligned with our core project goals. Our regular use of agile-style sprint planning and version tracking helped us stay focused and organized during development. This approach also allowed us to adapt to changes without losing sight of the overall direction of the project.

In the end, we believe we met the main objectives we initially defined. The core features of the application—including personalized point-of-interest recommendations, route update options, profile editing, archives, and social sharing—were all implemented and tested. The system runs reliably across Android and iOS platforms, and the backend services are hosted and live on AWS servers. Although we initially aimed to support CarPlay integration to extend usability for in-vehicle navigation, this feature was not implemented due to time constraints. Nonetheless, we successfully delivered a working product that fulfills the primary expectations. Our team is satisfied with the outcomes, and we believe the project met its purpose both technically and functionally.

7.4. New Knowledge Acquired and Applied

Throughout the development of the SürDur project, our team had the opportunity to learn and apply a wide range of new technologies, methodologies, and tools that we had limited or no experience with before. One of the most significant areas of learning was the use of large-scale real-world data for building a recommendation system. We worked with over 70,000 place entries from FourSquare, and through this process, we gained experience in data cleaning, semantic clustering, and vector-based similarity calculations using techniques such as cosine similarity and the OpenAI embedding API. This helped us understand how machine learning concepts can be applied practically in a real application.

On the backend side, we deepened our knowledge in developing backend services using FastAPI. We also became more comfortable with deploying services to the cloud using AWS tools like EC2, RDS, and S3. Setting up authentication using OAuth2, implementing rate limiting, and ensuring secure data access gave us

hands-on experience in backend security and robustness. In addition, we learned to use Python virtualization and GitHub Actions for automating deployment pipelines, which significantly improved our DevOps skills.

From a frontend and mobile development perspective, working with React Native allowed us to implement cross-platform features that run on both Android and iOS devices. We also learned to design responsive UI components and integrate external APIs such as Google Maps and OpenRouteService. Overall, this project provided us with a valuable opportunity to connect the theoretical knowledge gained during our undergraduate education with practical, real-world software engineering challenges. Each team member was exposed to new technologies and responsibilities that contributed to both individual and collective growth.

8. Conclusion and Future Work

Overall, SürDur aims to enhance road trip experiences by offering users personalized stopover recommendations along the route and integrated live navigation. The application's personalized point-of-interest suggestion system, dynamic route adjustment, and social media features work together to create a richer and more interactive journey for travelers. This version of our application successfully delivers all core functionalities planned, including destination selection, route creation, personalized POI recommendation, live navigation, social sharing and personal archive features.

However, there are still areas open to future development. Occasionally, minor delays occur when retrieving place data from third-party APIs, which can briefly affect the responsiveness of the suggestion system. Although this does not impact the overall functionality, optimizing data retrieval speed is a priority for the next iterations. Additionally, the personalized recommendation system can be further improved by introducing more reliable user profiling techniques based on data collected from active users in a timespan. In future versions, the application will expand its database to include more diverse stopovers, implement additional route filters such as budget-friendly options, and introduce a real-time incident alert system during navigation.

9. Glossary

- **Point of Interest (POI)** – A location that may be of interest to a traveler, such as restaurants, tourist attractions, gas stations, and rest stops.
- **Route Planning** – The process of determining the best path from a starting point to a destination, considering factors such as road conditions, distance, and travel preferences.
- **Personalized Recommendations** – Suggestions for POIs that are tailored to a user's preferences, travel history, and behavior.
- **Navigation System** – A digital system that provides real-time route guidance using GPS and mapping technologies.
- **Social Media Integration** – The incorporation of social networking features, such as sharing routes, voting on places, and following other users.
- **Caching** – The temporary storage of frequently accessed data to reduce retrieval time and improve performance.
- **Machine Learning (ML)** – A subset of artificial intelligence that enables a system to learn from data and improve its recommendations over time.
- **OAuth2** – An authorization framework that enables third-party applications to grant access to user accounts without exposing login credentials.
- **API** – Application Programming Interface
- **AWS** – Amazon Web Services
- **CRUD** – Create, Read, Update, Delete (basic database operations)
- **DBMS** – Database Management System
- **DDoS** – Distributed Denial of Service (a cyber attack)
- **GPS** – Global Positioning System
- **ML** – Machine Learning
- **ORM** – Object-Relational Mapping (a programming technique for interacting with databases)
- **POI** – Point of Interest
- **RDS** – Relational Database Service (AWS service for database hosting)
- **SQL** – Structured Query Language (used for managing databases)
- **UI** – User Interface
- **UX** – User Experience

10. References

[1] Amazon Web Services, "Amazon RDS for MySQL pricing," [Online]. Available: https://aws.amazon.com/tr/rds/mysql/pricing/?pg=pr&loc=2. [Accessed: Nov. 19, 2024].

[2] Sphinx Solution, "Cost to put an app on the App Store," [Online]. Available: https://www.sphinx-solution.com/blog/cost-to-put-an-app-on-the-app-store/. [Accessed: Nov. 19, 2024].

[3] IEEE, "IEEE Standard 1471: Recommended practice for architectural description of software-intensive systems," [Online]. Available: https://standards.ieee.org/ieee/1471/2187/. [Accessed: Nov. 19, 2024].

[4] Object Management Group, "Unified Modeling Language (UML), version 2.5.1," [Online]. Available: https://www.omg.org/spec/UML/2.5.1/About-UML/. [Accessed: Nov. 19, 2024].

[5] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language user guide," IEEE, 1999. [Online]. Available: https://ieeexplore.ieee.org/document/720574. [Accessed: Nov. 19, 2024].

[6] International Organization for Standardization, "ISO 31000: Risk management," [Online]. Available: https://www.iso.org/iso-31000-risk-management.html/. [Accessed: Nov. 19, 2024].

[7] New Jersey Institute of Technology, "IEEE citation style guide," [Online]. Available: https://researchguides.njit.edu/ieee-citation/ieeereferencing/. [Accessed: Nov. 19, 2024].

[8] W. Pree, "Design patterns for object-oriented software development," IEEE, 1995. [Online]. Available: https://ieeexplore.ieee.org/document/875998/. [Accessed: Nov. 19, 2024].

[9] UML Diagrams, "UML 2.5 diagrams overview," [Online]. Available: https://www.uml-diagrams.org/uml-25-diagrams.html. [Accessed: Nov. 19, 2024].

[10] IEEE, "IEEE Standard 830: Recommended practice for software requirements specifications," [Online]. Available: https://standards.ieee.org/ieee/830/1222/. [Accessed: Nov. 19, 2024].

[11] International Organization for Standardization, "ISO 9001: Quality management systems," [Online]. Available: https://scc.isolutions.iso.org/obp/ui#iso:pub:PUB100464. [Accessed: Nov. 19, 2024].
[12] George Mason University, "IEEE style citation guide," [Online]. Available: https://infoguides.gmu.edu/ieee_style#s-lg-box-29326431. [Accessed: Nov. 19, 2024].