AN INDUSTRIAL ORIENTED MINI PROJECT REPORT

ON

E-CHARGE

(An Electric Vehicle Charging Station Finder App)

Submitted to

JAWAHARLAL NEHRU TECHNOLOGY UNIVERSITY HYDERABAD

(In partial fulfilment of the requirements for the award of bachelor degree)

In

COMPUTER SCIENCE AND ENGINEERING

Submitted by

YELLA NOOKARAJU	**20QM1A05A6**
SHIVAGHONI KIRAN GOUD	**20QM1A05A9**
SAYYED USMAN	**20QM1A0585**
B SANDEEP	**21QM5A0501**

Under the esteemed guidance

Of

DR. S. BHARATH REDDY

Associate Professor

Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Approved by AICTE, New Delhi Affiliated to JNTUH Hyderabad.

Chilkur(V), Moinabad Mandal, R.R Dist. -501504, Ph:9247033008,9000633008. Website: www.kgr.ac.in

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Batch: 2020-2024

Department of Computer Science and Engineering

KG REDDY
College of Engineering
& Technology
AN AUTONOMOUS INSTITUTION

CERTIFICATE

This is to certify that the project report entitled **“E-CHARGE”** (An Electric Vehicle Charging Station Finder App) that is being submitted by Yella Nookaraju (20QM1A05A6), Shivaghoni Kiran Goud (20QM1A05A9), Sayyed Usman (20QM1A0585), B.Sandeep (21QM5A0501) under the guidance of **Dr.S.Bharath Reddy** with fulfilment for the award of the Degree of **Bachelor of Technology in Computer Science and Engineering** to the Jawaharlal Nehru Technological University is a record of bonafide work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any graduation degree.

DR.S.BHARATH REDDY
Associate Professor
INTERNAL GUIDE, CSE Dept.

DR.S.BHARATH REDDY
Associate Professor
Head of Department, CSE Dept.

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

This is to place on our record my appreciation and deep gratitude to the persons without whose support this project would never have been this successful.

We would like to express our profound sense of gratitude to our Director **DR. ROHIT KANDAKATLA** of **KGRCET** for his guidance, encouragement and for all the facilities to complete this project. We have immense pleasure in expressing thanks and deep sense of gratitude for valuable time with us and laying down his valuable suggestions to complete the project successfully on time.

It is with immense pleasure that we would like to express our indebted gratitude to **DR. Y. VIJAYALATA, Principal, K G Reddy College of Engineering & Technology**, for providing a great support and for giving us the opportunity of doing the project.

At the same time, we feel elated to the, **DR. S. BHARATH REDDY, Associate Professor, HOD of Department of CSE, K.G. Reddy College of Engineering & Technology**, for inspiring us all the way and for arranging all the facilities and resources needed for our project.

We would like to take this opportunity to thank our internal guide **DR. S. BHARATH REDDY, Associate Professor, Department of CSE, K.G. Reddy College of Engineering & Technology,** who has guided us a lot and encouraged us in every step of the project work, his valuable moral support and guidance throughout the project helped us to a greater extent.
We would like to take this opportunity to specially thank our Project Coordinator, **Mr. SURENDRA TRIPATI, Assistant Professor, Department of CSE, K.G. Reddy College of Engineering & Technology,** who guided us in our project.

Finally, we express our sincere gratitude to all the members of the faculty of the Department of Computer Science and Engineering, our friends and our families who contributed their valuable advice and helped us to complete the project successfully.

YELLA NOOKARAJU	**20QM1A05A6**
SHIVAGHONI KIRAN GOUD	**20QM1A05A9**
SAYYED USMAN	**20QM1A0585**
B SANDEEP	**21QM5A0501**

DECLARATION

This is to certify that the major project titled **"E-CHARGE (Electrical Vehicle Charging Station Finder App)"** is a bonafide work done by us in fulfillment of the requirements for the award of the degree **Bachelor of Technology** in **Computer Science and Engineering submitted, to the Department of C.S.E, KG Reddy College of Engineering and Technology**, Chilkur, Moinabad, Hyderabad.

We also declare that this project is a result of our own effort and has not been copied or intimated from any source. Citations from any websites are mentioned in the bibliography. This work was not submitted earlier at any other university for the award of any degree.

YELLA NOOKARAJU	**20QM1A05A6**
SHIVAGHONI KIRAN GOUD	**20QM1A05A9**
SAYYED USMAN	**20QM1A0585**
B SANDEEP	**21QM5A0501**

ABSTRACT

Electric vehicles are becoming increasingly popular, but finding charging stations can still be a challenge for many drivers. This creates a need for a reliable and user-friendly EV charging station finder app that can provide real-time information on the location and availability of charging stations based upon their type of electric vehicle. As more and more people switch to electric vehicles (EVs), there is a growing need for a reliable and user-friendly app that can help EV drivers find charging stations. While there are already some apps available for this purpose, they often lack real-time information, are limited in their coverage, or are difficult to use. This can lead to frustration and inconvenience for EV drivers, who may struggle to plan their routes or find charging stations in unfamiliar areas. Overall, the domain of the EV charging station finder app is dynamic and evolving, with new charging stations and technologies being introduced regularly. This app will be flexible and adaptable to meet the changing needs of the EV charging ecosystem.

List of Figures Page No.

INDEX

1. INTRODUCTION

1.1 Basic Introduction

An Electric Vehicle (EV) Charging Station Finder App is a software application designed to assist electric vehicle owners in locating charging stations for their vehicles. With the growing popularity of electric vehicles as a more sustainable mode of transportation, these apps have become essential tools for EV owners to navigate and plan their journeys effectively.

The primary purpose of an EV charging station finder app is to provide real-time information about the nearest charging stations, their availability, types of charge stations, and other relevant details. This empowers EV drivers to make informed decisions on where and when to charge their vehicles, ensuring they have sufficient power for their trips.

The Electric Vehicle (EV) Charging Station Finder App is a revolutionary tool designed to streamline the adoption of electric vehicles in an increasingly eco-conscious world. This app addresses a fundamental concern for EV owners - locating charging stations quickly and efficiently. By offering real-time information, seamless navigation, and user interaction, the app empowers users to make informed decisions, enhance convenience, and contribute to a sustainable future.

The Electric Vehicle (EV) Charging Station Finder App represents a breakthrough solution in the realm of sustainable transportation. As the world embraces cleaner energy alternatives, the app emerges as a beacon, addressing a critical hurdle for EV adoption. Navigating the landscape of charging stations is a paramount concern for electric vehicle owners, and this app meets that challenge head-on. By seamlessly connecting users with nearby charging points and providing essential station information, the app empowers individuals to embrace electric mobility with confidence, contributing to a greener planet and a brighter automotive future.

1.2 Purpose of the EV Charging Station Finder App

The purpose of the Electric Vehicle Charging Station Finder App is to simplify the experience of owning and driving electric vehicles. By providing users with a convenient and efficient way to locate nearby charging stations, the app aims to alleviate the concerns of range anxiety and enhance the accessibility of electric mobility. With real-time information, intuitive navigation, and user engagement, the app supports the adoption of sustainable transportation options and contributes to a cleaner and greener future.

1.3 Scope of the EV Charging Station Finder App

The scope of an EV charging station finder app encompasses a wide range of functionalities and features that cater to the needs of electric vehicle owners and the overall sustainable transportation ecosystem. The EV Charging Station Finder App aims to provide a user-friendly and comprehensive solution for electric vehicle owners to locate, access, and engage with charging stations. The app's scope includes:

1. **Charging Station Discovery:**
- Real-time location-based search for nearby charging stations.
- Display of charging station availability status.
2. **Navigation and Directions:**
- Integration of navigation services to guide users to selected charging stations.
3. **User Interaction:**
- User account creation and profile management.
- Submission of user reviews and ratings.
4. **User Experience:**
- Intuitive and user-friendly interface for seamless interaction.
- Personalized user profiles to track favourite stations.
5. **Platform Compatibility:**
- Development for mobile platforms (iOS and Android) to ensure broad accessibility.

The app's primary focus is on delivering a convenient, reliable, and engaging solution that supports electric vehicle users in their journey towards sustainable transportation.

1.4 Goals of EV Charging Station Finder App

The goals of an EV charging station finder app are centered around providing electric vehicle (EV) owners with a convenient, efficient, and seamless experience when it comes to locating and utilizing charging infrastructure. Here are the key goals of such an app:

1. **Enhance Convenience for EV Drivers:**
   - The app aims to eliminate "range anxiety" by enabling EV drivers to easily find charging stations along their routes, promoting confidence in EV usage.
2. **Facilitate Real-Time Access to Information:**
   - Providing up-to-the-minute information about charging station
3. **Provide Personalization:**
   - User profiles and preferences allow for a tailored experience, including saving favorite stations
4. **Future Expansion:**
   - To continuously improve and expand the app's features and functionalities.

In summary, an EV charging station finder app aims to create a symbiotic relationship between EV owners and the charging infrastructure, fostering a seamless and sustainable transportation ecosystem.

These goals collectively contribute to the app's overarching objective: to make electric vehicle ownership more convenient, enjoyable, and environmentally friendly.

1.5 Features of Our Project

Here are some potential features that your Electric Vehicle Charging Station Finder App could include:

- **Location-Based Search:** Users can search for charging stations based on their current location or specify a different area.
- **Real-Time Availability:** Display real-time availability status of charging stations, indicating whether they are occupied or vacant (Future Scope).
- **Station Details:** Provide comprehensive information about each charging station.
- **Navigation Integration:** Seamlessly integrate navigation services to guide users to their chosen charging station.
- **User Accounts:** Enable users to create accounts, personalize their profiles, and manage their favourite charging stations.
- **Filtering Options:** Offer filters for users to narrow down charging station search results based on criteria such as charging speed and connector type (Future Scope).
- **User-Friendly Interface:** Design an intuitive and user-friendly interface that simplifies the app's navigation and use.
- **Notifications:** Send notifications to users about station availability, special offers, or important updates.
- **Payment Integration:** Allow users to pay for charging sessions directly through the app, if applicable. (Future Scope)
- Support for Multiple Platforms: Develop the app for both iOS and Android to reach a broader audience.
- **Favourites:** Allow users to track their favourite stations.

2. SYSTEM ANALYSIS

2.1 Existing System

Existing EV charging station finder systems offer a range of functionalities to assist electric vehicle owners in locating charging stations conveniently.

PlugShare:

- One of the most popular apps, PlugShare, provides real-time maps with locations of various charging stations, including both public and private ones.
- It offers a variety of features such as user reviews, station ratings, photos, and information about charging station types and connectors.
- Users can plan trips, share their own stations, and even coordinate charging with others.

2.2 Proposed System

The proposed EV charging station finder system aims to provide an advanced and user-centric solution for electric vehicle (EV) owners to effortlessly locate, access, push notifications, payment mode (future scope), status of the charge station and utilize charging infrastructure. This system leverages cutting-edge technology and innovative features to enhance the overall EV charging experience.

2.3 Overall Description

The Electric Vehicle (EV) Charging Station Finder App is a revolutionary tool designed to address the specific needs of electric vehicle owners in locating and accessing charging infrastructure conveniently. With the increasing adoption of electric vehicles as a sustainable mode of transportation, this app emerges as an essential solution.

In summary, the EV Charging Station Finder App serves as a ground breaking solution that bridges the gap between electric vehicle ownership and convenient charging access. By leveraging technology and user-centric features, it empowers electric vehicle owners to embrace cleaner, more sustainable transportation while fostering a connected community of environmentally conscious travellers.

2.4 Feasibility Study

The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running systems. All systems are feasible if they have unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation.

1. Technical Feasibility:

- **App Development Platform:** Determine the suitable development platform (e.g., native or cross-platform) based on your team's skills and the target user base.
- **Integration with APIs:** Evaluate the availability and compatibility of APIs for map integration, navigation, and real-time charging station data.

2. Financial Feasibility:

- **Cost Estimation:** Calculate the costs associated with app development, including design, programming, testing, and maintenance. Consider factors like hiring developers, licensing fees, and server costs.
- **Revenue Model:** Define how the app will generate revenue, whether through advertisements, in-app purchases, subscription plans, or partnerships with charging station operators.

3. Operational Feasibility:

- **Charging Station Data:** Assess the feasibility of obtaining real-time and accurate charging station data. Consider partnerships with data providers or creating a database through user contributions.
- **Navigation Services:** Evaluate the feasibility of integrating navigation services for guiding users to charging stations.

2.5 SDLC Model

The Software Development Life Cycle (SDLC) model you choose for developing the EV Charging Station Finder App will play a crucial role in defining the project's structure, stages, and how it progresses from concept to deployment.

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously.

Agile SDLC Model for EV Charging Station Finder App:

1. **Planning and Requirement Analysis:**
- Define the project scope, objectives, and features.
- Identify user requirements, including map integration, real-time data, user profiles, payment options, etc.

2. **Iteration 1 - Sprint 1: Design and Prototyping:**
- Develop wireframes and prototypes of the app's key screens.
- Identify the technology stack, tools, and frameworks required for development.

3. **Iteration 2 - Sprint 2: Core Functionality Development:**
- Develop the foundational features of the app, including user registration, login, and map integration.
- Implement the backend APIs for user authentication and basic station information.

4. **Iteration 3 - Sprint 3: Map Integration and Real-Time Data:**
- Integrate Google Maps or other mapping services into the app.
- Develop the logic to fetch and display real-time charging station data, including availability and types of connectors.

5. **Iteration 4 - Sprint 4: User Profiles and Preferences:**
- Implement user profile creation, editing, and preferences settings.
- Enable users to save favorite stations and set notification preferences.

6. **Iteration 5 - Sprint 5: Payment Integration and Reviews (Future Scope):**
- Integrate payment gateways for in-app charging station payments.
- Develop the functionality for users to leave reviews and ratings for stations.

7. **Iteration 6 - Sprint 6: Notifications and Alerts:**

- Develop the notification system to alert users about available stations and charging session updates.

8. **Iteration 7 - Sprint 7: Testing and Quality Assurance:**

- Conduct comprehensive testing, including unit testing, integration testing, and user acceptance testing.
- Address any issues, bugs, or performance concerns identified during testing.

9. **Iteration 8 - Sprint 8: Deployment and User Training:**

- Deploy the app to production servers or app stores.
- Provide user documentation and guides for using the app's features.

