FIND GREEN – ELECTRIC VEHICLE CHARGING BOOKING APPLICATION

COMPANY : OPENTURF TECHNOLOGIES

PURPOSE : INTERN PROJECT

DURATION : JAN 2023 – MAY 2023

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SYNOPSIS

Everyone these days knows the importance of saving fuel for the future generations. If the fuel usage is not keep on by the people, it will be impossible for future generations to even know what it was. As this thought is thriving incredibly these days, the automobile brands are also trying to make innovations that can lead us to save petrol on a vast scale. Keeping this idea in mind, automobile companies have created a revolutionary idea of electric vehicles.

Electric Vehicles are the future of the automobile industry. The companies manufacturing EVs have also set up charging stations across the country where the people can plug in their electric vehicle (EV) to recharge its batteries. These stations are typically found at public locations. Some EV charging stations are free to use, while others may require a fee. There comes a need for an application or a solution that helps in locating these charging stations to the owners of such vehicles.

The Find Green – Electric Vehicle Charging Station Finder App is a web application designed to help electric vehicle owners easily locate and navigate to charging stations in their area. The app provides a user-friendly interface for finding and selecting charging stations, as well as useful information about each station, such as availability, charging speed, and cost. This allows the user to book the available slots and make the payments online and get notified.

Find Green – Electric Vehicle Charging Station Finder Project is a cross platform system that will be used by EV owners. The solution is expected to display the nearby charging stations, their details such as available slots, type of connectors/plugs, charging types available, payment details.

1 INTRODUCTION

1.1 Company Profile

Openturf Technologies Private Limited is a world-class technology company specializing in the development of innovative software solutions that leverage the latest advances in Artificial Intelligence (AI) and Machine Learning (ML). The focus is on delivering cutting-edge technology solutions that drive business growth and help organizations stay ahead of the competition. The team of experts is dedicated to developing and implementing world-class CTO (Chief Technology Officer) models that help organizations optimize their operations, improve efficiency, and streamline their business processes.

At Openturf, they use AI and ML to develop customized solutions that meet the unique needs of our clients. Whether it's predictive analytics, natural language processing, computer vision, or any other cutting-edge technology, they have the expertise and experience to help organizations take full advantage of these powerful tools.Their innovative approach to software development has made us a leader in the industry, attracting top talent from around the world and building a reputation for delivering high-quality, reliable solutions. Their portfolio of services includes custom software development, mobile app development, web development, and data analysis and visualization. They believe in a collaborative approach to problem-solving and work closely with our clients to understand their unique needs and develop customized solutions that meet those needs.

At Openturf, they are committed to the highest standards of quality and security in all of their projects. They use the latest technologies and best practices to ensure that our solutions are both efficient and effective.

1.2 Project Overview

As of now electric charging stations are limited in India due to which people cannot find the right charging station which will save their time and money. EV charging stations requires space like parks, malls, societies. For private and semipublic charging stations, this space is available in the parking areas of the societies, apartment buildings, or of commercial or public or institutional areas. Due to this there is more difficulty for EV owners to find charging stations nearby them.

The problem is not only to find the charging station but also to charge it quickly because of the time required to charge the EV's. This leads to inconvenience of EV users as requires a lot of time, so need of slot booking is require in the charging of EV's. As electrical vehicle industry is growing in India and less charging stations are available and also new registrations of charging station is growing, so there is no availability of these growing charging station on virtual Maps. This leads to inconvenience of user for finding charging station virtually.

When a customer buys an electric vehicle, the maintenance of these vehicle is not like the ordinary cars. One have to seek some help such as an Electric vehicle charging station finder system to find charging stations. An electric vehicle charging station finder app can save the time to find these charging station rather than search independently. One cannot find charging stations like the petrol or diesel or CNG station which are available everywhere. Due to this problem, the developers have to plan the charging of these vehicles, but with the help of this apps which directly navigate the user to nearby EV charging stations.

An Find Green-Electric Vehicle Charging Station Finder System will show the nearby location of charging across stations across the locality as well as nearby the destination. The user will get various information about the stations such as how many ports are available and how variety of chargers available at that station. The app provides real-time availability of the stations, photos of the stations, and cost of the charging of vehicle at the station.

In this project, the developers will design and develop an app and website which will find nearby charging stations of users locality. The system will show all nearby electric vehicle charging stations. The user can directly navigate to this charging stations. This system will provide a facility of booking slots for charging the Electric vehicle of the users based on the type and charging port of their vehicle in their convenient time slots. This system will save a lot of time of Electric vehicle owners.

Find Green – Electric Vehicle Charging Station Finder Project is a cross platform system that will be used by EV owners. The solution is expected to display the nearby charging stations, their details such as available slots, type of connectors/plugs, charging types available, payment details. Find Green – Electric Vehicle Charging Station Finder Project can be accessed through both mobile (Android and IOS) and Web application interface.

1.3 System Configuration

The software and hardware needed for the development and implementation of the application are discussed in the following sub-sections under hardware requirements and software requirements.

1.3.1 Hardware Requirements

Processor	:	Intel®Core™ i3-6006U CPU@2.00GHz
RAM	:	4GB
Hard Disk	:	500GB

1.3.2 Software Requirements

Operating system	:	WINDOWS 10
Frontend	:	React Native/ React JS
Backend	:	NodeJS

Database	:	PostgreSQL

2 SYSTEM ANALYSIS

2.1 Existing System

• ***Limited personalization:*** An existing app may not have the ability to provide personalized recommendations based on the user's past charging habits and preferences.

• ***Usage tracker:*** Existing EV apps do not have any usage tracker for the users making it less user friendly.

• ***Limited charging options:*** The existing apps only show charging stations for certain networks or providers, which could limit the options for users.

• ***Limited information:*** Many of the existing apps do not provide all the information that users need, such as the types of connectors available at a charging station or the cost of charging.

2.2 Proposed System

- ***Real-time charging station location and availability information:*** The app would allow the users to quickly find the nearest charging station and check the availability of charging spots in real-time. Also, information like charging connectors and cost would be displayed.

- ***Slot Booking system:*** The users would be able to reserve charging spots in advance, reducing the chances of arriving at a charging station only to find all spots taken.
- ***Advanced features such as route optimization:*** An EV app can take into account the remaining charge of the EV and also the location of charging stations. This feature would allow the user to optimize the route and reach the destination with the least amount of charge required.

- ***Personalized recommendations:*** The users would be able to receive personalized recommendations for charging stations based on their past charging habits and preferences.

- ***My Usage Feature:*** The user would be able to view their daily/weekly/monthly usage of Power consumed and Cost spent.

2.3 Requirement Analysis

The following are the modules that are required for the development of this project. • Registration

- Navigation
- Notification
- Charging Stations
- Search

• Filter

- Slot Booking
- User Profile
- Charging History
- My Bookings

Registration:

The users should be able to register themselves and the users have to fill in the sign up details. Users need to enter login details to access the application.

Notification:

The users will receive notifications for charging stations available, reminders for the booking made etc.

Navigation:

With the help of GPS and Google Maps (APIs) users can track to find stations nearby. Users can also get directions to the CS. Google Maps can be used to navigate across the location.

Search and Filter:

Search for the Charging Station and Various filters can be used to sort the available station.

