
ChargeMind

OUR TEAM

Selin Deniz
202011051

Meleksu Özdoğan
202011054

Hanife Müge Karakaya
202011044

Lorin Melek Vural
202011035

Kaan Baydemir
202011070

Academic Advisor: Dr. Asst. Prof. Serdar Arslan

Havelsan SUIT Mentor: Berkay Yılmaz

TeamID: 202412

Table of Contents

ChargeMind in General

- **Artificial Intelligence-Based Automation for Increasing Needs of Electric Vehicle**
- Customization **User-Specific** Route Planning
  - Intensity According to Personal Priority,
  - Time,
  - Pricing Optimization

- Artificial Intelligence Supported **Density Estimation** and **Recommendation System**
- **Station Rating** Based on User Comments
- Eco-Friendly and Sustainable Charging Solutions with **Carbon Footprint Tracking**

WORK PLAN

Start Date 21/10/2024	WORK PLAN															
Week	Current State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Procedural Steps		30-Sep-24	7-Oct-24	14-Oct-24	21-Oct-24	28-Oct-24	4-Nov-24	11-Nov-24	18-Nov-24	25-Nov-24	2-Dec-24	9-Dec-24	16-Dec-24	23-Dec-24	30-Dec-24	6-Jan-25	13-Jan-25
Team Setup	Completed																
Project Proposal Form	Completed																
Project Selection Form	Completed																
Project Work Plan	Completed																
Literature Review	Completed																
Marketing Research	Completed																
Software Requirements Specification	Completed																
Project Webpage	Completed																
Software Design Description	Completed																
Project Report / Tracking Form	Completed																
Presentation	Completed																

Market Research
Features of Current Applications

Lixhium

- Navigation Integration: It has the feature of directing users to the selected charging station.

Voltla

- Navigation Integration: Provides direct guidance to charging stations.

ChargePrice

- Price Comparison: Provides cost-effective options by comparing prices of different charging providers.
- User-Friendly Price Optimization: Price-oriented optimization; focuses especially on affordable charging options.

PlugShare

- User Comments and Evaluations and Station Features
- Station Availability Notification: Instantly shows whether the charging station is empty or occupied.

Open ChargeMap

- Similar Features with PlugShare
  - PlugShare Directs User to Google Maps
  - Open Charge Map Creates Its Own Map