10. **Iteration 10 - Sprint 10: Post-Deployment Monitoring and Updates:**

- Monitor the app's performance, user feedback, and any issues post-deployment.
- Release regular updates to address bugs, introduce new features, and enhance user experience.

The Agile SDLC model allows for flexibility, iterative development, and the ability to respond to changing requirements and user feedback. This is particularly beneficial for a project like the EV Charging Station Finder App, where continuous improvement and user-centric design are essential for success.

3. SYSTEM REQUIREMENT SPECIFICATIONS

3.1 Software Requirements

- Tools - Chrome or Microsoft Edge
- Platform – Mobiroller
- Modules – Map, Standard Content, FAQ, Settings

3.2 Hardware Requirements

- Processor – IntelCoreI3 or Above
- RAM – 4 GB (min)
- Hard-Disk – 1 TB (min)
- GPU for Good Cost/Performance

4. SYSTEM DESIGN

4.1 Design Overview

Electric Vehicle Charging Station Finder application that we developed is using MobiRoller Website. MobiRoller is a mobile app builder that allows you to create custom mobile apps without extensive coding knowledge.

1. **Define Your App's Purpose:**

- Clearly outline the primary purpose and goals of your app. Understand what problems it will solve or what value it will provide to users.

2. **Design the User Interface (UI):**

- Use MobiRoller's drag-and-drop interface to design the app's user interface.

  <u>Consider the following elements:</u>

- **Screens:** Create individual screens for different app functions (e.g., home screen, profile screen, settings).
- **Navigation:** Design the app's navigation menu or tabs for easy user access.
- **Layout:** Choose layouts for your screens, including the arrangement of buttons, images, and text.
- **Color Scheme:** Select a color palette that aligns with your brand and app's theme.
- **Typography:** Pick fonts and text styles for a consistent and visually appealing UI.
- **Images and Icons:** Upload or choose images and icons that enhance the app's aesthetics.

4.2 System Architecture

The system architecture of MobiRoller apps is abstracted from users, as they do not have direct access to the underlying technical infrastructure. Instead, users work within MobiRoller's user-friendly interface to design and configure their apps. Below, I'll provide a simplified overview of the architectural components involved in the MobiRoller platform:

1. **User Interface (UI):**

- Users interact with MobiRoller through a web-based interface where they design and configure their mobile apps.
- The UI provides tools for creating and customizing app screens, layouts, navigation menus, and features.

2. **Frontend:**

- The frontend of MobiRoller is responsible for rendering the user interface and providing an interactive experience for users.
- It uses web technologies (HTML, CSS, JavaScript) to allow users to design their apps using a drag-and-drop interface.

3. **Backend Services:**

- MobiRoller's backend services handle various functionalities required for app creation and management. These include:
- **App Configuration:** Storing and managing app configuration data, such as screen layouts, content, and feature settings.
- **User Management:** Managing user accounts, permissions, and access controls.
- **Data Storage:** Storing user-generated content, app data, and media files.
- **App Templates:** Providing pre-designed app templates that users can customize.
- **Publishing:** Handling the process of publishing apps to app stores.

4. **Databases:**

- MobiRoller uses databases to store user data, app configurations, and app content.
- Databases ensure data persistence and availability across app sessions.

5. **Cloud Infrastructure (Optional):**

- MobiRoller may utilize cloud-based services to host and scale its platform, ensuring high availability and scalability.

6. **Content Delivery Network (CDN):**

- To optimize the delivery of media files (images, videos) and app content, a CDN may be employed to distribute content to users efficiently.

7. **Third-Party Services:**

- MobiRoller integrates with third-party services for various purposes, such as analytics, payment processing, and user authentication.

8. **Analytics and Monitoring:**

- MobiRoller may use analytics and monitoring tools to gather insights into user behavior, app performance, and usage metrics.

9. **Security:**

- Security measures are implemented to protect user data, prevent unauthorized access, and ensure the integrity of the platform.

10. App Stores: MobiRoller facilitates the process of publishing apps to major app stores like the Google Play Store and Apple App Store.

It's important to note that the exact technical details of MobiRoller's architecture may not be publicly disclosed, as it's a proprietary platform. However, the above overview gives a general idea of the components and functionalities involved in the system architecture of MobiRoller apps. Users primarily interact with the frontend and the user interface, while the backend handles the complexities of app creation and management.

4.3 Modules Description

MobiRoller, as a mobile app builder platform, provides various modules and features that users can utilize to create and customize their mobile apps. These modules are designed to simplify app development and cater to different app requirements.

1. **Screen Builder:**

- Allows users to create app screens by dragging and dropping elements like buttons, images, text, and forms.
- Users can customize the layout, design, and content of each screen.

2. Navigation Menu:

- Provides options to create navigation menus, tabs, or sidebars for easy app navigation.
- Users can define menu items and link them to specific screens or sections.

3. Standard Content Module:

- This is ideal for screens consisting of a picture and a brief description. You may use this screen to give information about basically anything.

4. Map Module:

- You can make your office, branch, event or any other place displayed on the map and let the users find their location.

5. **Settings Module:**

- Your users can change basic app settings such as instant notification, message tone, font size and language.

6. **FAQ Module:**

- You can create frequently asked questions screen and provide informative content within this screen to users.

4.4 UML Design

The UML stands for Unified modelling language, is a standardized general-purpose visual modelling language in the field of Software Engineering. It is used for specifying, visualizing, constructing, and documenting the primary artefacts of the software system. It helps in designing and characterizing, especially those software systems that incorporate the concept of Object orientation. It describes the working of both the software and hardware systems.

Goals of UML

- Since it is a general-purpose modeling language, it can be utilized by all the modelers.
- UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.
- The UML diagrams are made for business users, developers, ordinary people, or anyone who's looking forward to understanding the system, such that the system can be software or non-software.
- Thus it can be concluded that the UML is a simple modeling approach that is used to model all the practical system

UML Building Block

UML is composed of three main building blocks, i.e., things, relationships, anddiagrams. Building blocks generate one complete UML model diagram by rotating aroundseveral different blocks. It plays an essential role in developing UML diagrams. The basicUML building blocks are enlisted below:

1. Things
2. Relationships
3. Diagrams

Things and its types

- **Structural Things**
- **Behaviour Things**
- **Grouping Things**

- **Annotation Things**

Structural Thing:

Nouns that depict the static behaviour of a model are termed as structural things. They display the physical and conceptual components. They include class, object, interface, node, collaboration, component, and a use case.

Class: A Class is a set of identical things that outlines the functionality and properties of an object.

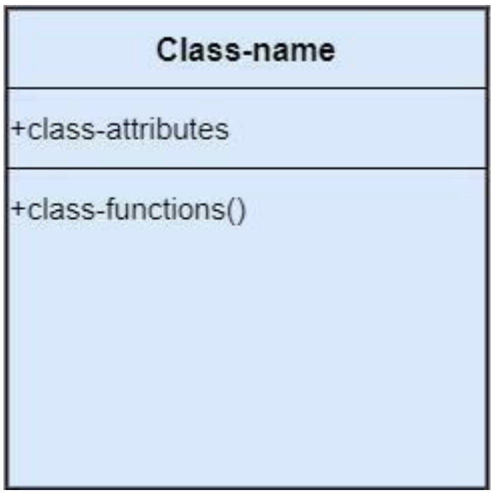


Figure: 4.4.1 Notation of Class

Object: An individual that describes the behaviour and the functions of a system. The notation of the object is similar to that of the class; the only difference is that the object name is always underlined and its notation is given below.

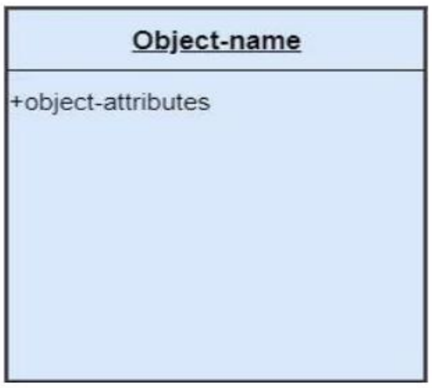


Figure: 4.4.2 Notation of Object

Interface: A set of operations that describes the functionality of a class, which is implemented whenever an interface is implemented.

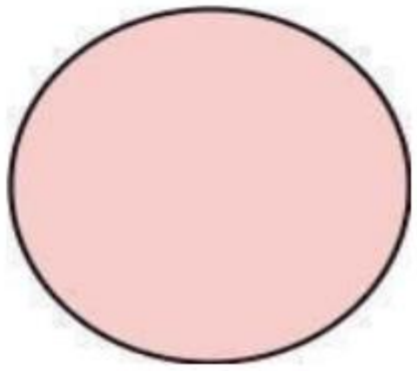


Figure: 4.4.3 Notation of Interface

Use case: Use case is the core concept of object-oriented modeling. It portrays a set of actions executed by a system to achieve the goal.

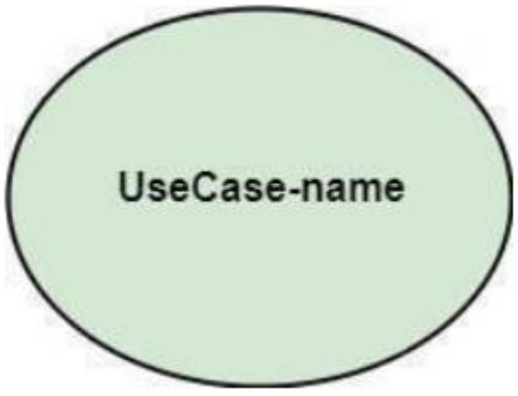


Figure: 4.4.4 Notation of Use Case

Actor: It comes under the use case diagrams. It is an object that interacts with the system, for example, a user.

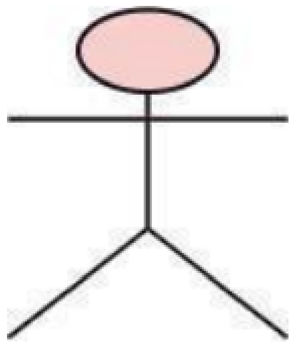


Figure: 4.4.5 Notation of Actor

Behavioural Things

They are the verbs that encompass the dynamic parts of a model. It depicts the behaviour of a system. They involve state machine, activity diagram, interaction diagram, grouping things, annotation things.

State Machine: It defines a sequence of states that an entity goes through in the software development life cycle. It keeps a record of several distinct states of a system component.

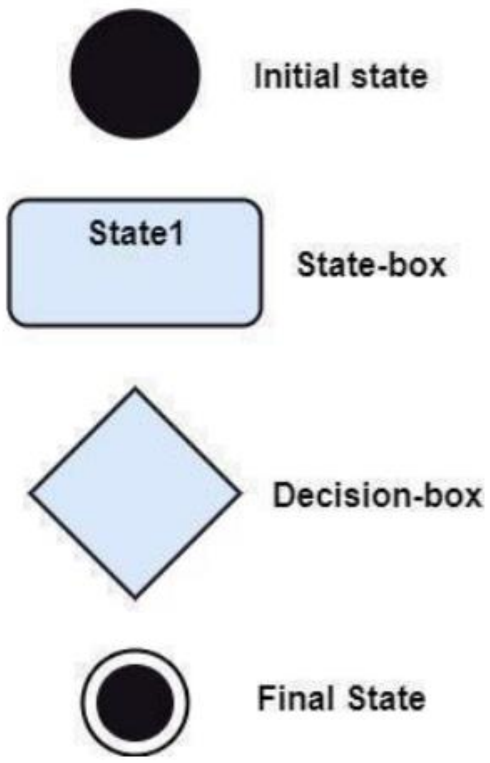


Figure: 4.4.6 State Machine

Activity Diagram: It portrays all the activities accomplished by different entities of a system. It is represented the same as that of a state machine diagram. It consists of an initial state, final state, a decision box, and an action notation.

Grouping Things

It is a method that together binds the elements of the UML model. In UML, the package is the only thing, which is used for grouping.

Package: Package is the only thing that is available for grouping behavioral and structural things.

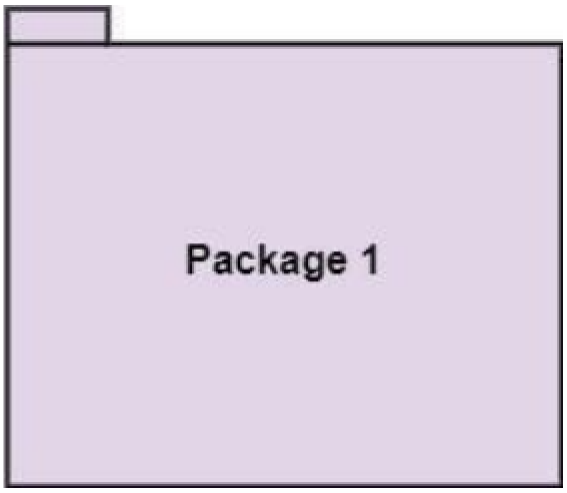


Figure: 4.4.7 Notation of Package

Annotation Things

It is a mechanism that captures the remarks, descriptions, and comments of UML model elements. In UML, a note is the only Annotational thing.

Note: It is used to attach the constraints, comments, and rules to the elements of the model. It's Kind of yellow sticky.

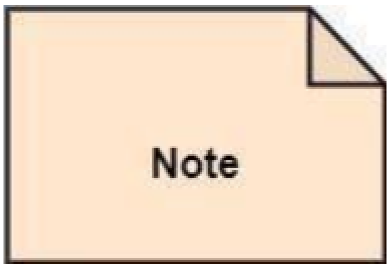


Figure: 4.4.8 Notation of Note

Relationships

It illustrates the meaningful connections between things. It shows the association between entities and defines the functionality of an application. There are four types of relationships given below:

- **Dependency:** Dependency is a kind of relationship in which a change in target element affects the source element, or simply we can say the source element is dependent on the target element. It is one of the most important notations in UML. It depicts the dependency from one entity to another. It is denoted by a dotted line followed by an arrow at one side as shown below

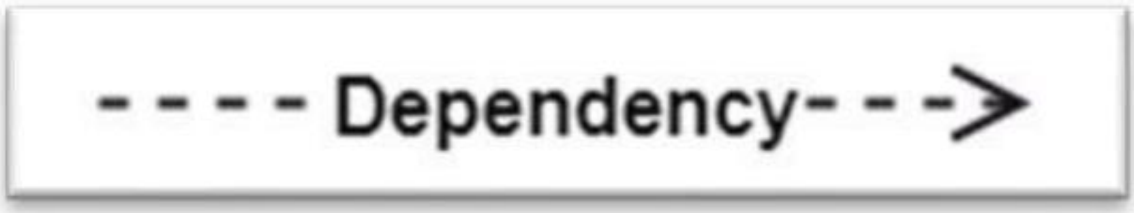


Figure: 4.4.9 Dependency

- **Association:** A set of links that associates the entities to the UML model. It tells how many elements are actually taking part in forming that relationship. It is denoted by a dotted line with arrowheads on both sides to describe the relationship with the element on both sides.