Slot Booking:

The users can book the charger in a particular station for a particular date and for specified amount of time.

Charging Stations and History:

Lists the details of all the available Charging Stations to book slots in advance and the users charging history.

My Bookings:

Upcoming and cancelled booking details are to be displayed.

Payments:

The users can have options to make payment via app using different payment options like debit/credit card, wallets and others.

2.4 System Study

2.4.1 Node JS

Whenever a client requests something from the client side of the application what happens is, the request is first sent to the server and then in that server some processing goes on for the validation of the client side request and after doing all such validation a response is sent to the client side. Basically for doing all such calculations and processing , the NodeJs framework of JavaScript is used.

For running the web applications outside the client's browser , NodeJS is basically used as an open-source and cross platform JavaScript runtime environment. For running the server side applications the developers use this. For building the I/O intensive applications like video streaming sites ,online chatting applications and many other applications , it is used. Many established tech giant companies and newly created start-ups are using NodeJs framework in their company.

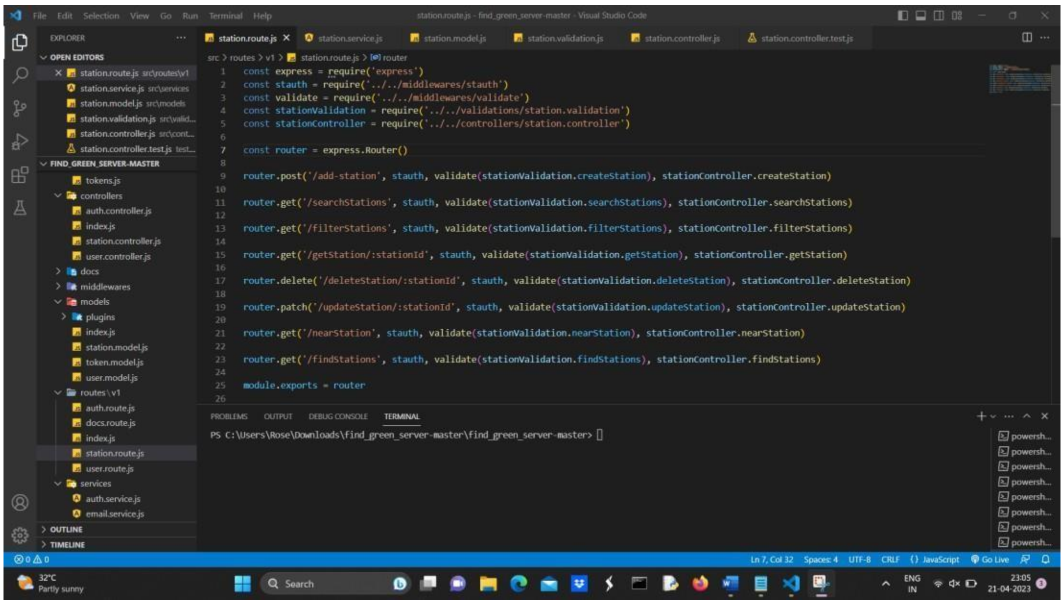
Node.js is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser. Developers use Node.js to create server-side web applications, and it is perfect for data-intensive applications since it uses an asynchronous, event-driven model.

Express is a flexible Node.js web application framework that provides a wide set of features to develop both web and mobile applications. It's a layer built on the top of the Node.js that helps manage a server and routes.

NPM is a popular Node.js package library and the jewel in the crown of the Node.js community. It has millions of downloadable libraries, organized according to specific requirements, and is the largest software registry in the world. NPM is free. These libraries are growing fast to this very day, and they strengthen the Node.js community.

Node.js has attracted the attention of businesses and organizations from all sectors. This is hardly a surprise, considering its versatility and strong community support. With adopters such as Netflix, Paypal, and other tech companies, Node.js has seen an exponential increase in its use in web development.

Fig 2.1 illustrates the nodejs code in the visual studio environment. It also consists of express framework which is used with routers.



```
const express = require('express')
const stauth = require('../../middlewares/stauth')
const validate = require('../../middlewares/validate')
const stationValidation = require('../../validations/station.validation')
const stationController = require('../../controllers/station.controller')

const router = express.Router()

router.post('/add-station', stauth, validate(stationValidation.createStation), stationController.createStation)

router.get('/searchStations', stauth, validate(stationValidation.searchStations), stationController.searchStations)

router.get('/filterStations', stauth, validate(stationValidation.filterStations), stationController.filterStations)

router.get('/getStation/:stationId', stauth, validate(stationValidation.getStation), stationController.getStation)

router.delete('/deleteStation/:stationId', stauth, validate(stationValidation.deleteStation), stationController.deleteStation)

router.patch('/updateStation/:stationId', stauth, validate(stationValidation.updateStation), stationController.updateStation)

router.get('/nearStation', stauth, validate(stationValidation.nearStation), stationController.nearStation)

router.get('/findStations', stauth, validate(stationValidation.findStations), stationController.findStations)

module.exports = router

```

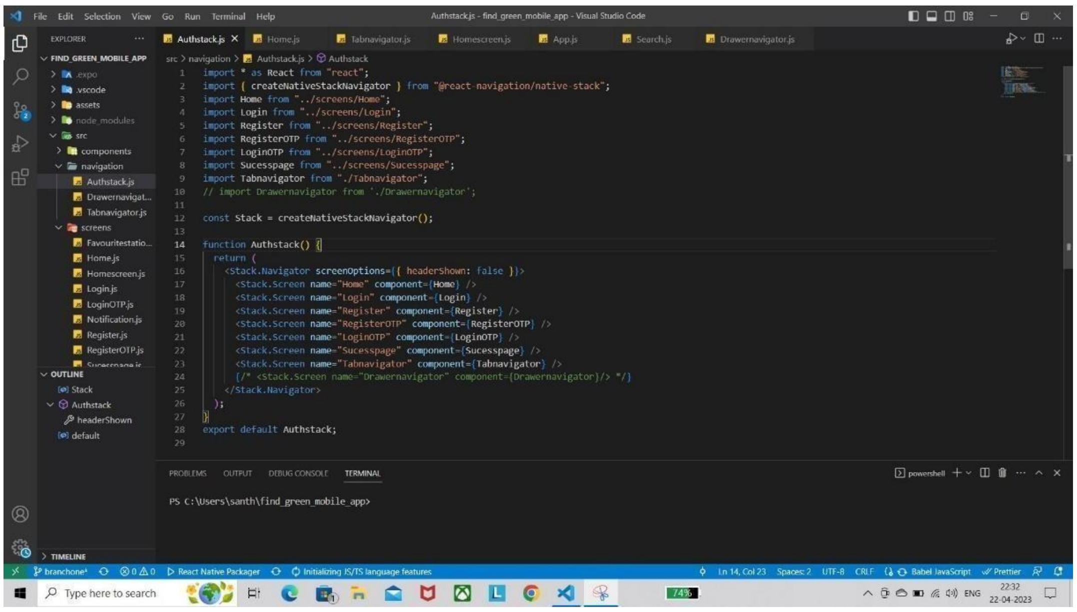
```
PS C:\Users\Rose\Downloads\find_green_server-master\find_green_server-master> 
```

Fig 2.1 NodeJs Code

2.4.2 React Native

React Native is an open-source JavaScript framework, designed for building apps on multiple platforms like iOS, Android, and also web applications, utilizing the very same code base. It is based on React, and it brings all its glory to mobile app development. React Native uses JavaScript to compile the app's user interface, but using native-OS views. For more complex features, it allows code implementation in OS-native languages (Swift and Objective-C for iOS, and Java and Kotlin for Android).