Problems

Incorrect and insufficiency route determination

Unnecessary use of areas reserved for charging stations

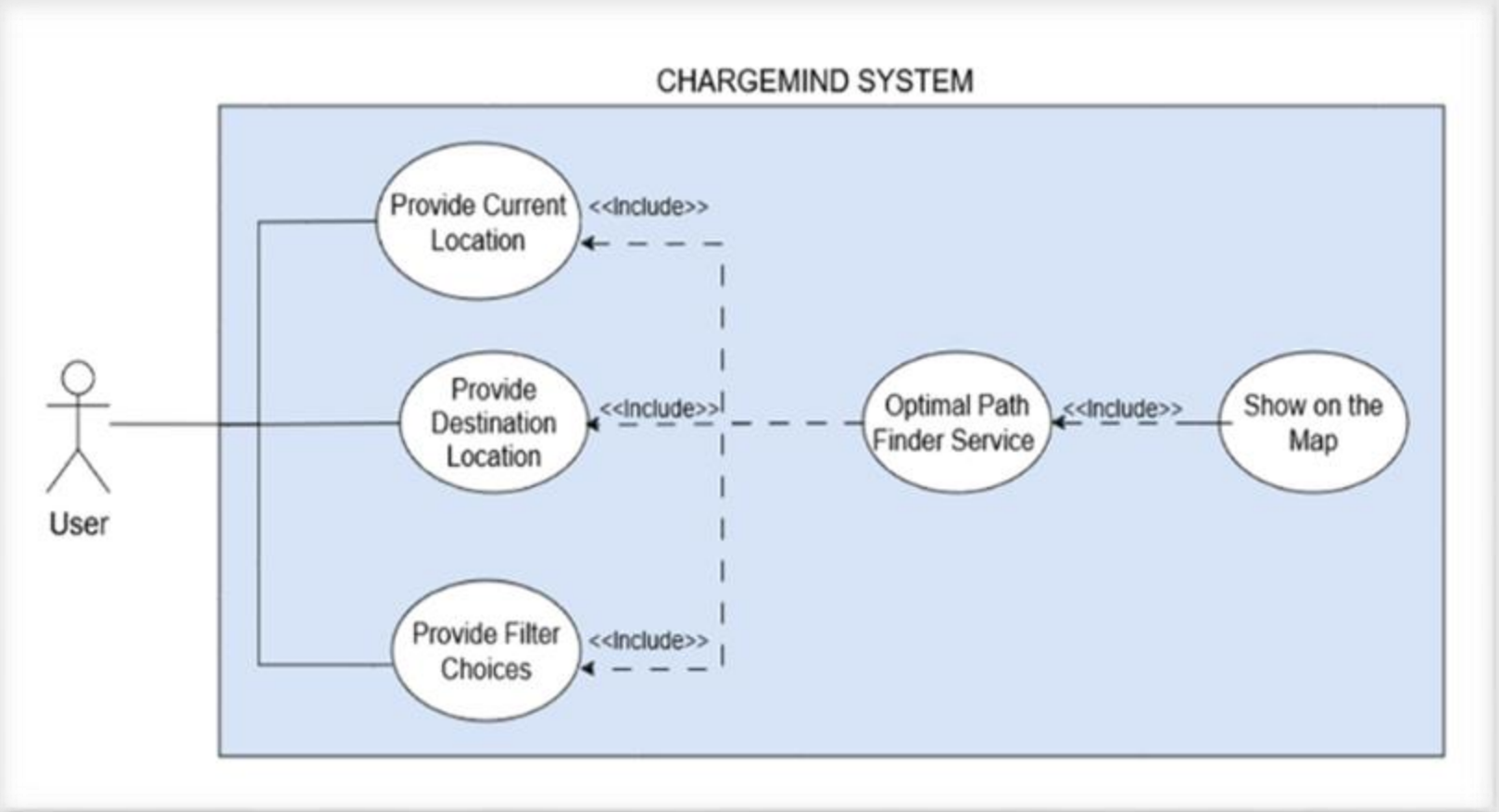
Misleading directions about station locations