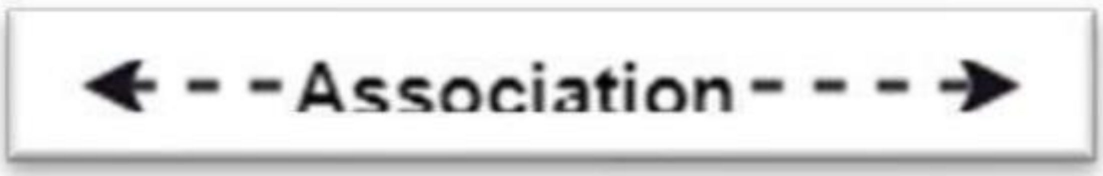


Figure: 4.4.10 Association

- **Generalization:** It portrays the relationship between a general thing (a parent class or superclass) and a specific kind of that thing (a child class or subclass). It is used to describe the concept of inheritance. It is denoted by a straight line followed by an empty arrowhead at one side.

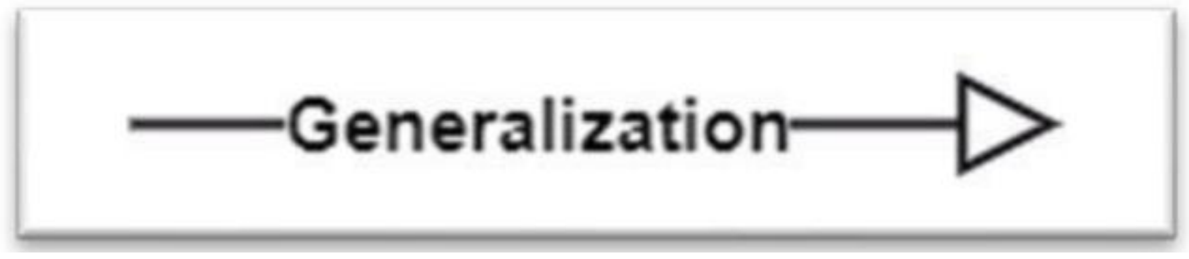


Figure: 4.4.11 Generalization

- **Realization:** It is a semantic kind of relationship between two things, where one defines the behaviour to be carried out, and the other one implements the mentioned behaviour. It has interfaces. It is denoted by a dotted line with an empty arrowhead at one side.

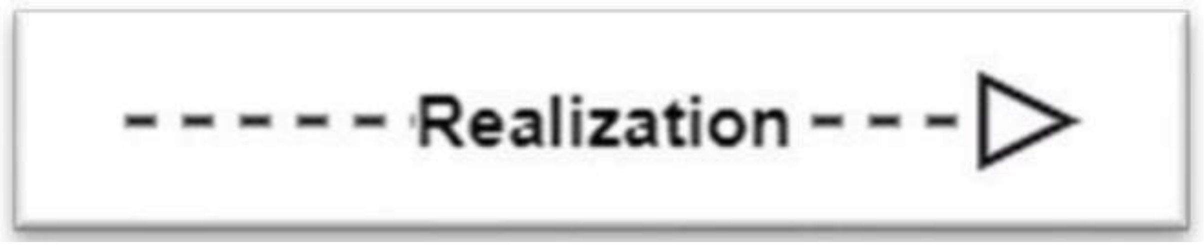


Figure: 4.4.12 Realization

DIAGRAMS

The diagrams are the graphical implementation of the models that incorporate symbols and text. Each symbol has a different meaning in the context of the UML diagram.

There are thirteen different types of UML diagrams that are available in UML 2.0, such that each diagram has its own set of symbols. And each diagram manifests a different dimension, perspective, and view of the system.

UML diagrams are classified into three categories that are given below:

1. Structural Diagram

2. Behavioural Diagram

3. Interaction Diagram.

CLASS DIAGRAM

The class diagram is the main building block of object oriented modeling. It is used for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. A class with three sections, in the diagram, classes is represented with boxes which contain three parts:

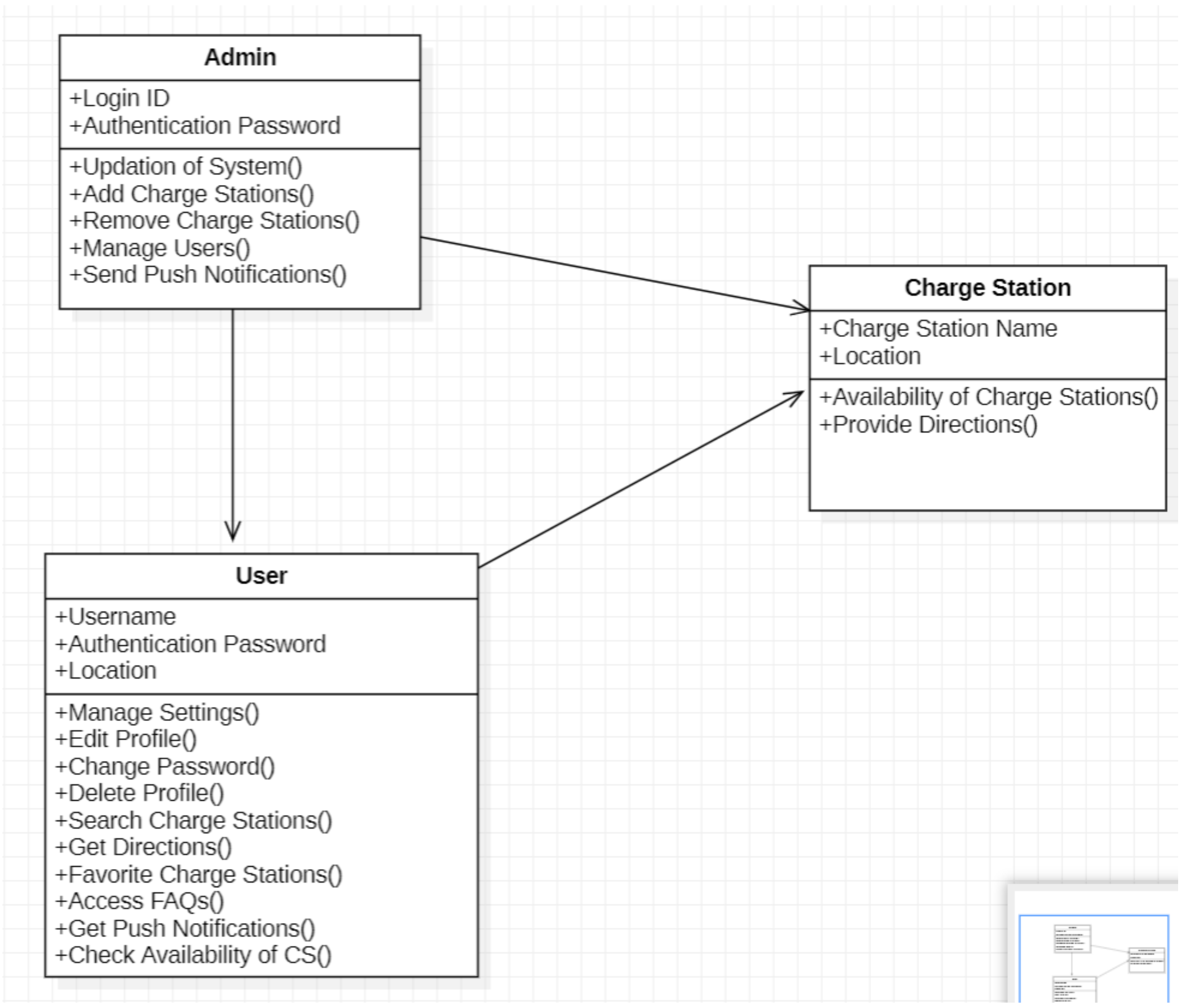


Figure: 4.4.13 Class Diagram

USE-CASE DIAGRAM

A use case diagram at its simplest is a representation of a user's interaction with the system and depicts the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

Overall Use-case Diagram

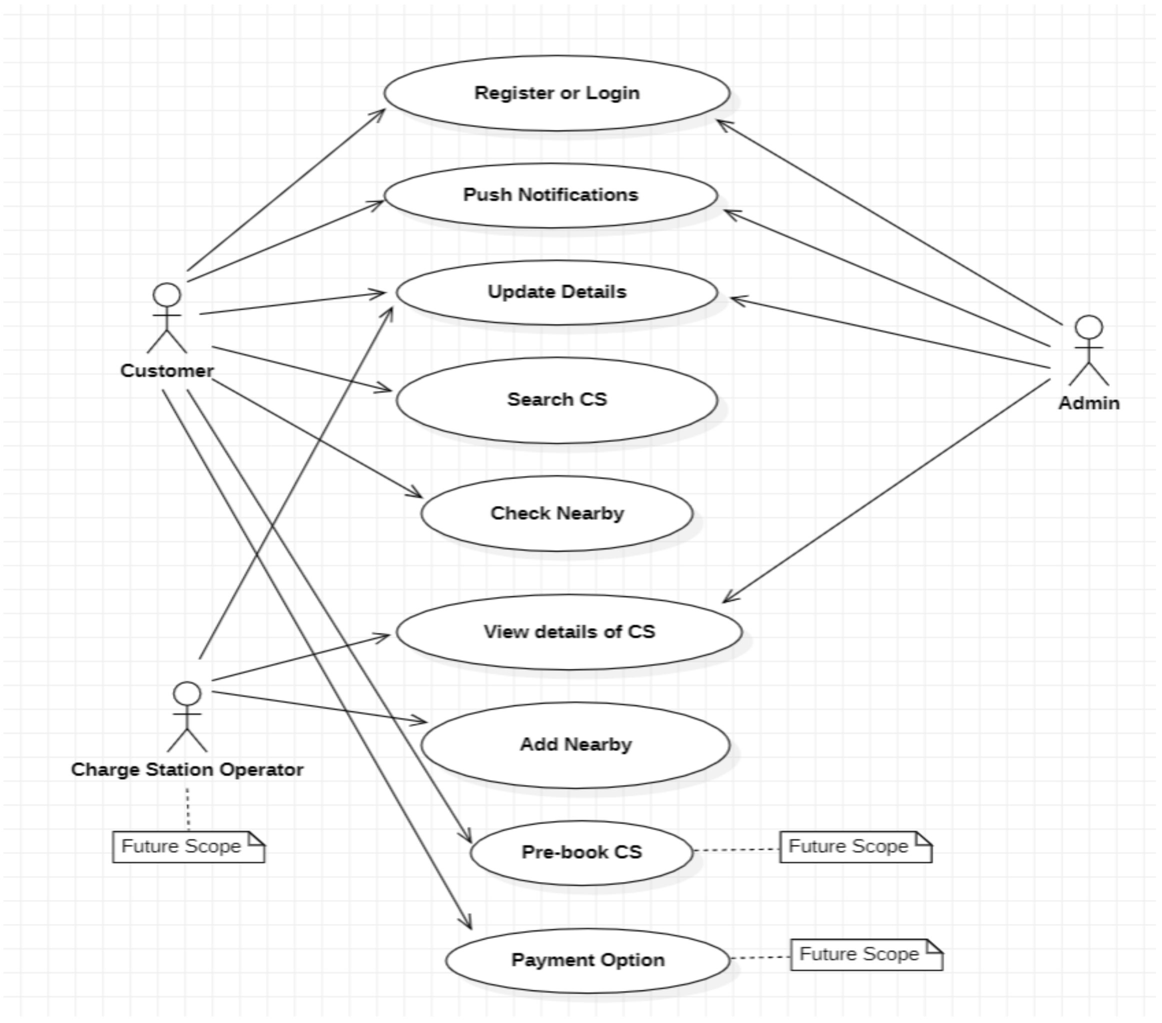


Figure: 4.4.14 Use Case Diagram

ER Diagram

Entity-Relationship (ER) diagrams are not typically a part of the Unified Modeling Language (UML), but they are a distinct type of diagram used in database design and modeling. ER diagrams are a visual representation of the entities (objects or concepts) in a database and the relationships between them. These diagrams are especially useful for designing relational databases.

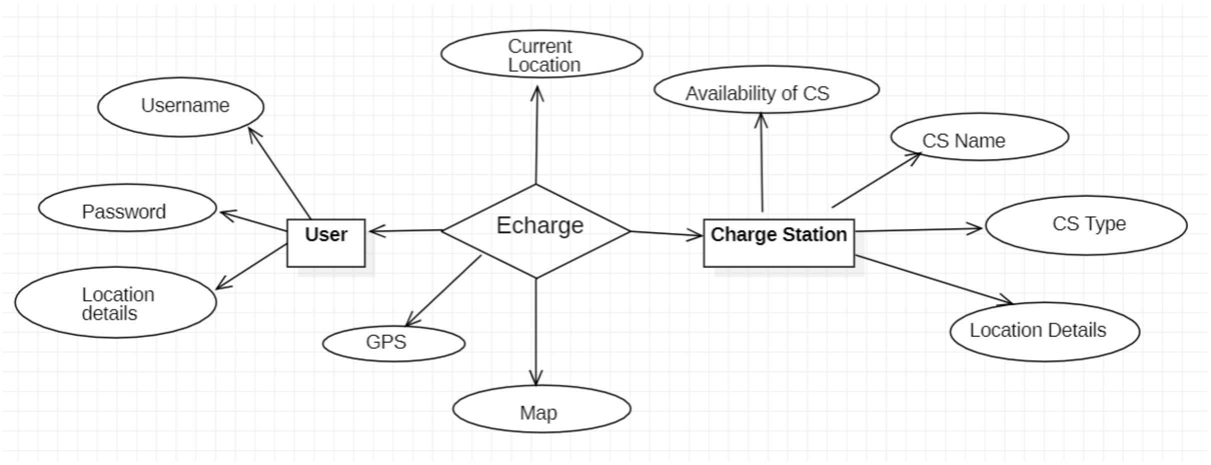


Figure: 4.4.15 ER Diagram

4.5 Deployment Diagram

A deployment diagram is one of the structural diagrams in the Unified Modeling Language (UML), used to model and visualize the physical deployment of software components and hardware nodes within a system or application. Deployment diagrams provide a clear and detailed representation of how software artifacts are distributed across various hardware nodes or devices in a real-world environment, such as servers, computers, routers, and more.