Starting from version 0.70, the default engine is set as Hermes, resulting in faster application start time and a smaller APK (Android) file size. While the IPA (iOS) file size may have increased slightly, there is a memory consumption reduction. Fig 2.2 illustrates the react native code in the visual studio environment.



```
import * as React from "react";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import Home from "../screens/Home";
import Login from "../screens/Login";
import Register from "../screens/Register";
import RegisterOTP from "../screens/RegisterOTP";
import LoginOTP from "../screens/LoginOTP";
import Sucesspage from "../screens/Sucesspage";
import Tabnavigator from "./Tabnavigator";
// import Drawernavigator from './Drawernavigator';

const Stack = createNativeStackNavigator();

function Authstack() {
  return (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Login" component={Login} />
      <Stack.Screen name="Register" component={Register} />
      <Stack.Screen name="RegisterOTP" component={RegisterOTP} />
      <Stack.Screen name="LoginOTP" component={LoginOTP} />
      <Stack.Screen name="Sucesspage" component={Sucesspage} />
      <Stack.Screen name="Tabnavigator" component={Tabnavigator} />
      {/* <Stack.Screen name="Drawernavigator" component={Drawernavigator}/> */}
    </Stack.Navigator>
  );
}
export default Authstack;
```

```
PS C:\Users\santh\find_green_mobile_app>
```

Fig 2.2 React Native Code

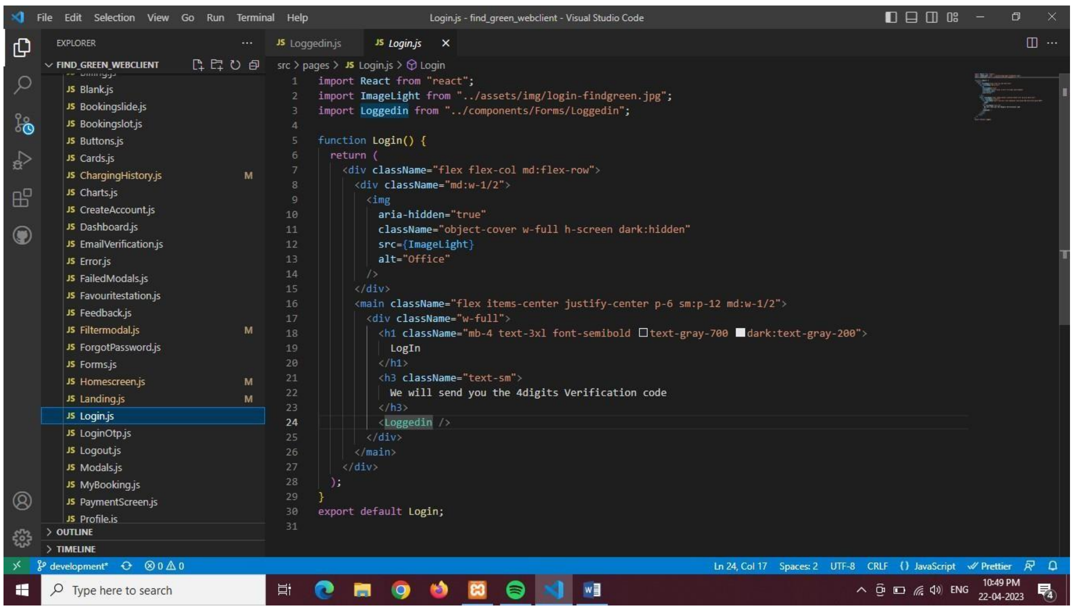
2.4.3 React JS

React is a popular JavaScript library used for web development. React.js or ReactJS or React are different ways to represent ReactJS. Today's many large-scale companies (Netflix, Instagram, to name a few) also use React JS.

React.js is a front-end JavaScript framework developed by Facebook. To build composable user interfaces predictably and efficiently using declarative code, we use React. It's an open-source and component-based framework responsible for creating the application's view layer.

ReactJS divides the UI into multiple components, making the code easier to debug. This way, each function is assigned to a specific component, and it produces some HTML which is rendered as output by the DOM.

Fig 2.3 illustrates the reactjs code developed in the visual studio environment.



```
import React from "react";
import ImageLight from "../assets/img/login-findgreen.jpg";
import Loggedin from "../components/Forms/Loggedin";

function Login() {
  return (
    <div className="flex flex-col md:flex-row">
      <div className="md:w-1/2">
        <img
          aria-hidden="true"
          className="object-cover w-full h-screen dark:hidden"
          src={ImageLight}
          alt="Office"
        />
      </div>
      <main className="flex items-center justify-center p-6 sm:p-12 md:w-1/2">
        <div className="w-full">
          <h1 className="mb-4 text-3xl font-semibold text-gray-700 dark:text-gray-200">
            LogIn
          </h1>
          <h3 className="text-sm">
            We will send you the 4digits Verification code
          </h3>
          <Loggedin />
        </div>
      </main>
    </div>
  );
}
export default Login;
```

Fig 2.3 ReactJs Code

2.5 Software Specification

2.5.1 Operating System

Windows 10 is a personal computer operating system developed by Microsoft as part of Windows NT family of operating systems. Development of 10 occurred as early as 2006 under the code name "Blackcomb".

2.5.2 Android Studio

Android Studio is the official integrated development environment (IDE) for Android application development. It is based on IntelliJ IDEA, a Java integrated development environment for software, and incorporates its code editing and developer tools.

To support application development within the Android operating system, Android Studio uses a Gradle-based build system, Android Emulator, code templates and GitHub integration. Every project in Android Studio has one or more modalities with source code and resource files. These modalities include Android app modules, Library modules and Google App Engine modules.

Android Studio uses an Apply Changes feature to push code and resource changes to a running application. A code editor assists the developer with writing code and offering code completion, refraction and analysis. Applications built in Android Studio are then compiled into the APK format for submission to the Google Play Store.

Android Studio is available for macOS, Windows and Linux desktop platforms. It replaced Eclipse Android Development Tools (ADT) as the primary IDE for Android application development. It is used to develop applications using react native in this project.

2.5.3 PostgreSQL

PostgreSQL is one of the most advanced general-purpose object-relational database management system and is open-source. Being an open-source software, its source code is available under PostgreSQL license, a liberal open source license. Anyone with the right skills is free to use, modify, and distribute PostgreSQL in any form. As it is highly stable, very low effort is required to maintain this DBMS.

The POSTGRES project aimed at adding fewest features like the ability to define various data types and to fully describe relationships – something used widely, but maintained completely by the end-user. POSTGRES used various ideas of Ingres, but had its unique source code.

The initial version of PostgreSQL was designed to run on UNIX-like platforms. However, it was then evolved to be mobile so that it could run on other platforms such as Mac OS X, Solaris, and Windows.