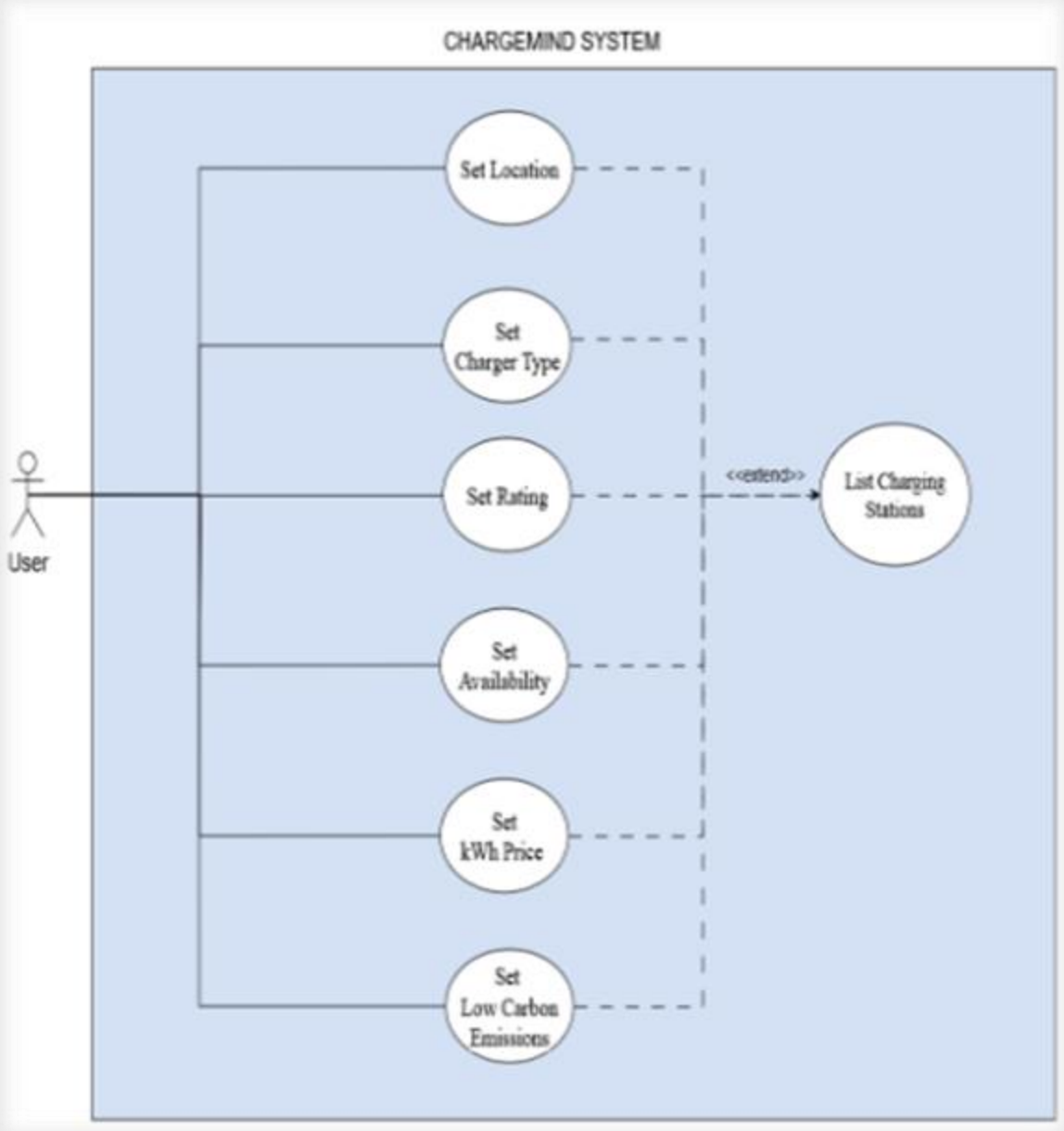
Inability to charge due to excessive density