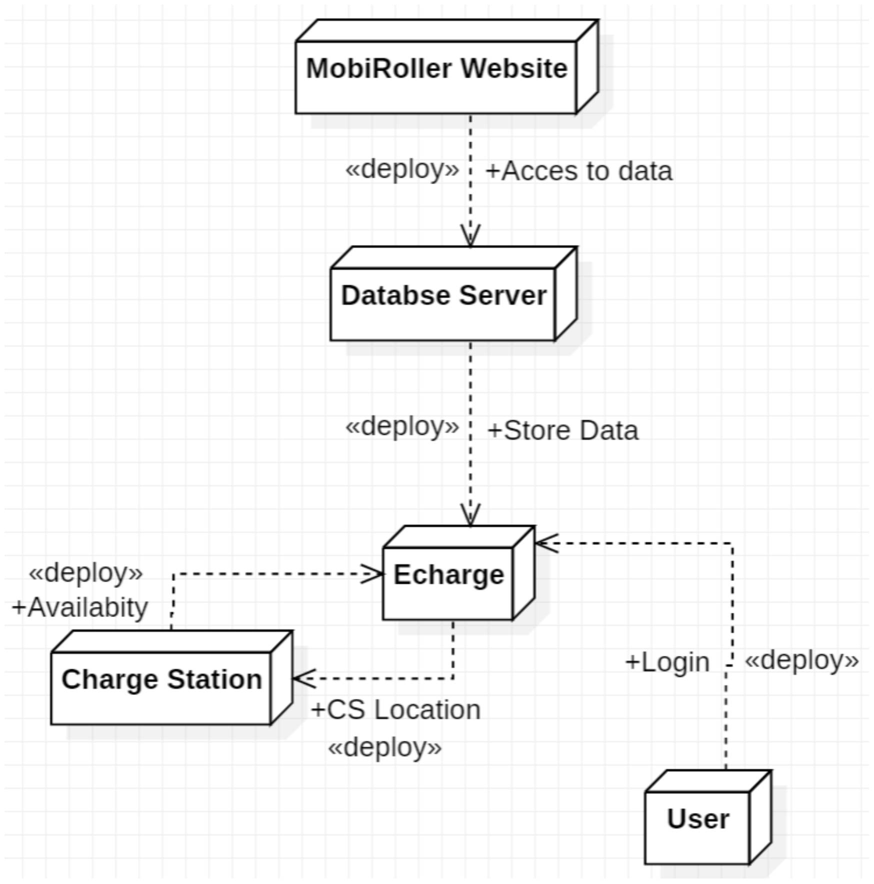


Figure: 4.5.1 Deployment Diagram

5. SYSTEM IMPLEMENTATION

5.1 About MobiRoller

Mobiroller is a mobile app builder platform that enables individuals and businesses to create mobile applications without extensive coding knowledge. While it's primarily focused on app development rather than website creation, you can still use Mobiroller to design a website using its user-friendly tools and templates.

Platform: Mobiroller

- **Overview:** Mobiroller is a versatile mobile app development platform that simplifies the process of creating and managing mobile applications. While its primary purpose is app development, Mobiroller offers website building capabilities, making it an attractive choice for users looking to establish a web presence without deep technical expertise.

- **Key Features:**

a) **No-Code Development:** Mobiroller's user-friendly interface allows users to create mobile apps and websites without the need for extensive coding skills. This feature appeals to both beginners and experienced developers.

b) **Template Library:** The platform offers a library of pre-designed templates for both apps and websites. Users can choose from a variety of templates to suit their specific needs, ranging from e-commerce to social networking.

c) **Drag-and-Drop Editor:** Mobiroller includes a drag-and-drop editor that simplifies the design and layout process. Users can add elements, customize the appearance, and arrange content with ease.

d) **Customization:** Users have the flexibility to customize the design, colors, fonts, and branding elements to align with their brand identity.

e) **Integration:** Mobiroller allows integration with various third-party services and plugins, enhancing the functionality of both apps and websites. This includes features like push notifications and payment gateways.

f) **App Store Deployment:** For mobile apps, Mobiroller provides the option to publish apps to popular app stores, such as Apple App Store and Google Play Store.

g) **Responsive Design:** Websites created using Mobiroller are designed to be responsive, ensuring a seamless user experience across different devices and screen sizes.

- **Use Cases:**

a) **Mobile App Development:** Mobiroller is ideal for individuals and businesses looking to create mobile applications for various purposes, including e-commerce, social networking, content delivery, and more.

b) **Website Creation:** While Mobiroller's primary focus is on app development, it offers the ability to create simple websites, making it suitable for small businesses, portfolios, or personal websites.

- **Technical Documentation:**

a) **APIs and Integration:** If you plan to integrate your Mobiroller app or website with external services or databases, you'll need to consult the platform's API documentation to understand the integration process.

b) **Customization and Styling:** Technical documentation may include details on how to customize templates, design elements, and style sheets to achieve a unique look and feel for your app or website.

c) **Deployment Guides:** If you're building a mobile app, you may find deployment guides for publishing your app on app stores.

This brief overview can serve as an introduction to Mobiroller for your technical document. Depending on your specific use case and requirements, you can delve deeper into technical details, tutorials, and documentation provided by Mobiroller to create a more comprehensive document.

5.2 Implementation of the Application

Implementing an EV Charging Station Finder App using MobiRoller involves several steps, from designing the app's interface to configuring its functionality. Here's a brief overview of how you can implement such an app:

1. Design Your App:

Use MobiRoller's intuitive drag-and-drop interface to design your app's user interface (UI). Consider the key features you want to include, such as a map view for finding charging stations and screens for user profiles and reviews.

2. Navigation and Layout:

Create a navigation structure that allows users to easily access different sections of the app, such as the map, user profile, and settings. Customize the layout and design elements to match the app's theme.

3. Map Integration:

Integrate a mapping service, such as Google Maps, to provide location-based features. This allows users to view their current location, search for nearby charging stations, and get directions.

4. Database Setup:

Configure a database to store charging station data. Define data fields for station names, addresses, charging rates, connector types, and availability.

5. User Authentication:

Implement user authentication features to allow users to create accounts, log in, and access personalized features like saving favorite charging stations.

6. Charging Station Listings:

Create a screen that lists available charging stations based on the user's current location or search criteria. Display relevant information about each station, including real-time availability.

7. Reviews and Ratings:

Enable users to leave reviews and ratings for charging stations. Implement a screen where users can read and write reviews, and display average ratings for each station.

8. Route Planning:

Develop a feature that helps users plan routes to charging stations, taking into account factors like their current battery charge, charging station availability, and estimated charging times.

9. Notifications:

Configure push notifications to inform users about available charging stations, charging progress, or when it's time to unplug their vehicles.

10. Payment Integration (Optional):

If you plan to offer paid services, integrate a secure payment gateway to allow users to pay for charging sessions directly within the app.

11. Testing:

Thoroughly test your app's functionality and usability on various devices. Ensure that the map integration, search, and route planning features work seamlessly.

12. Publish Your App:

Once you're satisfied with your app, use MobiRoller's publishing features to make it available on app stores, such as the Google Play Store for Android or the Apple App Store for iOS.

13. User Engagement and Marketing:

Promote your app to potential users through marketing efforts, social media, and relevant forums or communities. Encourage user engagement and collect feedback for future improvements.

14. Maintenance and Updates:

Continuously monitor your app's performance and user feedback. Release regular updates to fix bugs, add new features, and stay current with the latest mobile OS versions.

15. Analytics and Optimization:

Implement analytics tools to track user behaviour and app performance. Use this data to make informed decisions and optimize your app for better user experiences.

Implementing an EV Charging Station Finder App in MobiRoller allows you to create a user-friendly and functional app without extensive coding. By following these steps and staying attentive to user needs, you can develop a valuable tool for electric vehicle owners and enthusiasts.

5.2.1 Required Modules for Implementation

Implementing an EV Charging Station Finder App using MobiRoller involves creating and configuring various modules to provide a seamless user experience. Here's a brief overview of each module that the app contains and how to implement them:

1. **User Registration and Authentication Module:**

- This module allows users to create accounts and log in securely.
- Implementation involves configuring user registration forms, setting up authentication methods, and managing user profiles.

2. **User Profile Module:**

- Users can customize their profiles, including adding profile pictures and updating personal information.
- Implement this module by creating user profile screens and enabling users to edit their details.

3. **Charging Station Listings Module:**

- This module displays a list of charging stations based on user preferences or location.
- Implement it by integrating with charging station databases or APIs, configuring search filters, and designing the station listing screens.

4. **Station Details Module:**

- Users can view detailed information about each charging station, including location, charging rates, reviews, and available connectors.

- Implement this module by creating a screen to display station details and integrating real-time data.

5. User Reviews and Ratings Module:

- This module allows users to leave reviews and ratings for charging stations.
- Implement it by creating forms for user reviews, displaying ratings, and enabling users to read and write reviews.

6. Map Integration Module:

- This module integrates mapping services to display charging station locations on a map.
- Implement it by configuring map views, adding markers for charging stations, and enabling location-based features.

7. Route Planning Module:

- Users can plan routes to charging stations, considering factors like current battery charge and station availability.
- Implement this module by integrating routing algorithms and providing turn-by-turn directions.

8. User Preferences Module:

- Users can set preferences, such as favorite charging stations or notification settings.
- Implement this module by creating screens for user preferences and storing user settings in the database.

9. Push Notifications Module:

- This module sends push notifications to users, informing them about available charging stations or updates.
- Implement it by configuring push notification services and defining notification triggers.

10. Payment Integration Module (Optional):

- If you offer paid services, integrate a payment gateway for in-app transactions.

- Implement this module by integrating secure payment processing and creating screens for payment.

11. Analytics and Reporting Module:

- This module collects data on user behavior, app performance, and usage metrics.
- Implement analytics tools to gather insights into how users interact with the app.

12. Settings and Preferences Module:

- Users can adjust app settings, such as language, units of measurement, and notifications.
- Implement this module by creating screens for app settings and allowing users to customize their experience.

13. Feedback and Support Module:

- Users can provide feedback, report issues, or seek assistance.
- Implement this module by creating feedback forms, integrating chat or email support, and offering user assistance.

14. Legal Compliance Module:

- Ensure that the app complies with legal requirements, including privacy policies and terms of service.
- Implement this module by adding legal agreements within the app and ensuring user consent.

15. Cross-Platform Compatibility Module:

- Ensure that the app functions seamlessly on both iOS and Android devices.
- Implement cross-platform compatibility during the design and development process.

By carefully implementing each of these modules within MobiRoller, you can create a feature-rich and user-friendly EV Charging Station Finder App that meets the needs of electric vehicle owners and enthusiasts.

5.3 Running Application

Running an application created using the MobiRoller website involves several steps, from designing your app to deploying it on mobile devices. Here's a brief overview of the process:

1. Design Your App:

- Use the MobiRoller website to design your app's user interface and configure its features. You can customize the app's layout, content, and functionality using MobiRoller's drag-and-drop interface.

2. Content Creation:

- Add and organize content within your app. This may include text, images, videos, forms, and other multimedia elements. Ensure that your content aligns with your app's purpose and target audience.

3. Configuration and Settings:

- Configure various settings for your app, such as navigation menus, user authentication, payment integration (if applicable), push notifications, and social media integration.

4. Testing:

- Before publishing your app, thoroughly test it using MobiRoller's testing tools or by publishing a beta version to a limited audience. Testing helps identify and fix any issues or bugs.

5. Branding and Customization:

- Customize your app's branding by uploading logos, icons, and splash screens. Ensure that your app's branding aligns with your organization or business.

6. User Authentication:

- If your app requires user registration and authentication, configure these features using MobiRoller's modules.

7. Payment Integration (If Applicable):

- If your app involves e-commerce or in-app purchases, integrate a secure payment gateway (e.g., Stripe, PayPal) to enable transactions within the app.

8. Analytics and Monitoring:

- Implement analytics tools within your app to track user behavior, app performance, and engagement metrics. This data can help you make informed decisions for app improvements.

9. Publish Your App:

- Use MobiRoller's publishing features to publish your app to app stores such as the Google Play Store (for Android) or the Apple App Store (for iOS). Follow the guidelines and requirements of the respective app store.

10. User Engagement and Marketing:

- Once your app is live, focus on user engagement and marketing. Promote your app through various channels, including social media, email marketing, and your website.

11. App Maintenance:

- Regularly update your app to fix bugs, add new features, and ensure compatibility with the latest operating system versions. Listen to user feedback and make necessary improvements.

12. User Support:

- Provide user support through documentation, tutorials, and customer support channels to assist users with any issues or questions they may have.

13. Analytics and Optimization:

- Continuously monitor app analytics to gain insights into user behavior. Use this data to optimize your app, enhance user experience, and achieve your app's goals.

Running an app created with MobiRoller is a dynamic process that involves not only app development but also ongoing management, marketing, and user engagement. Regularly updating and improving your app is essential to its long-term success.

5.4 Configuring Database

MobiRoller is a mobile app builder platform that simplifies the process of creating mobile apps. While MobiRoller allows you to design and configure various app features, it may not provide direct access to configure a database. However, you can typically integrate your MobiRoller app with an external database or utilize cloud-based databases for storing and managing app data. Here's a general guide on how to configure a database for your MobiRoller app:

1. Choose a Database Solution:

- Decide whether you want to use a cloud-based database service (e.g., Firebase, AWS DynamoDB) or set up your own database server.