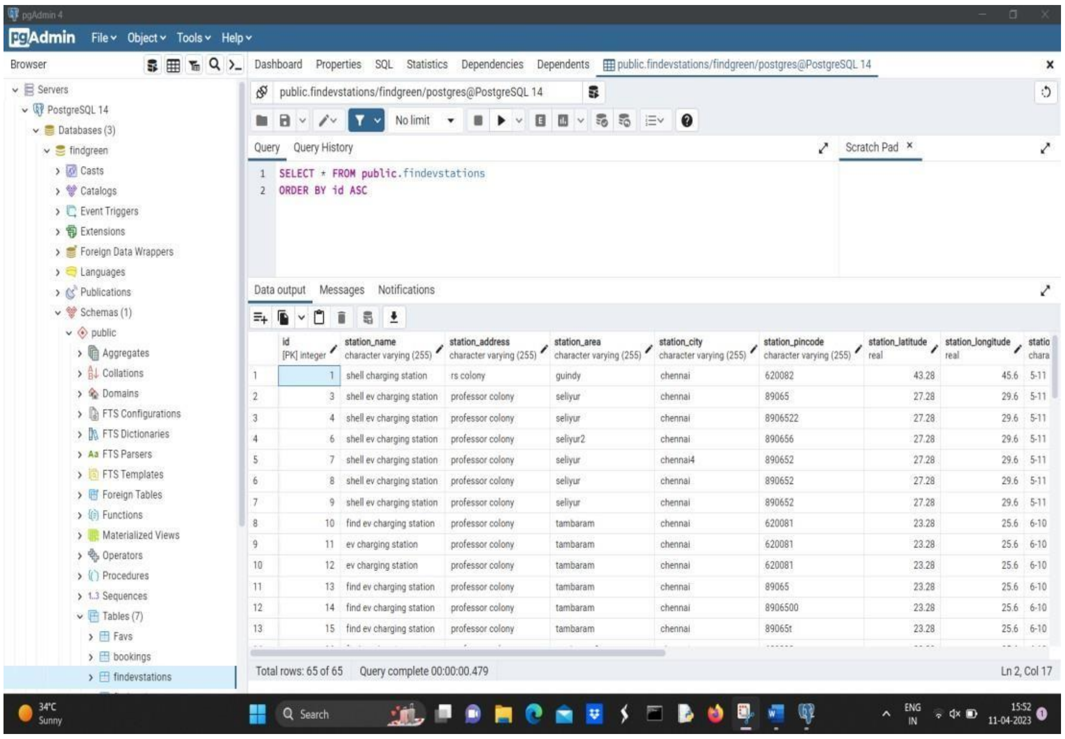
2.5.4 pgAdmin

PgAdmin is considered the most advanced open-source GUI tool designed for the most advanced relational database management tools. In other words, pgAdmin and PostgreSQL together create a powerful (and free) combination for the tech stack.

The developers can use PGAdmin to do any Postgres database administration tasks. Pgadming can be run as an web or desktop application. As the developers put it, pgAdmin 4 is the latest version, and its development involved a complete rewrite of the original pgAdmin tool. This version was created using a combination of Javascript/jQuery and Python. Now, the users can use pgAdmin as a desktop runtime or on a web application server, depending on your use case. No matter how the user run pgAdmin 4, the users will find that it easily accepts one or more users and creates an environment that looks and feels like a desktop application.

While it's possible to accomplish these same things without pgAdmin, the value of this tool is that it provides a user-friendly data administration interface for the users to handle SQL queries, maintenance, and other necessary processes without using command line prompts.

Fig 2.4 illustrates the pgAdmin screen. It contains all the databases, tables etc. Here all the input from the users are stored in the particular tables of the database. The data can be retrieved from the tables of the database. The developers can add, update, delete and view the data in the database. Here it illustrates show the charging station details of the project.



```
SELECT * FROM public.findevstations
ORDER BY id ASC
```

	id [PK] integer	station_name character varying (255)	station_address character varying (255)	station_area character varying (255)	station_city character varying (255)	station_pincode character varying (255)	station_latitude real	station_longitude real	statio chara
1	1	shell charging station	rs colony	guindy	chennai	620082	43.28	45.6	5-11
2	3	shell ev charging station	professor colony	seliyur	chennai	89065	27.28	29.6	5-11
3	4	shell ev charging station	professor colony	seliyur	chennai	8906522	27.28	29.6	5-11
4	6	shell ev charging station	professor colony	seliyur2	chennai	890656	27.28	29.6	5-11
5	7	shell ev charging station	professor colony	seliyur	chennai4	890652	27.28	29.6	5-11
6	8	shell ev charging station	professor colony	seliyur	chennai	890652	27.28	29.6	5-11
7	9	shell ev charging station	professor colony	seliyur	chennai	890652	27.28	29.6	5-11
8	10	find ev charging station	professor colony	tambaram	chennai	620081	23.28	25.6	6-10
9	11	ev charging station	professor colony	tambaram	chennai	620081	23.28	25.6	6-10
10	12	ev charging station	professor colony	tambaram	chennai	620081	23.28	25.6	6-10
11	13	find ev charging station	professor colony	tambaram	chennai	89065	23.28	25.6	6-10
12	14	find ev charging station	professor colony	tambaram	chennai	8906500	23.28	25.6	6-10
13	15	find ev charging station	professor colony	tambaram	chennai	89065t	23.28	25.6	6-10

Total rows: 65 of 65 Query complete 00:00:00.479 Ln 2, Col 17

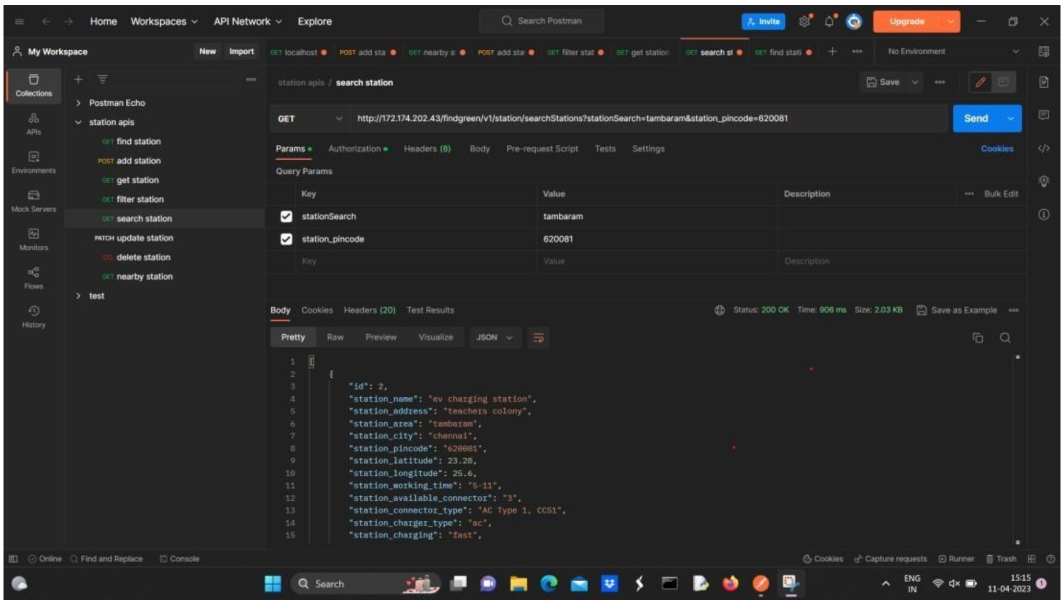
Fig 2.4 pgAdmin screen

2.5.5 Postman

Postman is one of the most popular software testing tools which is used for API testing. With the help of this tool, developers can easily create, test, share, and document APIs.