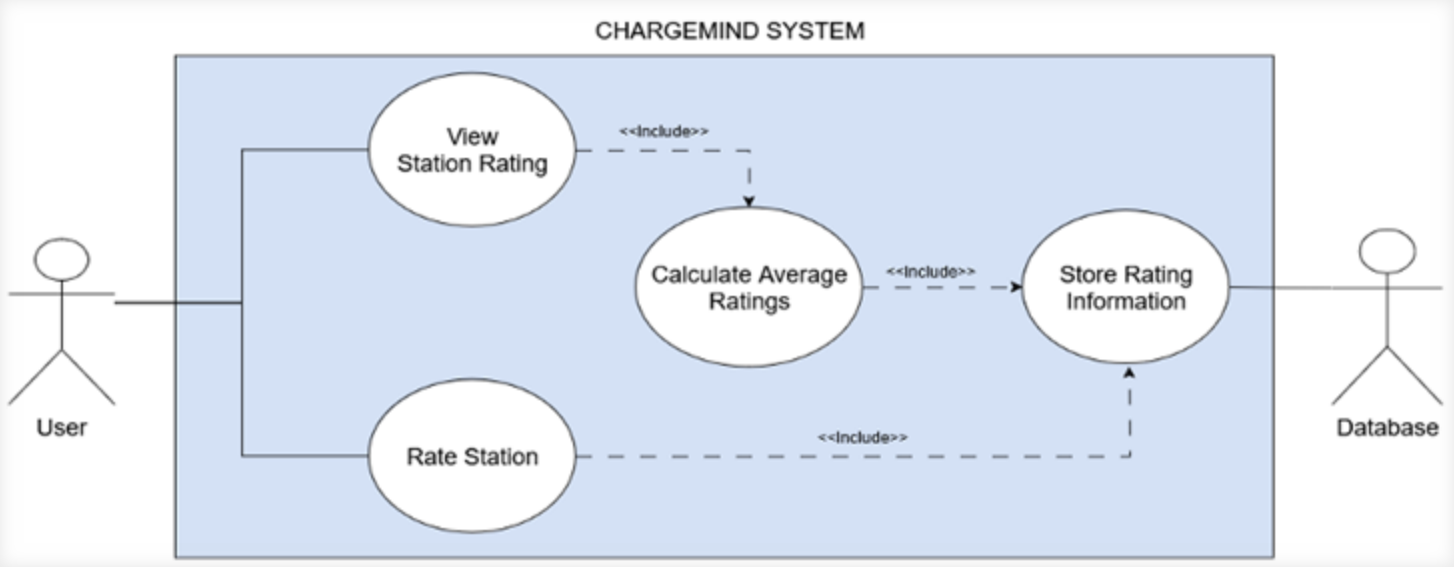
User Feedback

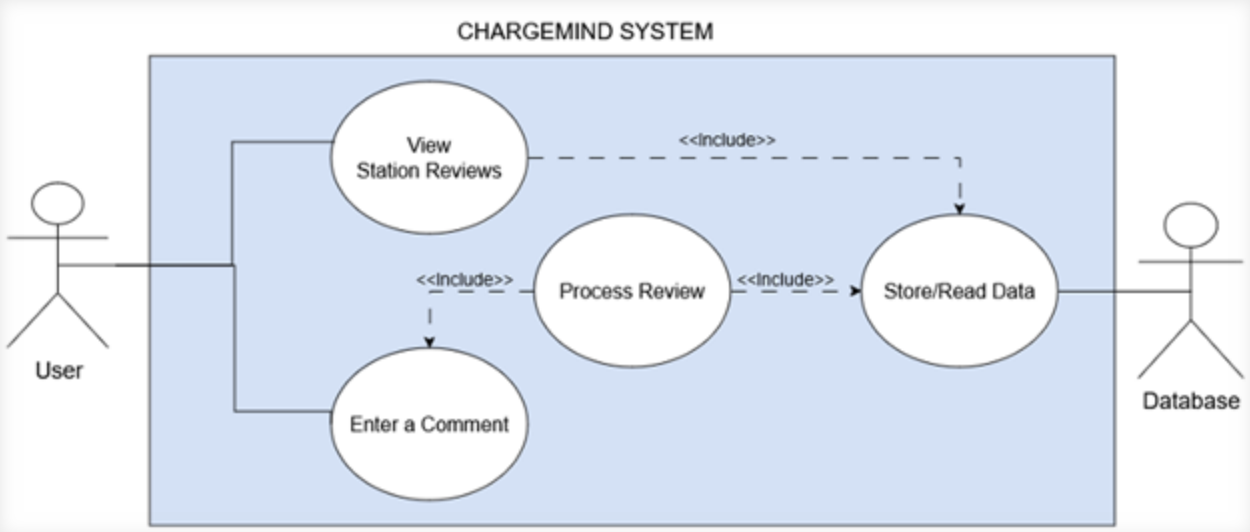
MAIN FUNCTIONALITIES

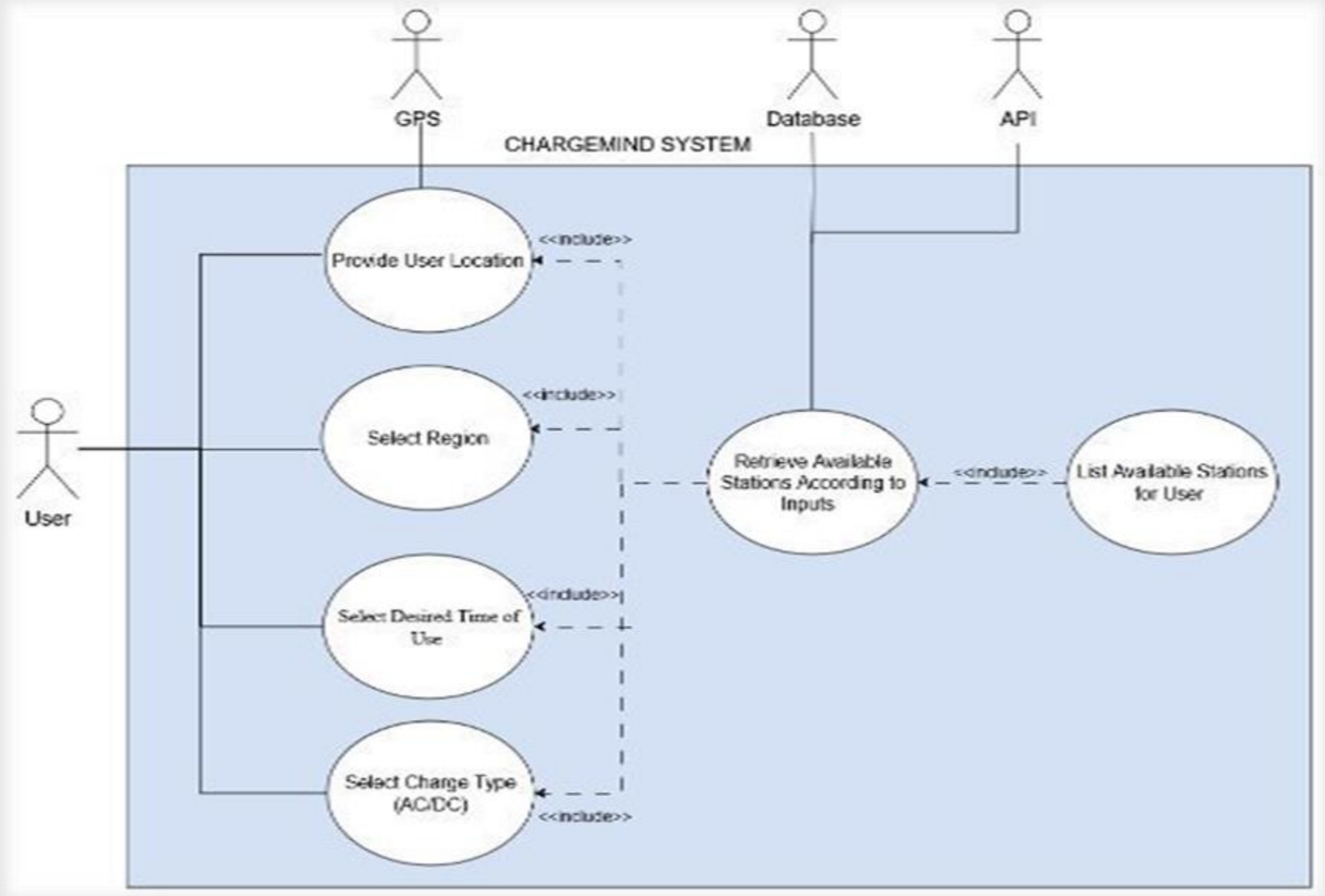
- **Plan Personalized Route For EV**
- **Filter**
- **User Reviews and Ratings**
- **Real-Time Availability Updates**
- **Recommendation System**
- **Reservation System**
- **Carbon Emission Calculation**