2. Set Up Your Database:

- If you choose a cloud-based service, follow the service's documentation to create a new database instance. Configure security settings, access controls, and data storage options as needed.
- If you're setting up your own database server, you'll need to choose a database management system (e.g., MySQL, PostgreSQL) and follow the installation and configuration instructions for your chosen system.

3. Define Your Database Schema:

- Plan your database schema based on the data your MobiRoller app will need to store. Define tables, fields, and relationships between data entities.

4. Secure Database Access:

- Implement strong security measures to protect your database. Set up authentication and authorization mechanisms to control who can access and modify data.

5. Create APIs or Endpoints:

- To interact with your database, you'll need to create APIs or endpoints that your MobiRoller app can communicate with. These APIs can be hosted on your server or through a cloud service like AWS Lambda.

6. Integrate APIs with MobiRoller:

- In your MobiRoller app, use the "Web View" or "API Integration" feature to connect to your database APIs. You can typically do this by specifying the API endpoints and methods (e.g., GET, POST) for retrieving and submitting data.

7. Testing and Debugging:

- Thoroughly test the integration between your MobiRoller app and the database. Ensure that data retrieval and submission work as expected. Debug any issues that arise.

8. Data Validation and Sanitization:

- Implement data validation and sanitization on both the client (MobiRoller app) and server (API endpoints) to prevent security vulnerabilities like SQL injection.

9. Monitoring and Maintenance:

- Set up monitoring tools to track the performance of your database and API endpoints. Regularly perform maintenance tasks such as data backups, updates, and security audits.

10. Scalability Planning:

- Plan for the scalability of your database as your app grows. Consider how to handle increased data volumes and traffic.

Please note that the specific steps and tools you use may vary depending on your chosen database solution and your app's requirements. Additionally, MobiRoller's capabilities and features may have evolved since my last update, so it's advisable to refer to the platform's documentation or contact their support for the most current guidance on integrating databases into your MobiRoller app.

5.5 Coding

MobiRoller is a mobile app builder platform that allows you to create mobile apps without extensive coding. It offers a user-friendly, drag-and-drop interface for designing and configuring your app's features. While it doesn't require traditional coding skills, you may need to use MobiRoller's visual features and configurations to achieve your desired functionality. Here's an overview of how to work with MobiRoller:

1. Sign In or Sign Up:

- Start by signing in to your MobiRoller account or creating a new one if you haven't already.

2. Create a New App:

- After logging in, you can create a new app project. Give your app a name and choose the platform (iOS, Android, or both).

3. Design Your App:

- Use the MobiRoller's intuitive drag-and-drop interface to design your app's user interface (UI). You can add screens, navigation menus, buttons, images, text, and other elements to create the app's layout.

4. Configuring Features: MobiRoller provides various modules and settings that allow you to configure different features for your app. These can include:

a) **User Registration and Authentication:** Configure user registration and login screens.
b) **Content Management:** Add and organize text, images, videos, and other content.
c) **Forms and Surveys:** Create interactive forms for data collection.
d) **E-commerce Integration:** Build product catalogs, shopping carts, and payment processing.
e) **Push Notifications:** Configure push notifications to engage users.
f) **Social Media Integration:** Integrate with social media platforms.
g) **User Analytics:** Implement analytics tools to track user behavior.
h) **User Management:** Manage user accounts and permissions.
i) **Custom Code Integration (Advanced):** Add custom code snippets or scripts if needed.

5. Content Population:

- Populate your app with content by adding text, images, videos, and other media. Organize content into categories or sections as needed.

6. Testing:

- MobiRoller typically offers testing tools that allow you to preview your app on different devices and simulate user interactions. Test your app thoroughly to ensure it works as expected.

7. Publishing:

- Once you're satisfied with your app, you can publish it to app stores. MobiRoller often provides features to guide you through the publishing process, including generating app store listings and assets.

8. User Support and Maintenance:

- After your app is live, provide user support and engage with user feedback. Regularly update your app to fix bugs, add new features, and stay compatible with the latest OS versions.

9. Analytics and Optimization:

- Use analytics data to monitor user engagement and app performance. Optimize your app based on user behaviour and feedback.

10. Community Building:

- Create a community around your app through social media, forums, or a dedicated website. Engaging with users can lead to word-of-mouth marketing and user-generated content.

While MobiRoller simplifies app development, it's essential to plan your app's functionality and design carefully to meet your users' needs effectively. MobiRoller's specific features and capabilities may have evolved since my last update, so referring to their latest documentation and tutorials is advisable for the most up-to-date information on creating apps in MobiRoller.

5.5.1 Sample Code / Extracted Code

```
package androidx.core.app;
import android.app.Activity;
import android.app.SharedElementCallback;
import android.content.Context;
import android.content.ContextWrapper;
import android.content.Intent;
import android.content.IntentSender;
import android.content.pm.PackageManager;
import android.graphics.Matrix;
import android.graphics.RectF;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Parcelable;
import android.text.TextUtils;
import android.view.Display;
import android.view.DragEvent;
import android.view.View;
import androidx.core.app.ActivityCompat;
import androidx.core.app.SharedElementCallback;
import androidx.core.content.ContextCompat;
import androidx.core.content.LocusIdCompat;
import androidx.core.view.DragAndDropPermissionsCompat;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
```

```
public class ActivityCompat extends ContextCompat {
    private static PermissionCompatDelegate sDelegate;

    public interface OnRequestPermissionsResultCallback {
        void onRequestPermissionsResult(int i, String[] strArr, int[] iArr);
    }

    public interface PermissionCompatDelegate {
        boolean onActivityResult(Activity activity, int i, int i2, Intent intent);

        boolean requestPermissions(Activity activity, String[] strArr, int i);
    }

    public interface RequestPermissionsRequestCodeValidator {
        void validateRequestPermissionsRequestCode(int i);
    }

    protected ActivityCompat() {
    }

    public static void setPermissionCompatDelegate(PermissionCompatDelegate
permissionCompatDelegate) {
        sDelegate = permissionCompatDelegate;
    }

    public static PermissionCompatDelegate getPermissionCompatDelegate() {
        return sDelegate;
    }
    @Deprecated
    public static boolean invalidateOptionsMenu(Activity activity) {
        activity.invalidateOptionsMenu();
        return true;
    }

    public static void startActivityForResult(Activity activity, Intent intent, int i,
Bundle bundle) {
        if (Build.VERSION.SDK_INT >= 16) {
            Api16Impl.startActivityForResult(activity, intent, i, bundle);
        } else {
            activity.startActivityForResult(intent, i);
        }
    }

    public static void startIntentSenderForResult(Activity activity, IntentSender
intentSender, int i, Intent intent, int i2, int i3, int i4, Bundle bundle) throws
IntentSender.SendIntentException {
        if (Build.VERSION.SDK_INT >= 16) {
            Api16Impl.startIntentSenderForResult(activity, intentSender, i, intent,
i2, i3, i4, bundle);
        } else {
            activity.startIntentSenderForResult(intentSender, i, intent, i2, i3, i4);
        }
    }

    public static void finishAffinity(Activity activity) {
        if (Build.VERSION.SDK_INT >= 16) {
            Api16Impl.finishAffinity(activity);
        } else {
            activity.finish();
        }
    }

    public static void finishAfterTransition(Activity activity) {
        if (Build.VERSION.SDK_INT >= 21) {
            Api21Impl.finishAfterTransition(activity);
        } else {
            activity.finish();
        }
    }

    public static Uri getReferrer(Activity activity) {
        if (Build.VERSION.SDK_INT >= 22) {
            return Api22Impl.getReferrer(activity);
        }
        Intent intent = activity.getIntent();
        Uri uri = (Uri)
intent.getParcelableExtra("android.intent.extra.REFERRER");
        if (uri != null) {
            return uri;
        }
        String stringExtra =
intent.getStringExtra("android.intent.extra.REFERRER_NAME");
        if (stringExtra != null) {
            return Uri.parse(stringExtra);
        }
        return null;
    }

    public static <T extends View> T requireViewById(Activity activity, int i) {
        if (Build.VERSION.SDK_INT >= 28) {
            return (View) Api28Impl.requireViewById(activity, i);
        }
        T findViewById = activity.findViewById(i);
        if (findViewById != null) {
            return findViewById;
        }
        throw new IllegalArgumentException("ID does not reference a View
inside this Activity");
    }

    public static void setEnterSharedElementCallback(Activity activity,
SharedElementCallback sharedElementCallback) {
        if (Build.VERSION.SDK_INT >= 21) {
            Api21Impl.setEnterSharedElementCallback(activity,
sharedElementCallback != null ? new
SharedElementCallback21Impl(sharedElementCallback) : null);
        }
    }

    public static void setExitSharedElementCallback(Activity activity,
SharedElementCallback sharedElementCallback) {
        if (Build.VERSION.SDK_INT >= 21) {
            Api21Impl.setExitSharedElementCallback(activity,
sharedElementCallback != null ? new
SharedElementCallback21Impl(sharedElementCallback) : null);
        }
    }

    public static void postponeEnterTransition(Activity activity) {
        if (Build.VERSION.SDK_INT >= 21) {
            Api21Impl.postponeEnterTransition(activity);
        }
    }

    public static void startPostponedEnterTransition(Activity activity) {
        if (Build.VERSION.SDK_INT >= 21) {
            Api21Impl.startPostponedEnterTransition(activity);
        }
    }

    public static void requestPermissions(final Activity activity, final String[]
strArr, final int i) {
        PermissionCompatDelegate permissionCompatDelegate = sDelegate;
        if (permissionCompatDelegate == null ||
!permissionCompatDelegate.requestPermissions(activity, strArr, i)) {
            int length = strArr.length;
            int i2 = 0;
            while (i2 < length) {
                if (!TextUtils.isEmpty(strArr[i2])) {
                    i2++;
                } else {
```

```
                throw new IllegalArgumentException("Permission request for
permissions " + Arrays.toString(strArr) + " must not contain null or empty
values");
            }
        }
        if (Build.VERSION.SDK_INT >= 23) {
            if (activity instanceof RequestPermissionsRequestCodeValidator) {
                ((RequestPermissionsRequestCodeValidator)
activity).validateRequestPermissionsRequestCode(i);
            }
            Api23Impl.requestPermissions(activity, strArr, i);
        } else if (activity instanceof OnRequestPermissionsResultCallback) {
            new Handler(Looper.getMainLooper()).post(new Runnable() {
                public void run() {
                    int[] iArr = new int[strArr.length];
                    PackageManager packageManager =
activity.getPackageManager();
                    String packageName = activity.getPackageName();
                    int length = strArr.length;
                    for (int i = 0; i < length; i++) {
                        iArr[i] = packageManager.checkPermission(strArr[i],
packageName);
                    }
                    ((OnRequestPermissionsResultCallback)
activity).onRequestPermissionsResult(i, strArr, iArr);
                }
            });
        }
    }
```

6. SYSTEM TESTING

6.1 Testing Introduction

Testing is a fundamental and critical phase in the software development lifecycle, aimed at ensuring the quality, reliability, and functionality of a software product or system. It involves systematically evaluating a software application or component to identify defects, errors, and discrepancies between the expected and actual behavior. Testing serves several essential purposes, including verifying that the software meets its specified requirements, validating that it functions correctly under various conditions, and ensuring that it can withstand the rigors of real-world usage.

Software testing encompasses a wide range of activities and methodologies, including manual and automated testing, white-box and black-box testing, functional and non-functional testing, among others. Manual testing involves human testers systematically executing test cases to simulate end-user interactions and evaluate the software's performance, usability, and functionality. Automated testing, on the other hand, utilizes software tools to execute test scripts and compare actual outcomes with expected results, streamlining the testing process and improving repeatability.

In addition to functional testing, non-functional testing focuses on aspects such as performance, security, scalability, and reliability. Performance testing evaluates how well the software performs under various loads and conditions, ensuring that it can handle the expected volume of users or transactions. Security testing assesses the software's resistance to security threats and vulnerabilities, aiming to protect sensitive data and prevent breaches. Scalability testing explores how the software adapts to changing workloads and resource demands, while reliability testing seeks to ensure that the software operates consistently without unexpected failures.