Postman is one of the most popular software testing tools which is used for API testing. With the help of this tool, developers can easily create, test, share, and document APIs. This tool has the ability to make various types of HTTP requests like GET, POST, PUT, PATCH, and convert the API to code for languages like JavaScript, Nodejs, Python etc.

Fig 2.5 illustrates the postman screen. Here the developers can create the APIs and pass the values of the input into the database. The developer the use the methods like POST to add data into the database, GET to get the values from the database, DEL to delete the data in database, PATCH to update the data in database etc.



GET http://172.174.202.43/findgreen/v1/station/searchStations?stationSearch=tambaram&station_pincode=620081

	Key	Value	Description
✓	stationSearch	tambaram	
✓	station_pincode	620081	

Status: 200 OK Time: 906 ms Size: 2.03 KB

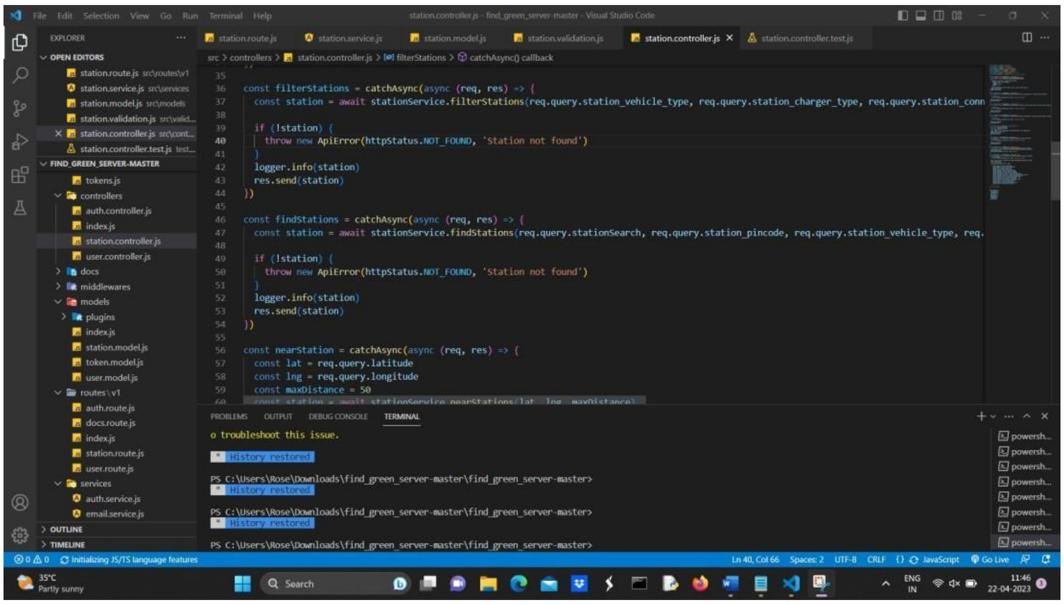
```
[
    {
        "id": 2,
        "station_name": "ev charging station",
        "station_address": "teachers colony",
        "station_area": "tambaram",
        "station_city": "chennai",
        "station_pincode": "620081",
        "station_latitude": 23.28,
        "station_longitude": 25.6,
        "station_working_time": "5-11",
        "station_available_connector": "3",
        "station_connector_type": "AC Type 1, CCS1",
        "station_charger_type": "ac",
        "station_charging": "fast",
```

Fig 2.5 Postman Screen

2.5.6 Visual Studio Code

Visual Studio Code is a code editor in layman's terms. Visual Studio Code is "a free-editor that helps the programmer write code, helps in debugging and corrects the code using the intelli-sense method. In normal terms, it facilitates users to write the code in an easy manner. Many people say that it is half of an IDE and an editor, but the decision is up to the coders. Any program/software that the developers see or use works on the code that runs in the background. Traditionally coding was used to do in the traditional editors or even in the basic editors like notepad! These editors used to provide basic support to the coders.

VI Editor, Sublime Text Editor, is one of the many kinds of editors that came into existence. The most prominent and which supports almost every coding language is VISUAL STUDIO CODE. Its features let the user modify the editor as per the usage, which means the user is able to download the libraries from the internet and integrate it with the code as per his requirements.



```
const filterStations = catchAsync(async (req, res) => {
  const station = await stationService.filterStations(req.query.station_vehicle_type, req.query.station_charger_type, req.query.station_conn

  if (!station) {
    throw new ApiError(httpStatus.NOT_FOUND, 'Station not found')
  }
  logger.info(station)
  res.send(station)
})

const findStations = catchAsync(async (req, res) => {
  const station = await stationService.findStations(req.query.stationSearch, req.query.station_pincode, req.query.station_vehicle_type, req.

  if (!station) {
    throw new ApiError(httpStatus.NOT_FOUND, 'Station not found')
  }
  logger.info(station)
  res.send(station)
})

const nearStation = catchAsync(async (req, res) => {
  const lat = req.query.latitude
  const lng = req.query.longitude
  const maxDistance = 50
```

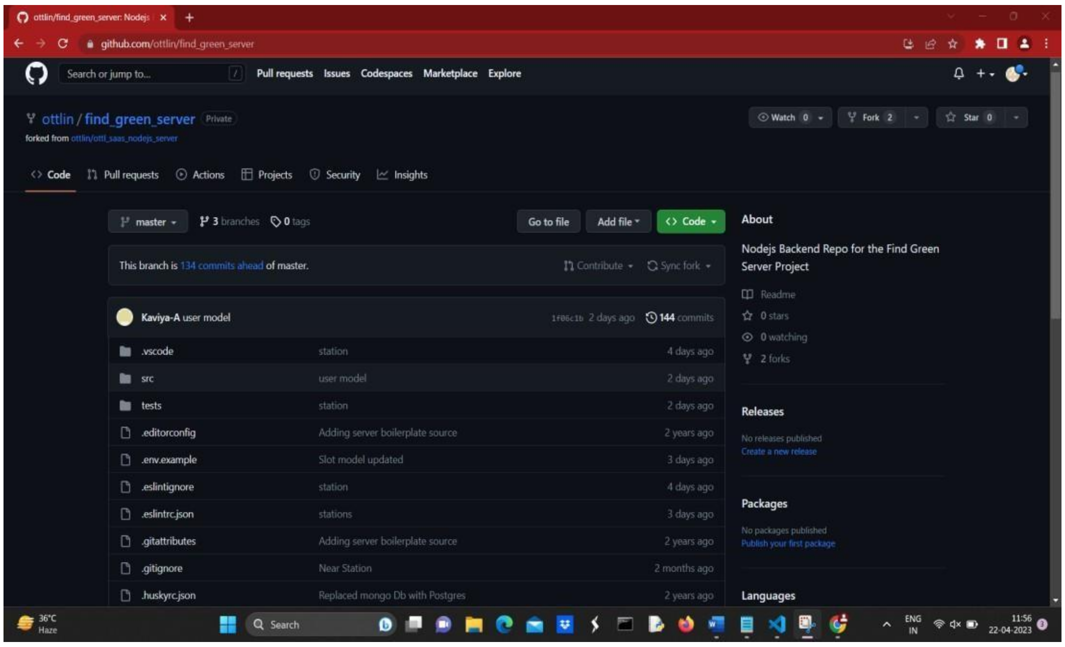
```
o troubleshoot this issue.

 *  History restored

PS C:\Users\Rose\Downloads\find_green_server-master\find_green_server-master>
 *  History restored

PS C:\Users\Rose\Downloads\find_green_server-master\find_green_server-master>
 *  History restored

PS C:\Users\Rose\Downloads\find_green_server-master\find_green_server-master>
```