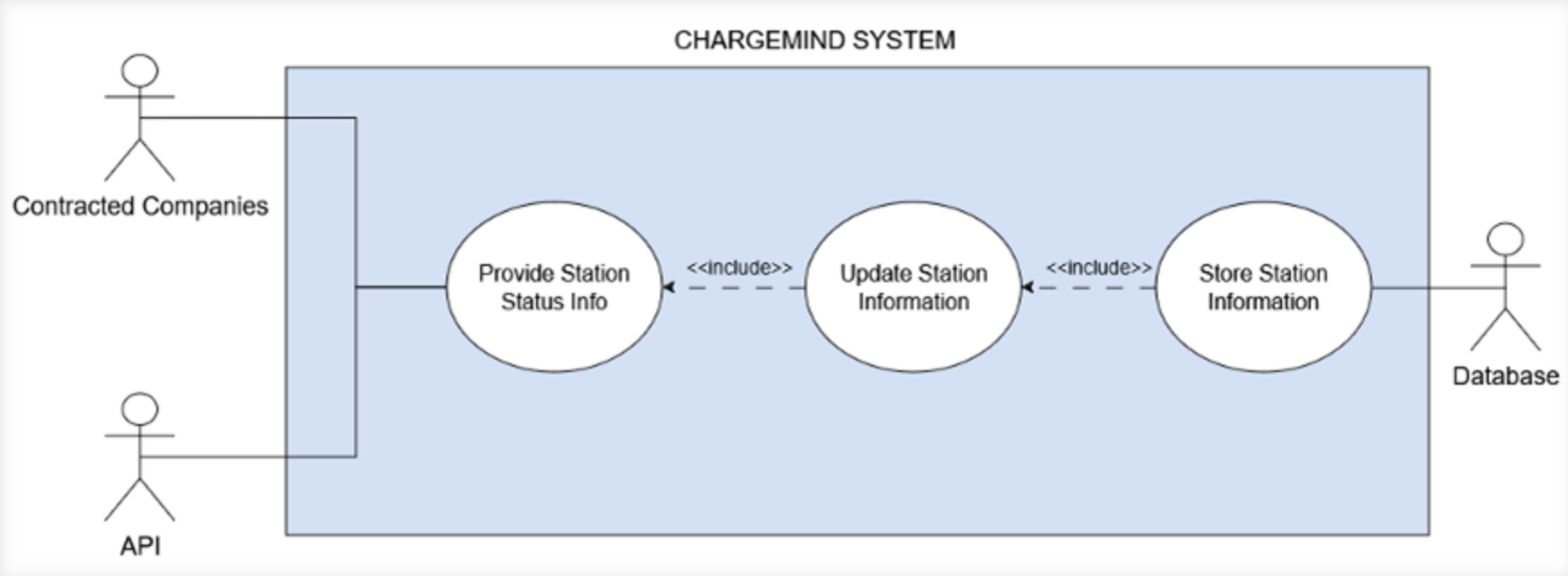

CHARGEMIND SYSTEM
Provide Current Location
<<Include>>
Provide Destination Location
<<Include>>
Optimal Path Finder Service
<<Include>>
Show on the Map
Provide Filter Choices
<<Include>>
User