6.2 TEST CASES

6.2.1 UNIT TESTING

Unit testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

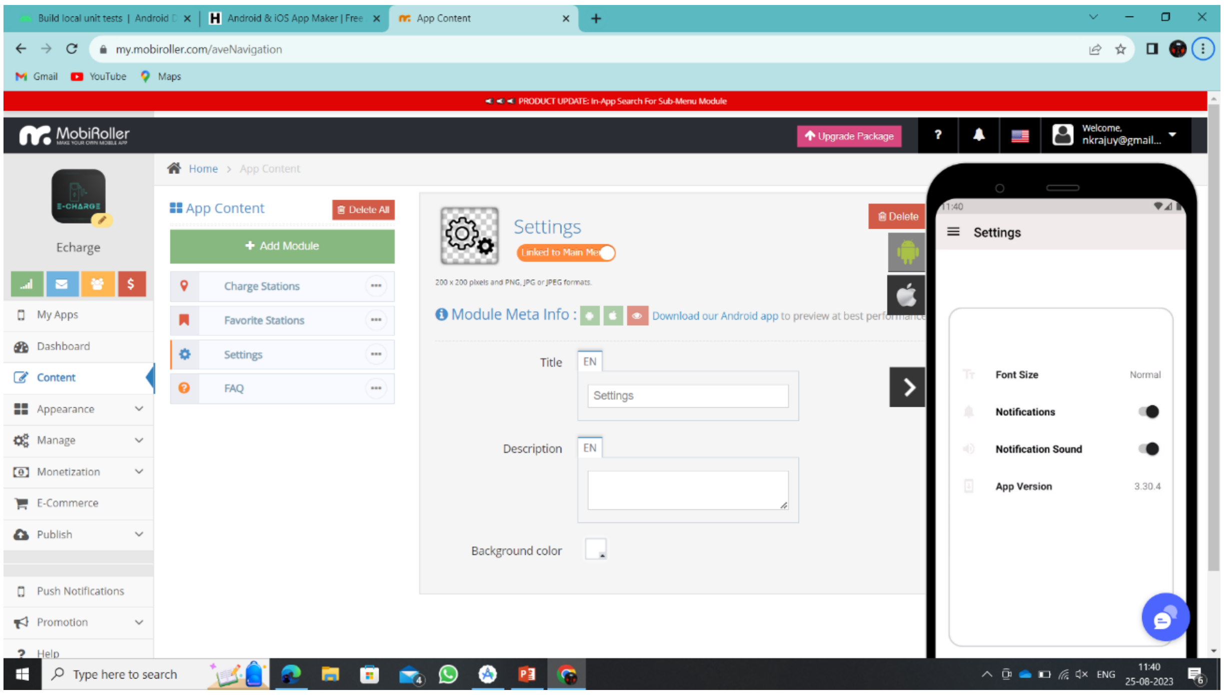
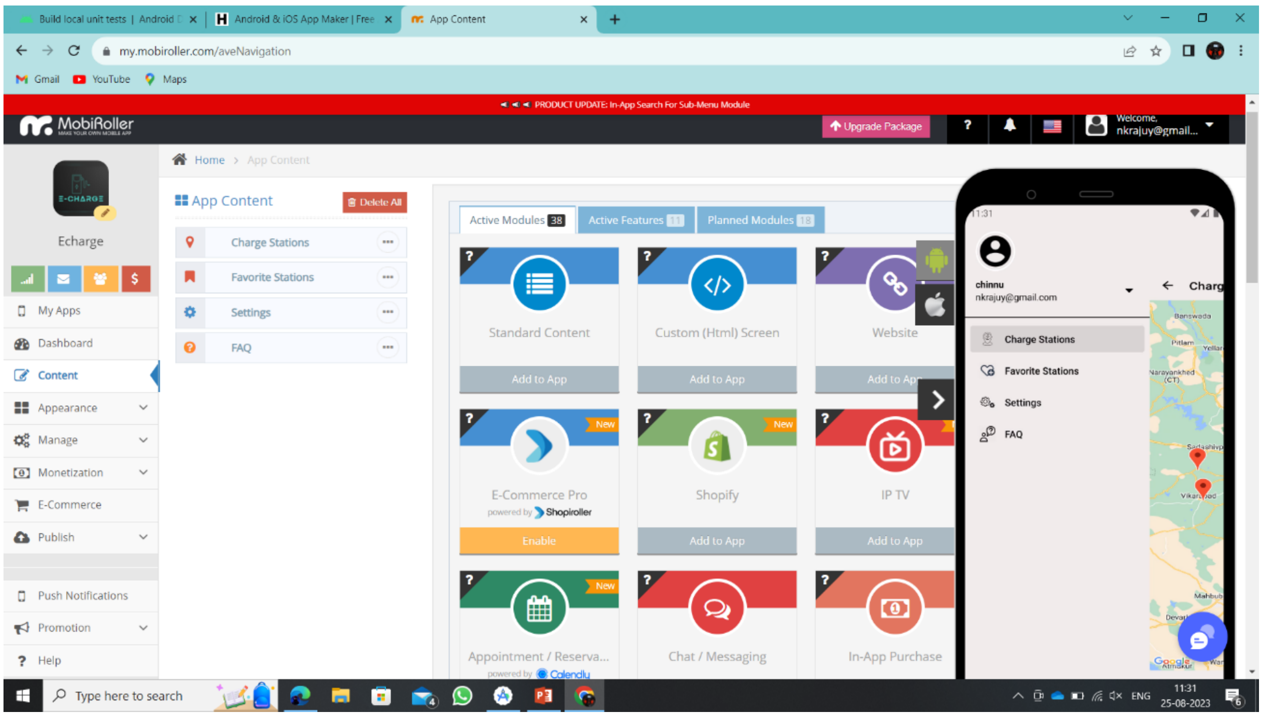


Figure: 6.2.1 Results of Unit Testing

6.2.2 INTEGRATION TESTING

Unit testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.



s

Figure: 6.2.2 Results of Integration Testing

6.2.3 API TESTING

API (Application Programming Interface) testing is a crucial aspect of software testing that focuses on evaluating the functionality, performance, security, and reliability of an application's APIs. APIs are sets of rules and protocols that allow different software systems to communicate and interact with each other, enabling the seamless exchange of data and services.

In summary, API testing is an integral part of software quality assurance, ensuring that APIs function correctly, perform well, and remain secure, contributing to the overall reliability and functionality of software applications that rely on these interfaces.

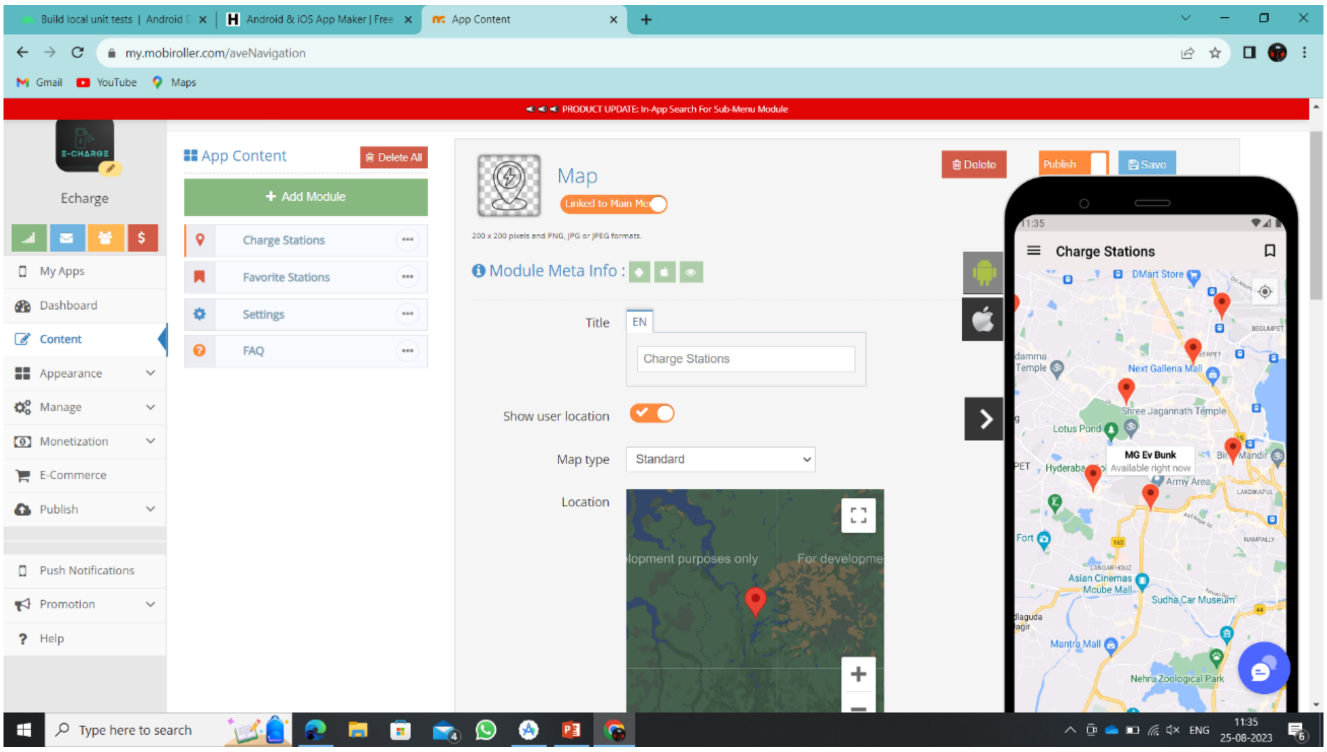


Figure: 6.2.3 Results of API Testing

6.2.4 UI/UX TESTING

UI/UX testing, also known as User Interface (UI) and User Experience (UX) testing, is a crucial phase in software development aimed at evaluating the usability, design, and overall user satisfaction with a digital product, such as a website, mobile app, or software application. This type of testing focuses on ensuring that the user interface is intuitive, visually appealing, and functional while providing a positive and efficient user experience.

UI/UX testing is essential for delivering a product that not only meets functional requirements but also delights users by offering an enjoyable and efficient interaction. By addressing these aspects during testing, software developers and designers can refine the user interface and experience, leading to higher user satisfaction and better overall product success.

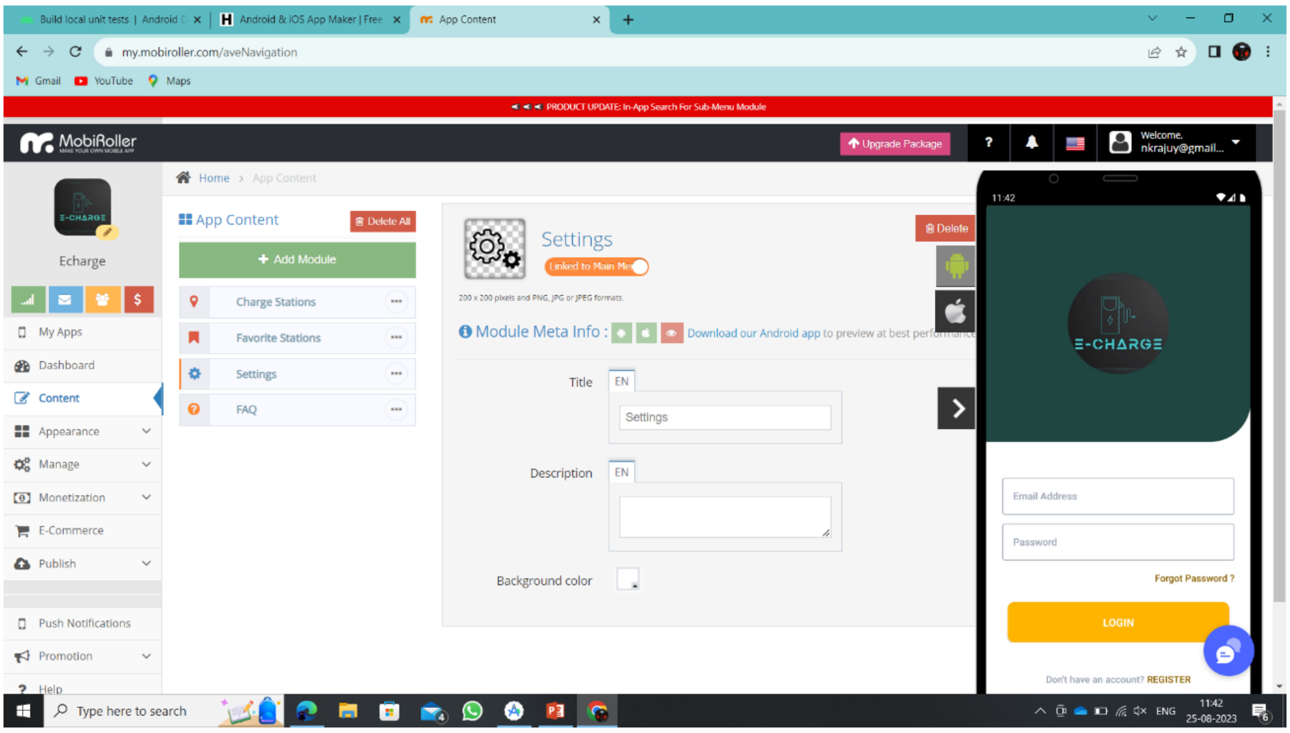


Figure: 6.2.4 Results of UI/UX Testing

6.2.5 LOCATION SERVICE TESTING

Location service testing is a crucial component of quality assurance for mobile applications that rely on geospatial data, such as navigation or location-based services. This type of testing focuses on ensuring that the app's location-related features work accurately and reliably, providing users with the expected experience.

During location service testing, various aspects are assessed. Accuracy is a fundamental concern, ensuring that the app accurately pinpoints a user's location, especially vital for EV Charging Station Finder Apps where precise location data is essential for finding nearby charging stations.

Handling permissions and privacy is equally important. Apps must request and manage location permissions effectively, respecting user privacy and adhering to relevant data protection regulations.