Fig 2.6 Visual Studio Code Screen

2.5.7 GitHub

GitHub is an online software development platform. It's used for storing, tracking, and collaborating on software projects. It makes it easy for developers to share code files and collaborate with fellow developers on open-source projects. GitHub also serves as a social networking site where developers can openly network, collaborate, and pitch their work. This free service comes with several helpful features for sharing code and working with others in real time.

While anyone can code independently, teams of people build most development projects. Sometimes these teams are all in one place at once time, but more often they work asynchronously. There are many challenges to creating collaborative projects with distributed teams. The company creates a GitHub link in which all the developers push their code and pull it anytime.



ottlin / find_green_server Private
forked from ottlin/ottl_saas_nodejs_server

3 branches 0 tags

This branch is 134 commits ahead of master.

Kaviya-A user model 1f06c1b 2 days ago 144 commits

.vscode	station	4 days ago
src	user model	2 days ago
tests	station	2 days ago
.editorconfig	Adding server boilerplate source	2 years ago
.env.example	Slot model updated	3 days ago
.eslintignore	station	4 days ago
.eslintrc.json	stations	3 days ago
.gitattributes	Adding server boilerplate source	2 years ago
.gitignore	Near Station	2 months ago
.huskyrc.json	Replaced mongo Db with Postgres	2 years ago

About

Nodejs Backend Repo for the Find Green Server Project

Readme
0 stars
0 watching
2 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

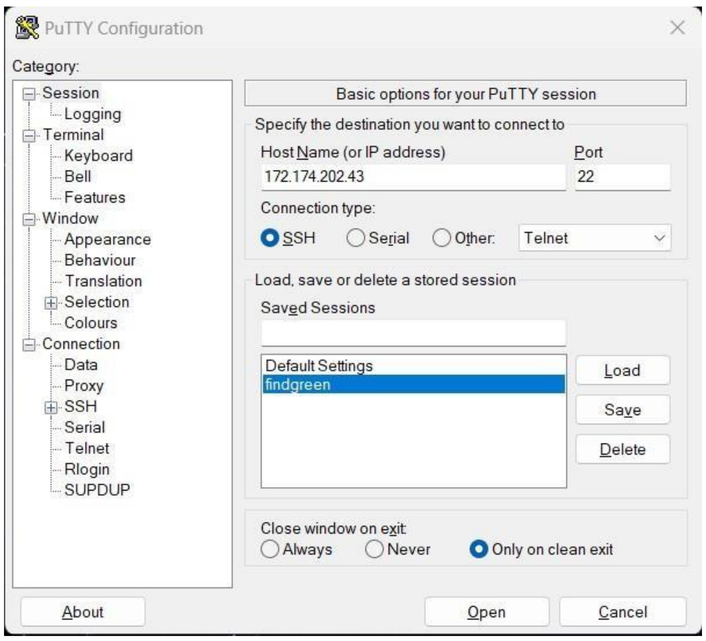
Fig 2.7 GitHub Screen

2.5.8 Putty

PuTTY is an open-source application making use of network protocols like Telnet and rlogin in Windows and UNIX platforms in conjunction with an xterm terminal emulator. It is a popular utility for connecting Linux servers from Microsoft operating system-based computers.

It even specifies its terminal type as xterm to the server; although this can be reconfigured. Most features like port forwarding and public keys are available through the command line options. The main window of PuTTY has the session which runs on the remote computer and through which one can send the commands directly to the remote computer. With regards to cut and paste features, PuTTY can be customized to act similarly to xterm.

It is easier to configure and is more stable. It is also more persistent in comparison to others, as a remote session can be resumed as soon the connection is restored after an interruption. It has an easy-to-use graphical user interface. Some terminal control sequences like the Linux console sequences which are unsupported by xterm are supported by PuTTY.



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 172.174.202.43

Port: 22

Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

Saved Sessions

Default Settings

findgreen

Close window on exit: Always Never [✓ Only on clean exit]

Fig 2.8 Putty Screen

3 SYSTEM DESIGN

3.1 Workflow Diagram

The user can click the login button to go to the login page if they already have an account or else they can click the register button which will take the user to the registration page.

The user can register by entering their first and last name, email id and mobile number and click ‘Create Account’ button. It will take the user to the OTP page. After clicking the ‘Create Account’ button in the registration page, it gets directed to this otp page. The user gets opt to their mobile number and email id.

They should enter the correct opt to confirm their registration. Once registered, the user can login and gets directed to the Homepage.

The user can filter the stations based on the charger type, availability, vehicle type, distance, connector type, favourite station, fast changing etc or search the station based on the address, name, area, city, pincode etc. Once Apply button is clicked, it list the station details based on the filters applied. The station details includes the station name, place, city, working hours, status, available connectors, connector types, rate etc. Here the user can add the station as favourite station or share the station to others or navigate to the station. The stations will be viewed either in the list format or map view.

Once the user selects a station in the Station page it displays the station name, place, working hours, connector type, charger type, capacity, tariff, amenities, date and time pickers etc. Here the user can book a slot by selecting the date, time and duration. Once chosen, the user click proceed, which will take the user to the payment page.

It will display the amount to be paid and the methods of payment. Here the user should select the payment method and proceed with payment. If the payment is done successfully and if the slot is free then the slot gets booked and shows success message. If the payment is not successful or if the slot is not free then the slot doesn’t get booked and it shows the booking failure page.

Once the booking and payment is completed successfully, it will take the user to the homepage. This is the workflow of the project. Fig 3.1 illustrates the workflow diagram of the project.