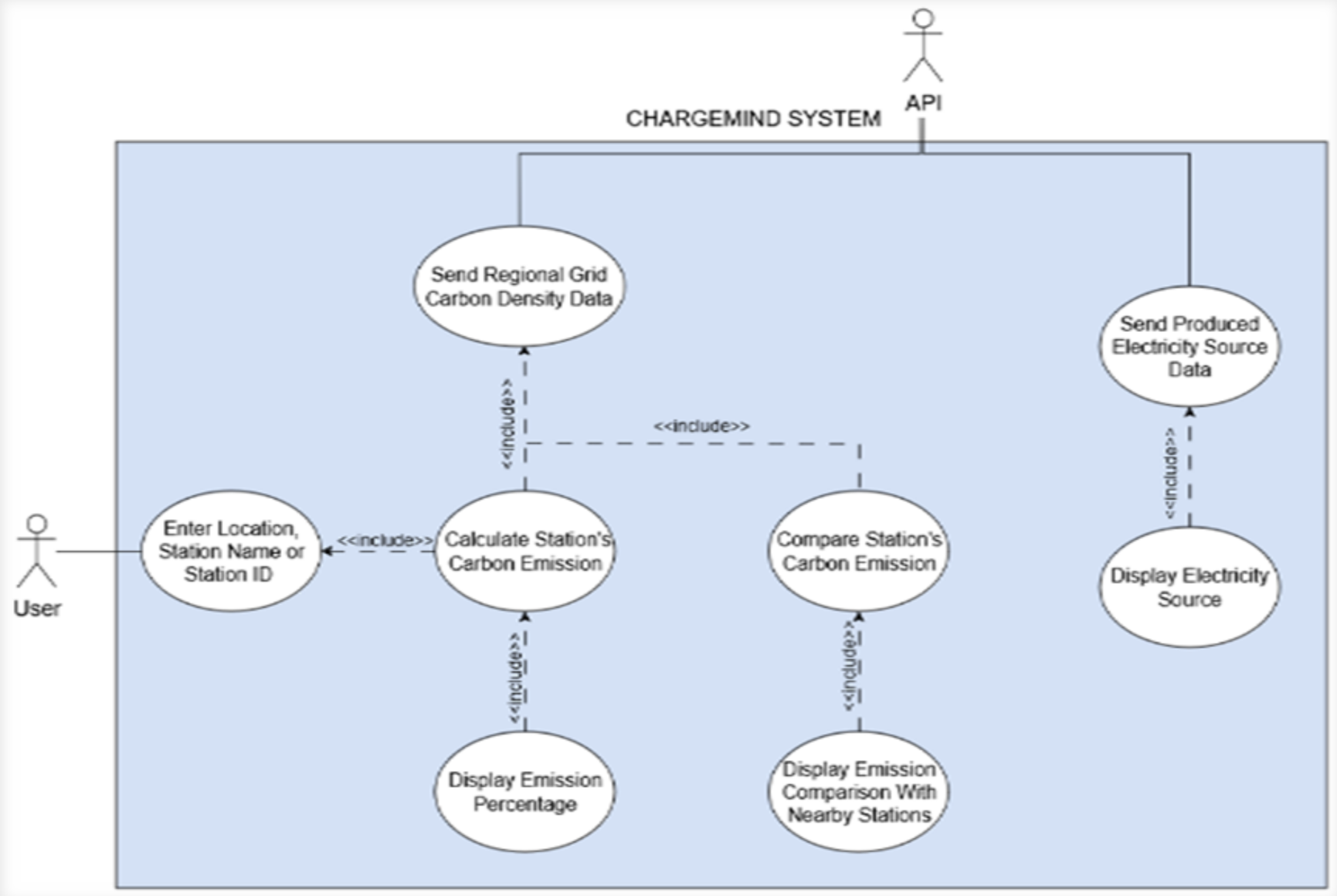

CHARGEMIND SYSTEM
Set Location
Set Charger Type
Set Rating
Set Availability
Set kWh Price
Set Low Carbon Emissions
<<extend>>
List Charging Stations
User


CHARGEMIND SYSTEM
View
Station Rating
<<Include>>
Calculate Average
Ratings
<<Include>>
Store Rating
Information
User
Rate Station
<<Include>>
Database


CHARGEMIND SYSTEM
View
Station Reviews
<<Include>>
<<Include>>
Process Review
<<Include>>
Store/Read Data
User
Enter a Comment
Database


GPS
Database
API
CHARGEMIND SYSTEM
Provide User Location
<<include>>
Select Region
<<include>>
Select Desired Time of Use
<<include>>
Select Charge Type (AC/DC)
<<include>>
User
Retrieve Available Stations According to Inputs
<<include>>
List Available Stations for User


CHARGEMIND SYSTEM
Contracted Companies
API
Provide Station Status Info
<<include>>
Update Station Information
<<include>>
Store Station Information
Database


API
CHARGEMIND SYSTEM
Send Regional Grid Carbon Density Data
Send Produced Electricity Source Data
<<include>>
<<include>>
<<include>>
Enter Location, Station Name or Station ID
<<include>>
Calculate Station's Carbon Emission
Compare Station's Carbon Emission
Display Electricity Source
User
<<include>>
<<include>>
Display Emission Percentage
Display Emission Comparison With Nearby Stations

General UI Design

ChargeMind Tech Stack

JavaScript

Python

Selenium

Git

MongoDB

PostgreSQL

Kubernetes

Spring Boot

Java

React Native

Docker

Amazon Web Services

IntelliJ

Flyway

Figma

Kafka

Jira

Postman

Grafana

GraphQL

RabbitMQ

Architectural Decisions

Micro/Macroservice

Test Driven Development (TDD)

Distributed Architecture

Event-Driven Architecture

Micro/Macroservices Architecture – Scalability

Handling Peak Demand:

- Rush hours or holidays
- Multiple EVs may start charging simultaneously at different locations.

Solution:

- Handling a high volume of concurrent requests without delays.