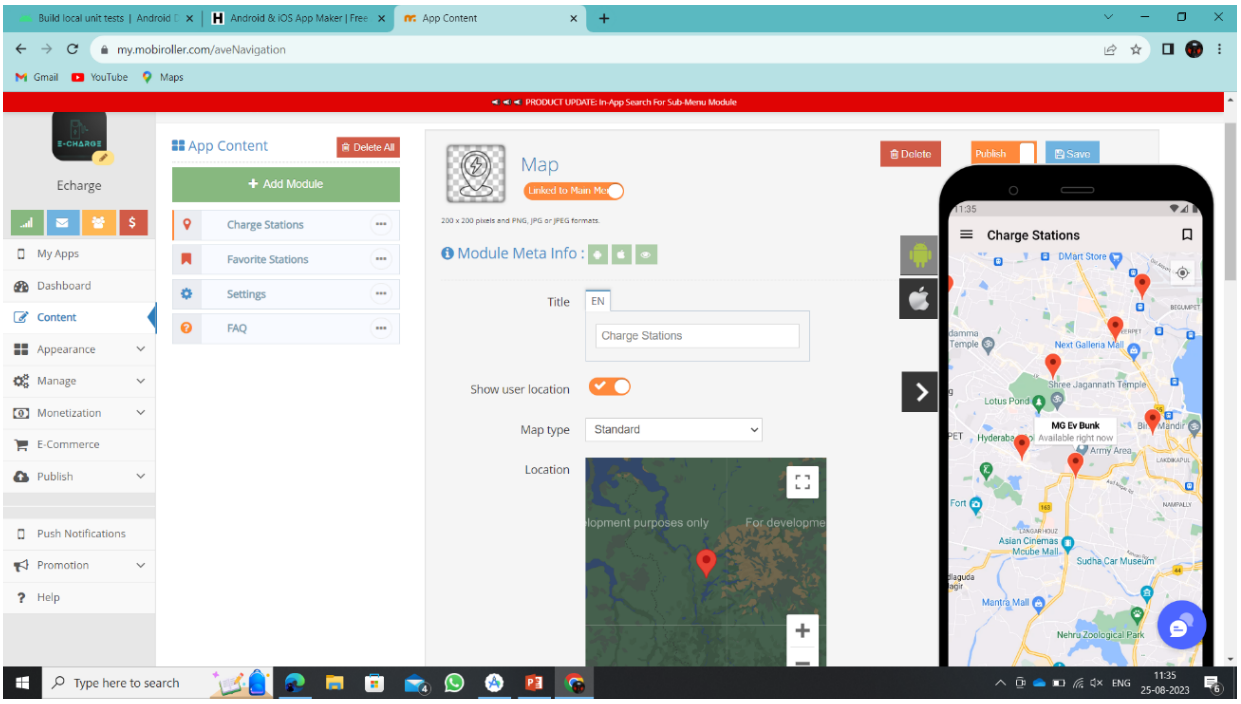
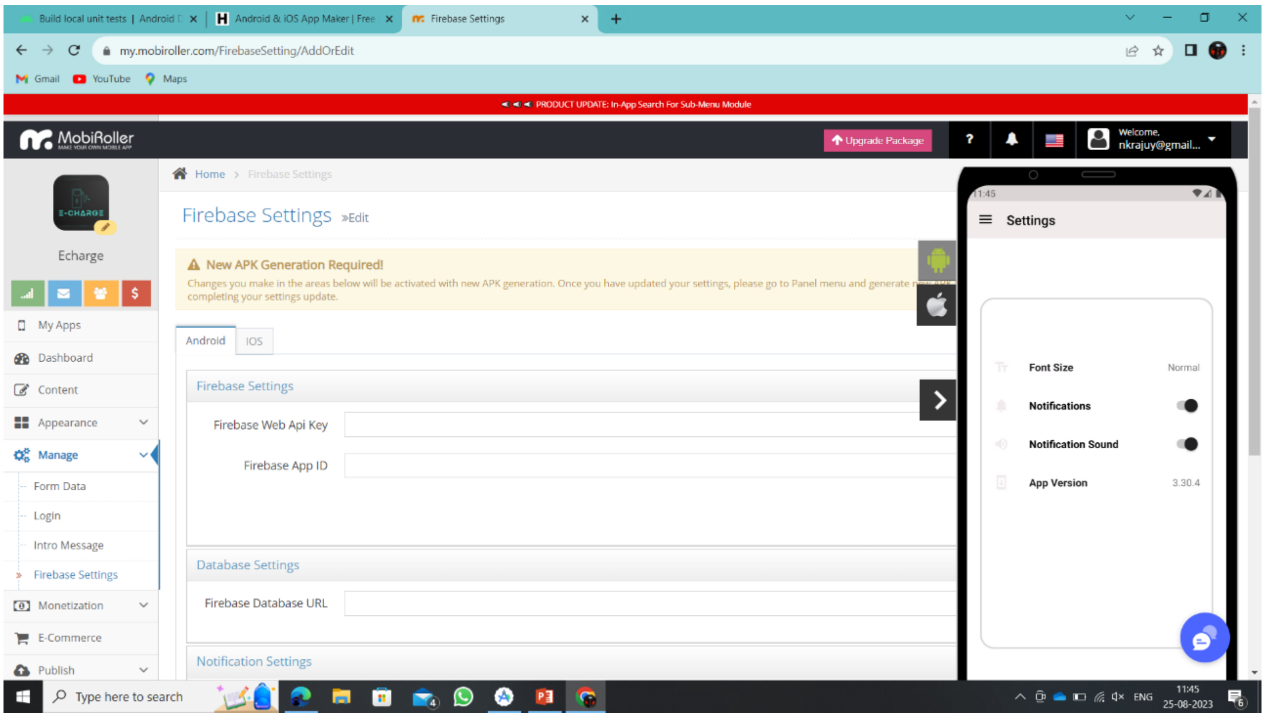


Figure: 6.2.5 Results of Location Service Testing

6.2.6 REGRESSION TESTING

Regression testing is a critical quality assurance process in software development that focuses on ensuring the continued functionality and stability of an application after code changes, updates, or new features are introduced. Its primary objective is to detect any unintended side effects or defects that may have been introduced as a result of these modifications.

During regression testing, a predefined set of test cases, which cover various aspects of the software's functionality, are re-executed to verify that the existing functionality remains intact. This testing technique helps in identifying and fixing any issues promptly, ensuring that the software continues to meet its original specifications and user expectations. Regression testing is typically automated to speed up the testing process, especially when frequent code changes are made.



Firebase Settings »Edit

New APK Generation Required!

Changes you make in the areas below will be activated with new APK generation. Once you have updated your settings, please go to Panel menu and generate n... completing your settings update.

Firebase Settings

Firebase Web Api Key

Firebase App ID

Firebase Database URL

Font Size Normal

Notifications

Notification Sound

App Version 3.30.4

Figure: 6.2.6 Results of Regression Testing

6.3 BLACK BOX TESTING

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioural Testing. There are many types of Black Box Testing but the following are the prominent ones –

- **Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** – Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