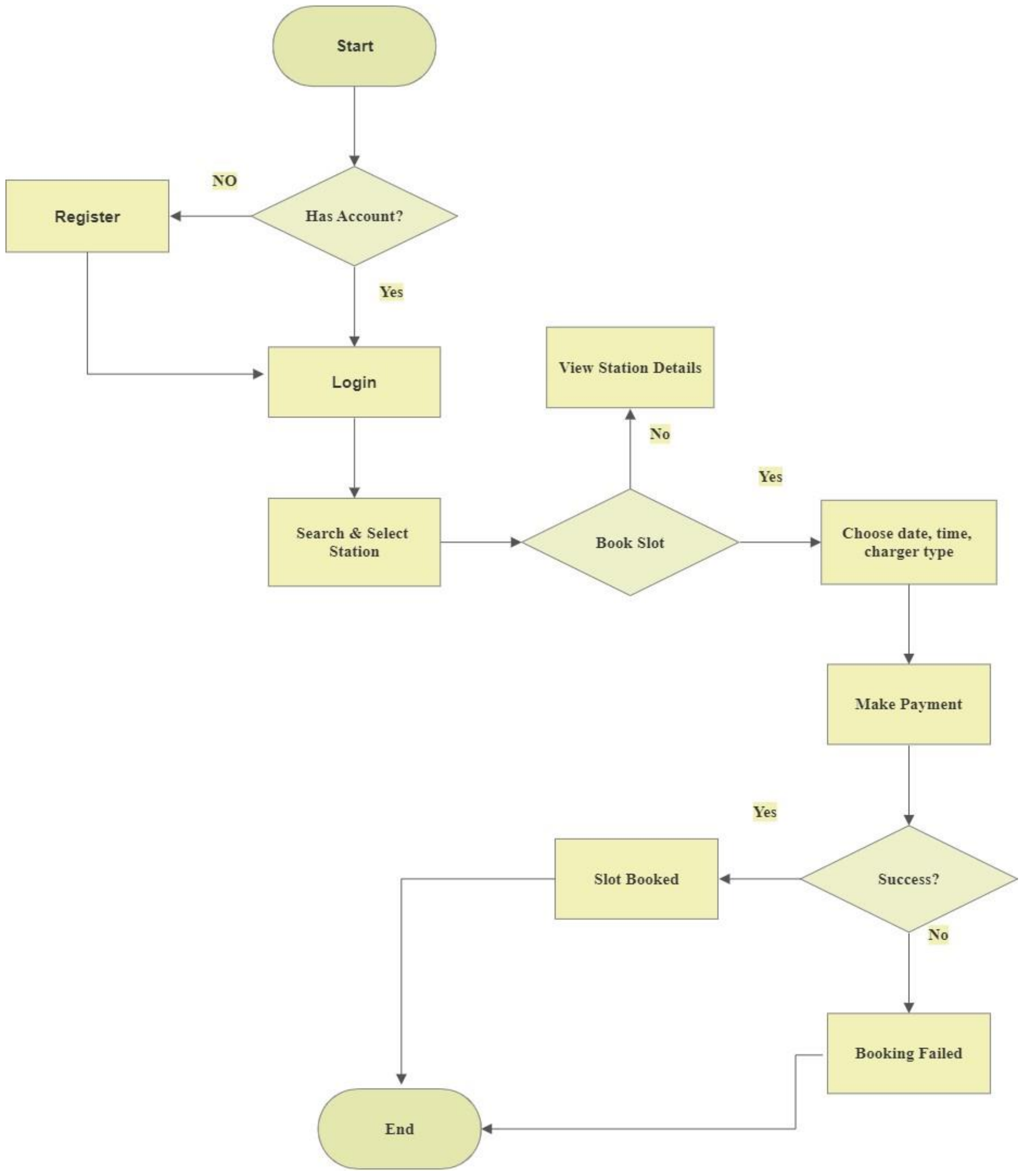


Fig 3.1 Project Workflow

3.2 Table Design

3.2.1 User Table

Table 3.1 gives the user_id (primary key), name, email, phone_no, country,postal_code, vehicle_model, vehicle_no, connector_type, profile_pic and status. The values which the users enter in the Registration page like first and last name, email id, mobile number and in My Profile Page like vehicle type, country, connector type, profile picture is stored in this table.

Table 3.1 User Table

Field	Data Type	Constraints
user_id	int	Primary Key
Name	varchar(50)	Not Null
Email	varchar(50)	Not Null
phone_no	Bigint	Not Null
Country	varchar(50)	Not Null
postal_code	Int	Not Null
vehicle_model	varchar(50)	Not Null
vehicle_no	Int	Not Null
connector_type	varchar(50)	Not Null
Profile_pic	varchar(50)	Not Null
Status	varchar(50)	Not Null

3.2.2 Station Table

Table 3.2 gives the station_id (primary key), station_name, latitude, longitude, address, area, city, working_time, connector_type, avaliable_connector, capacity and amenities. The station page display the station details from this table based on the filters applied. This table is also linked to Booking page and Favourite Page and tables.

Table 3.2 Station Table

Field	Data Type	Constraints

station_id	int	Primary Key
station_name	varchar(50)	Not Null
Latitude	Float	Not Null
Longitude	Float	Not Null
Country	varchar(50)	Not Null
Address	varchar(50)	Not Null
Area	varchar(50)	Not Null
City	varchar(50)	Not Null
working_time	varchar(50)	Not Null
connector_type	varchar(50)	Not Null
avaliable_connector	varchar(50)	Not Null
Capacity	Int	Not Null
Amenities	varchar(50)	Not Null

3.2.3 Booking Table

Table 3.3 gives the booking_id (primary key),user_id (foreign key), station_id (foreign key), date, time, duration, connector_type, capacity, payment and status. The values from the Booking page are stored in this table. This table is also linked with the user table and station table.

Table 3.3 Booking Table

Field	Data Type	Constraints
booking _id	int	Primary Key
user_id	int	Foreign Key
station_id	int	Foreign Key

Date	Date	Not Null
Time	Time	Not Null
Duration	Int	Not Null
connector_type	varchar(50)	Not Null
Payment	varchar(50)	Not Null
Capacity	Int	Not Null
Status	varchar(50)	Not Null

3.2.4 Favourites Table

Table 3.4 gives the fave_id (primary key),user_id (foreign key) and station_id (foreign key). This table is used to display the favourite stations of a particular user in the Favourites page. So this table is linked to station table and user table.

Table 3.4 Favourites table

Field	Data Type	Constraints
fave _id	int	Primary Key
user_id	int	Foreign Key
station_id	int	Foreign Key