User Growth:

- Support for increasing number of charging sessions, user notifications, and account authentications.

Solution:

- Independent scaling of components

Geographical Expansion:

- Expansion to new regions or countries

Solution:

- Seamless integration without re-architecting the system.

Micro/Macroservices Architecture

Cost efficient due to the horizontal scaling

Flexible, different services for different deployments and customers

Reliable due to the prevention of single point of failure via horizontal scaling

Maintaining data consistency

Complex
Containerization, orchestration, monitoring, and etc.

Resource overhead risk
Especially for early-stage

Operational
Latency

Inter-Service communication
(Coupling)

Event-Driven Architecture

Handling Peak Demand:
- Rush hours or holidays
- Thousands of user interacts with the system

High Availability and Fault Tolerance:
- Events are queued and processed later if a service is down

Solution

Asynchronous Communication

Decoupled Service Architecture:
- Decoupled services need to operate independently
- With low latency
- Without blocking each other

Real-Time User Updates:
- Instant notifications when a charging session starts, without waiting for backend processes to complete
- Ensuring real-time updates

Event-Driven Architecture

Highly scalable and flexible

Dynamic decision making and supports real-time monitoring

Fault tolerant due to the non-blocking operations of decoupled services (loose coupling high cohesion)

System and event design complexity

Data consistency challenges due to eventual consistency reliability

Handling duplicate events and event failures

Race conditions between services

Saga or CQRS patterns

Operations with stale data

Distributed Architecture

Data Handling and Scalability:
- Diverse and growing datasets
- Different regions

Need of High Availability:
- Peak hours
- Thousands of concurrent users accessing services (payments…)

Real-Time Data Handling:
- Need of instant feedback on charging progress, energy consumption, emissions, and so on

Solution: :
- Efficiently partitioning and replicating data across multiple servers

Solution:
- Processing and updating data in real-time
- Dynamic progress tracking

Distributed Architecture

Different flows requires different amount of consistency

Dynamic decision making and supports real-time monitoring

Fault tolerant due to the non-blocking operations of decoupled services (loose coupling high cohesion)

Improved performance with data partitioning

Geographic Redundancy and Disaster Recovery

Trade-Offs in decision making between consistency and availability for different flows

Need of balance between eventual and strong consistency

Higher attack surface / Security

Possible cost or resource overhead

Complexity on design and maintenance

TTD – Test First, Code After

Early bug detection	Increased code volume
Easier maintenance via continuous feedback and better refactoring	Time and effort for test maintenance
Improved code quality, so more reliable product	As the number of external dependencies increased test might be more complex to implement

Test Driven Development Strategy

Unit tests

- Individual component test (e.g., functions, methods) for each class
- Whether work as expected in isolation
- JUnit, Mockito
- For **normal cases** (typical inputs), **edge cases** (extreme or boundary conditions), **error handling** (invalid inputs, exceptions)
- Statement coverage, branch coverage, and path coverages will be
- Fuzz or boundary testing with mock data (mockito)

Integration tests

- Validation of the interaction between different components or service
- Spring Boot Test (Spring-based application components), Postman (API calls), embedded databases like H2 (simulation)
- Interaction Between Components:
  Verify that services interact correctly through APIs or service calls.
- End-to-End Functional Flow:
  Business processes work as intended when all components are integrated.
- Error Handling in Integration

Test Driven Development Strategy

Feature tests (BDD)

- Regular tests of the behavior of key features and business workflows
- Determine whether user expectations are met and system works as intended.
- Selenium, saucelab
- Behavioral Coverage:
  all key user scenarios are covered
- Flow Coverage:
  the workflows tested cover the entire end-to-end process
- Interaction Coverage:
  different features or services interact

Security tests

- Identify vulnerabilities and security flaws
- Data breaches, Authentication issues, Authorization issues, Vulnerabilities in API endpoints
- OWASP ZAP
- Will be applied after first release to the mock customer

Exploratory Tests

- Potential issues or edge cases through ad-hoc testing for fixing bugs early stage
- Features or areas of the application that may not be covered by formal test cases, based on tester intuition and experience

Thank you for listening

Questions