7. OUTPUT SCREENS

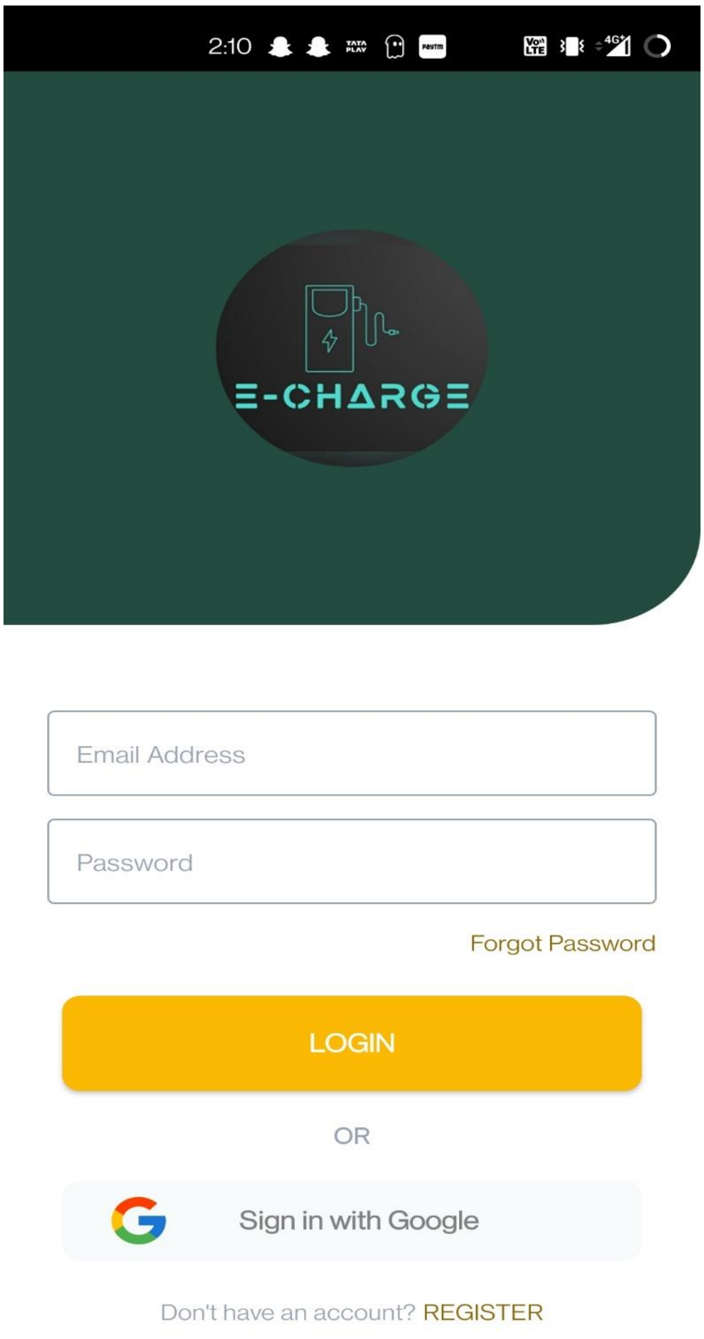


Figure: 7.1 Login Page

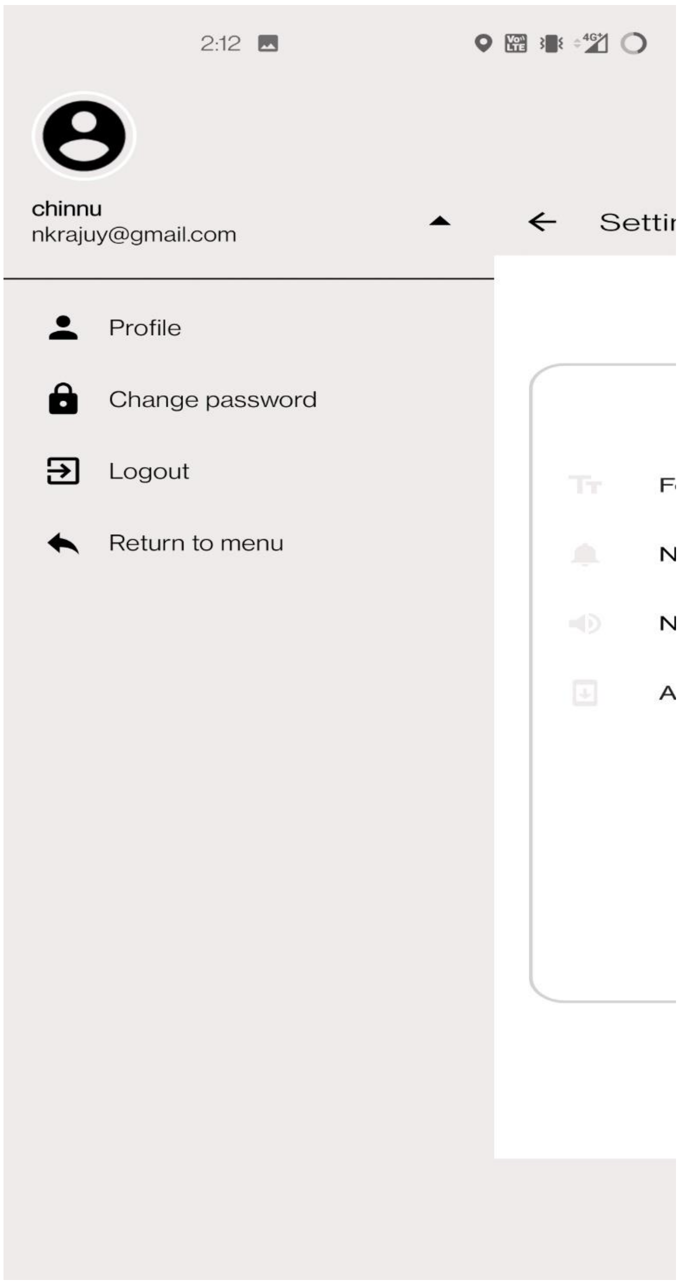


Figure: 7.2 User Profile

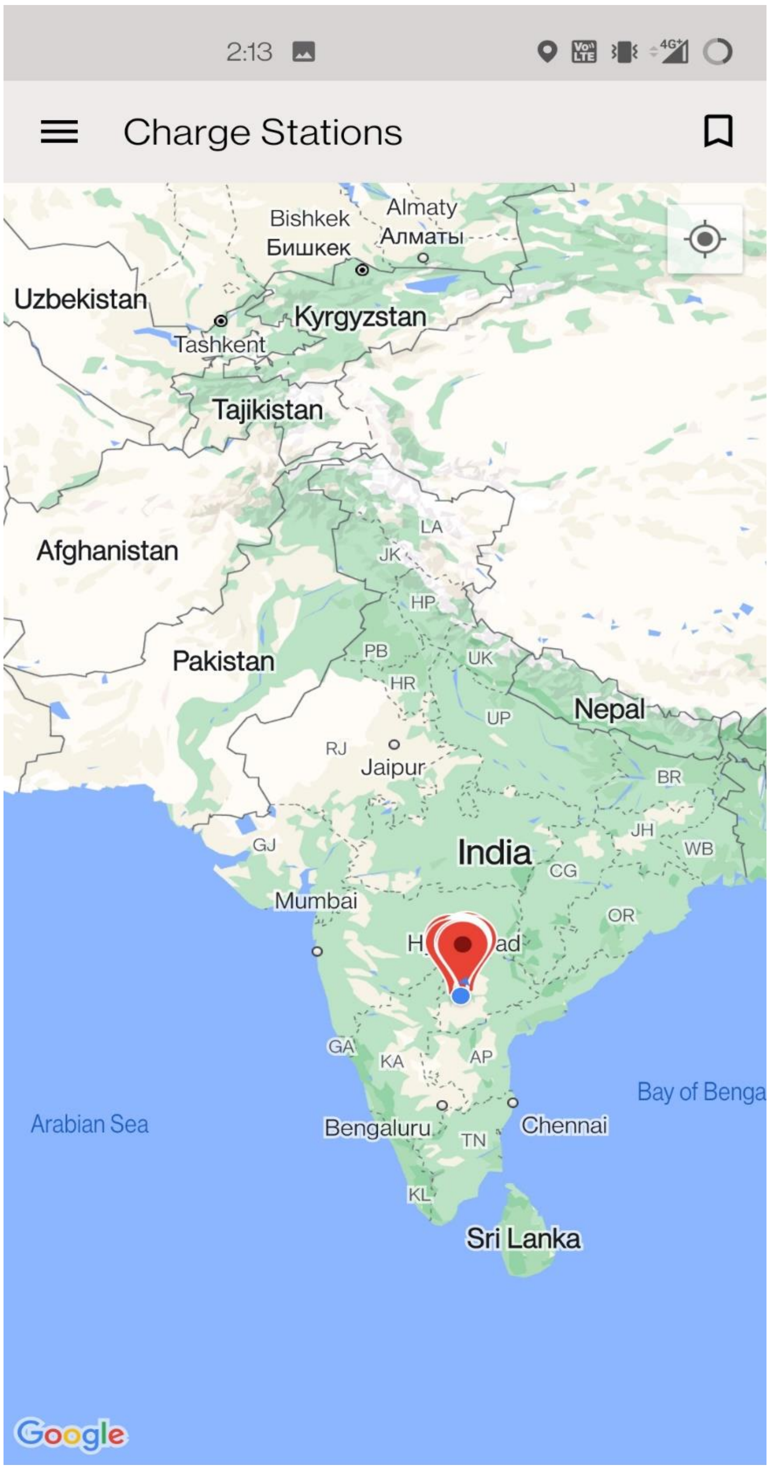


Figure: 7.3 Map Integration

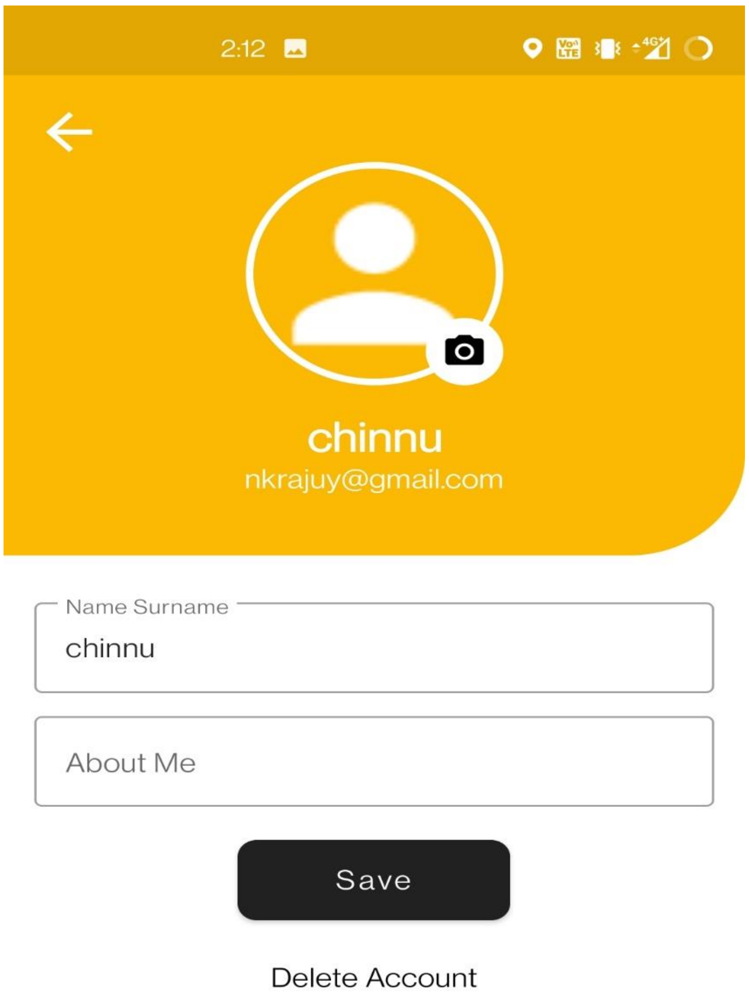
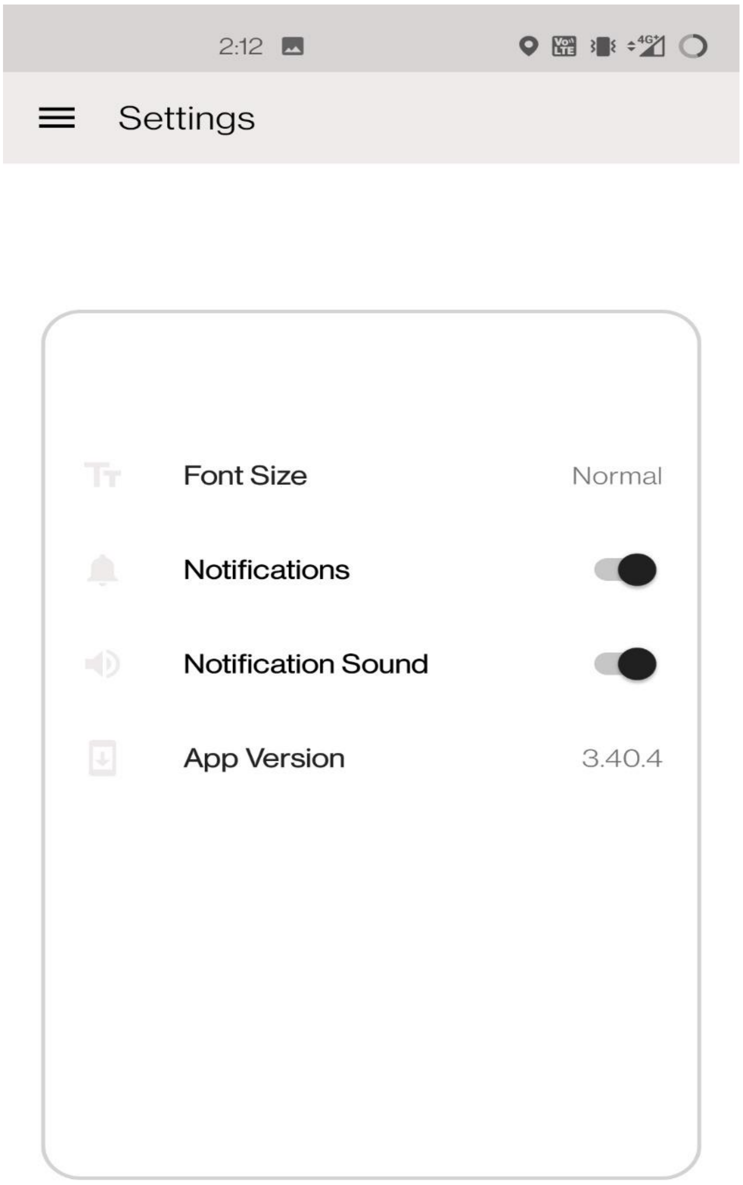


Figure: 7.4 User Interface


Settings

Font Size Normal

Notifications

Notification Sound

App Version 3.40.4

Figure: 7.5 Settings Interface

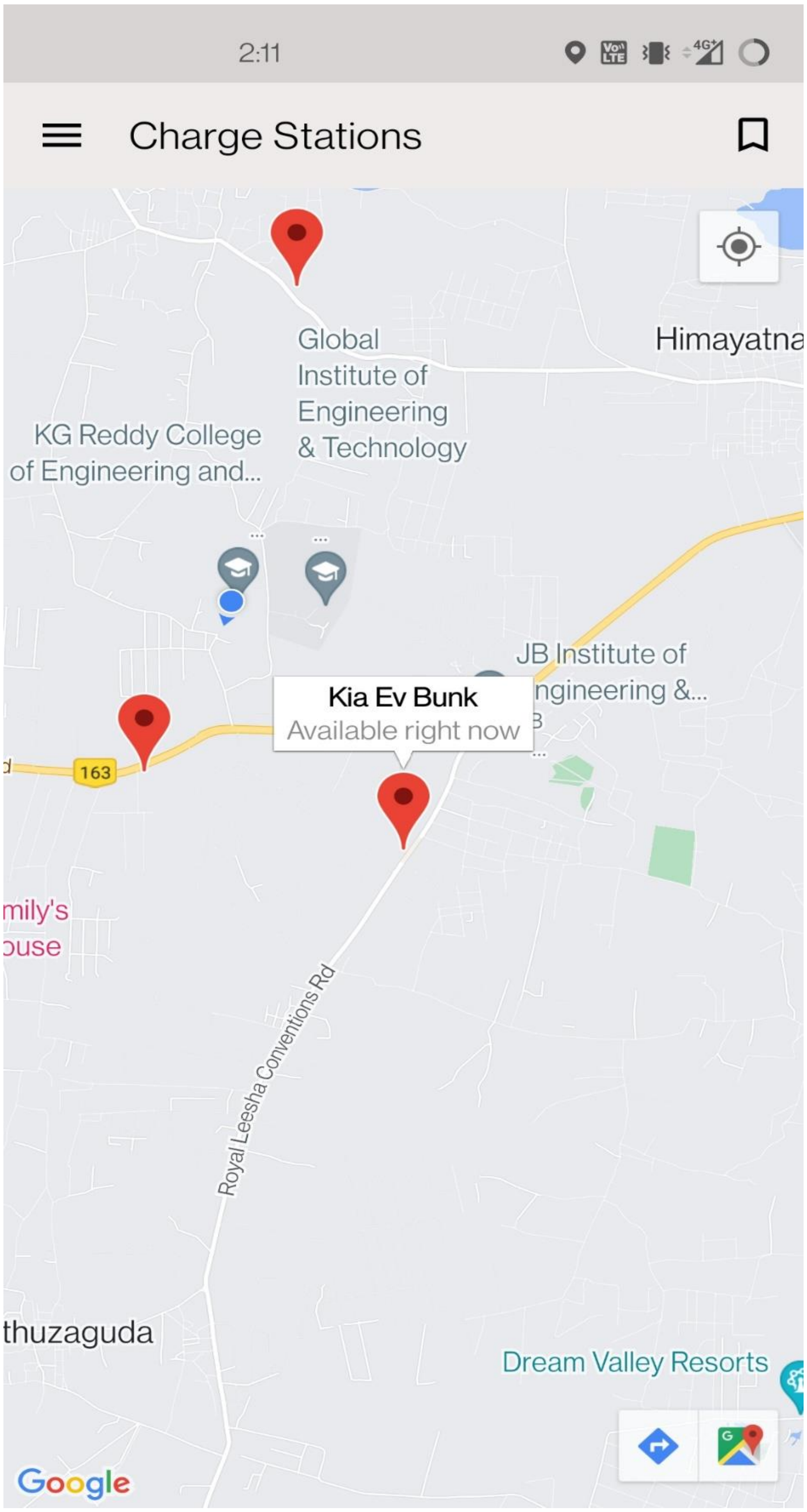

2:11
Charge Stations
Himayatna
Global Institute of Engineering & Technology
KG Reddy College of Engineering and...
JB Institute of ngineering &...
Kia Ev Bunk
Available right now
163
mily's ouse
Royal Leesha Conventions Rd
thuzaguda
Dream Valley Resorts
Google

Figure: 7.6 Charge Stations

8. CONCLUSION

In conclusion, the EV Charging Station Finder App is a valuable and innovative solution for electric vehicle owners and enthusiasts. It addresses the growing demand for convenient access to charging infrastructure while promoting sustainability and reducing the barriers to EV adoption. This app's user-friendly interface, real-time data updates, and location-based services make it an indispensable tool for those seeking efficient and reliable charging solutions.

The scope and goals of such an app extend beyond mere convenience; they align with broader environmental and societal objectives, including reducing carbon emissions, supporting renewable energy adoption, and promoting sustainable mobility solutions. Moreover, the EV Charging Station Finder App often represents a collaborative effort involving stakeholders such as EV manufacturers, charging station operators, and government agencies, all working together to create a more sustainable and efficient transportation landscape.

As the electric vehicle market continues to expand, the EV Charging Station Finder App serves as a pivotal tool in shaping the future of transportation, driving innovation in charging infrastructure, and empowering users to make eco-conscious choices. Its significance lies not only in its convenience but also in its contribution to a cleaner and more sustainable world.

9. FUTURE ENHANCEMENT

The future scope of an EV Charging Station Finder App is promising and will likely see continuous growth and innovation in several key areas:

1. **Integration of Renewable Energy:** With an increasing emphasis on sustainability, the app may evolve to include information about charging stations powered by renewable energy sources, such as solar or wind. This would align with the growing interest in eco-friendly charging options.

2. **Enhanced Route Planning:** Future versions of the app may provide even more sophisticated route planning features. This could include optimizing routes based on real-time traffic conditions, weather, and charging station availability.

3. **Electric Vehicle Support:** As electric vehicle technology advances, the app may need to adapt to support new vehicle models, different charging connector types, and varying charging rates. It might also incorporate features tailored to specific vehicle brands and models.

4. **Payment Integration and Subscriptions:** The app may further streamline the payment process for charging sessions. It could facilitate mobile payments directly within the app, offer subscription-based charging plans, and provide discounts or loyalty rewards for frequent users.

The main purpose of the project is to develop a useful product for EV users which will be very convenient for them. This app will not only provide service to the user but will also be used by the admin as an interactive system. It can also generate more data about the user who owns electric vehicles and the owners of the charging station. One can use it to find as well as navigate to stations. This app will also be expanded in the future as a commercial product with more features that will also use subscription packs, as well as features like charge and chill which will generate more revenue.

10. REFERENCES

[1] H. Li and L. Zhijian , "The study and implementation of mobile GPS navigation system based on Google Maps," in International Conference on Computer and Information Application, Tianjin, China, 2010.

[2] H. A. A. Dafallah, "Design and implementation of an accurate real time GPS tracking system," in The Third International Conference on e-Technologies and Networks for Development, Beirut, Lebanon, 2014.

[3] K. Nagaraj, B. Prabakaran and M. O. Ramkumar, "Application Development for a Project using Flutter," in 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2022.

[4] S. Boukhary and E. Colmenares , "A Clean Approach to Flutter Development through the Flutter Clean Architecture Package," in 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019.

[5] Y. Cheon, "Toward More Effective Use of Assertions for Mobile App Development," in IEEE International Conference on Progress in Informatics and Computing (PIC), Shanghai, China, 2021.

[6] Nishant S. Chaturkar , Rahul B. Lanjewar , Shreyash B. Wadaskar and Khushal D. Ingole , "Electric Vehicle Charging Station Finding App," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT, vol. 2, no. 2, pp. 50-60, 2022).

[7] S. Sharma, S. Khare, V. Unival and S. Verma , "Hybrid Development in Flutter and its Widgits," in 2022 International Conference on Cyber Resilience (ICCR), Dubai, United Arab Emirates, 2022.

[8] P. Aji, D. A. Renata, A. Larasati and Riza. , "Development of Electric Vehicle Charging Station Management System in Urban Areas," in 2020 International Conference on Technology and Policy in Energy and Electric Power (ICT-PEP), Bandung, Indonesia, 2020.

[9] N. Matanov, A. Zahov and I. Angelov , "Modeling of the Electric Vehicle Charging Process - Part 1," in 2021 13th Electrical Engineering Faculty Conference (BulEF), Varna, Bulgaria, 2021.

[10] D. Gong, M. Tang, B. Buchmeister and H. Zhang , "Solving Location Problem for Electric Vehicle Charging Stations—A Sharing Charging Model," IEEE Access, vol. 7, no. 9, pp. 138391-138402, 2019.

[11] J. Tan and L. Wang , "Real-Time Charging Navigation of Electric Vehicles to Fast Charging Stations: A Hierarchical Game Approach," IEEE Transactions on Smart Grid, vol. 8, no. 2, pp. 846-856, 2017.

[12] B. Al-Hanahi, I. Ahmad, D. Habibi and M. A. S. Masoum , "Charging Infrastructure for Commercial Electric Vehicles: Challenges and Future Works," IEEE Access, vol. 9, no. 2, pp. 121476-121492, 2021.