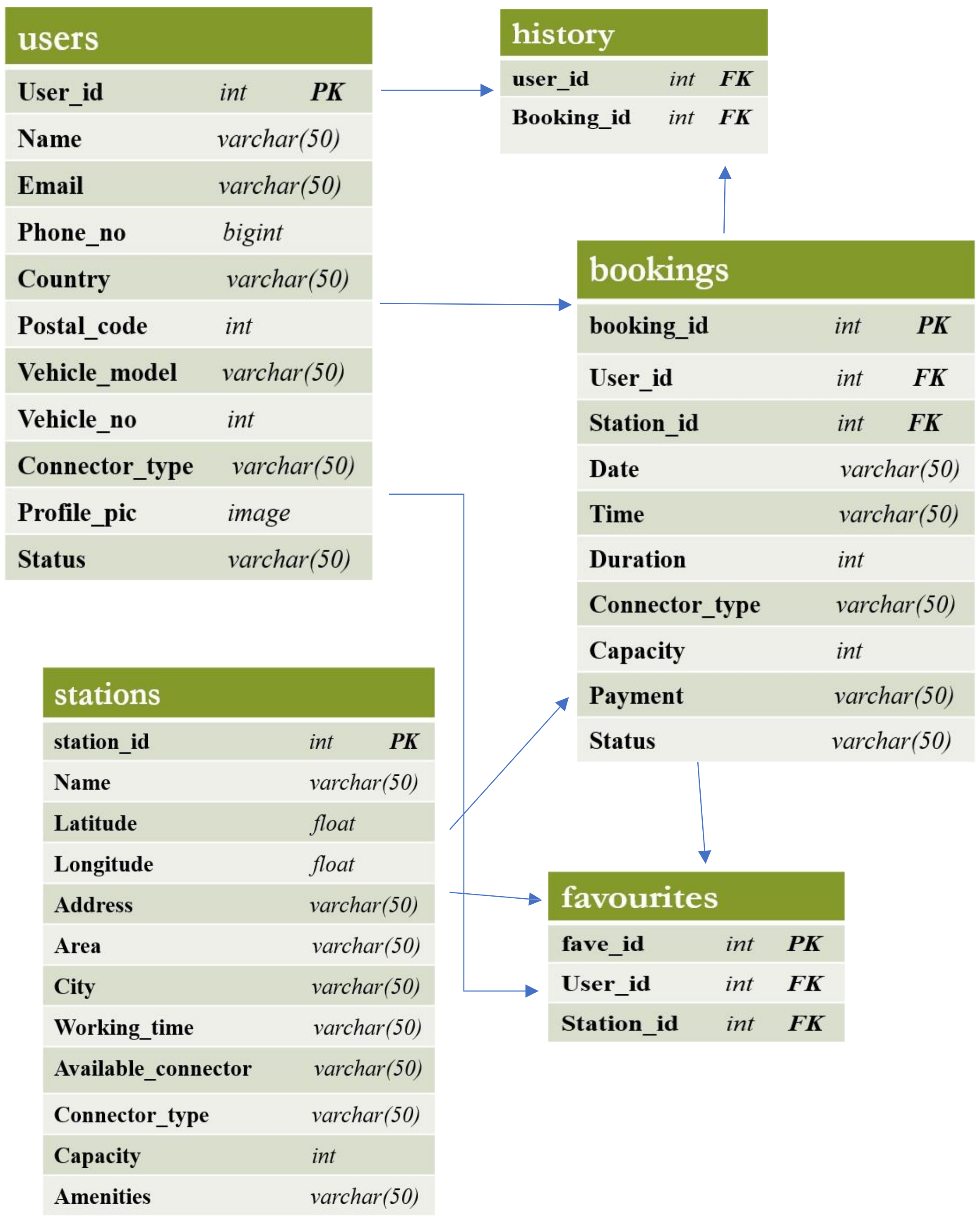
3.2.5 History Table

Table 3.5 gives the user_id (foreign key) and booking_id (foreign key). This table is used to display the Charging history of a particular person. This table is linked with the user table and booking table. It gets the value from these values and displays it.

Table 3.5 History table

Field	Data Type	Constraints
user_id	int	Foreign Key
booking _id	int	Foreign Key

3.3 Table Schema



users

User_id	*int*	*PK*
Name	*varchar(50)*	
Email	*varchar(50)*	
Phone_no	*bigint*	
Country	*varchar(50)*	
Postal_code	*int*	
Vehicle_model	*varchar(50)*	
Vehicle_no	*int*	
Connector_type	*varchar(50)*	
Profile_pic	*image*	
Status	*varchar(50)*	

history

user_id	*int*	*FK*
Booking_id	*int*	*FK*

bookings

booking_id	*int*	*PK*
User_id	*int*	*FK*
Station_id	*int*	*FK*
Date	*varchar(50)*	
Time	*varchar(50)*	
Duration	*int*	
Connector_type	*varchar(50)*	
Capacity	*int*	
Payment	*varchar(50)*	
Status	*varchar(50)*	

stations

station_id	*int*	*PK*
Name	*varchar(50)*	
Latitude	*float*	
Longitude	*float*	
Address	*varchar(50)*	
Area	*varchar(50)*	
City	*varchar(50)*	
Working_time	*varchar(50)*	
Available_connector	*varchar(50)*	
Connector_type	*varchar(50)*	
Capacity	*int*	
Amenities	*varchar(50)*	

favourites

fave_id	*int*	*PK*
User_id	*int*	*FK*
Station_id	*int*	*FK*

Fig 3.2 Table Design

4 MODULE DESIGN

4.1 Registration

The user can click the login button to go to the login page if they already have an account or else they can click the register button which will take the user to the registration page.

The user can register by entering their first and last name, email id and mobile number and click create account button. It will take the user to the OTP page. After clicking Create Account button in the registration page, it gets directed to this otp page. The user gets opt to their mobile number and email id. They should enter the correct opt to confirm their registration. Once registered, the user can login and gets directed to the Homepage. Fig 4.1 illustrates the flowchat of the registration.

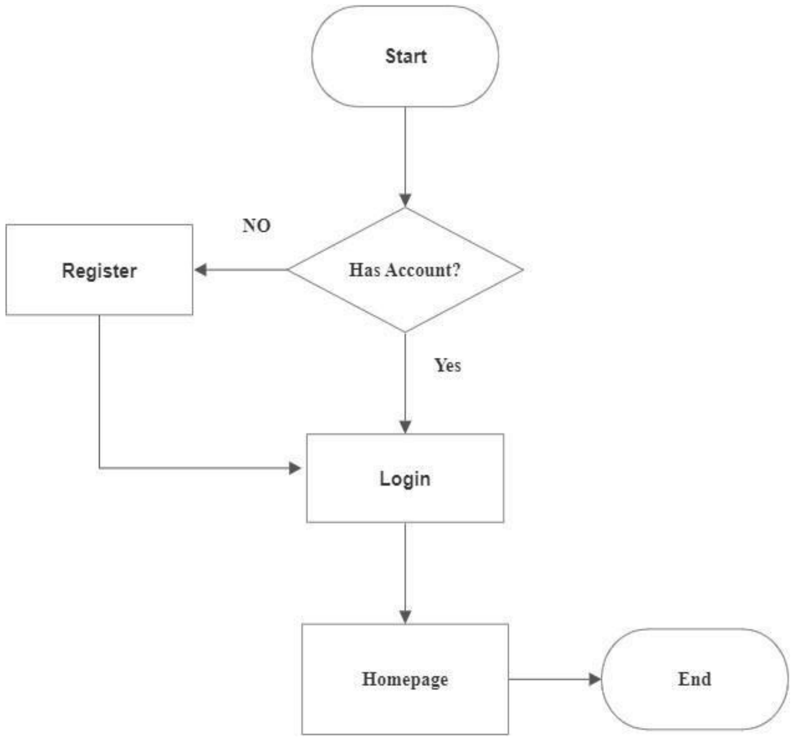


Fig 4.1 Registration flowchart

4.2 Search and Filter

The user can filter the stations based on the charger type, availability, vehicle type, distance, connector type, favourite station, fast changing etc or search the station based on the address, name, area, city, pincode etc. Once Apply button is clicked, it list the station details based on the filters applied. The station details includes the station name, place, city, working hours, status, available connectors, connector types, rate etc. Here the can add the station as their favourite station or share the station to others or navigate to the station. The stations will be viewed either in list format or map view. Fig 4.2 the flowchat of the search and filter.

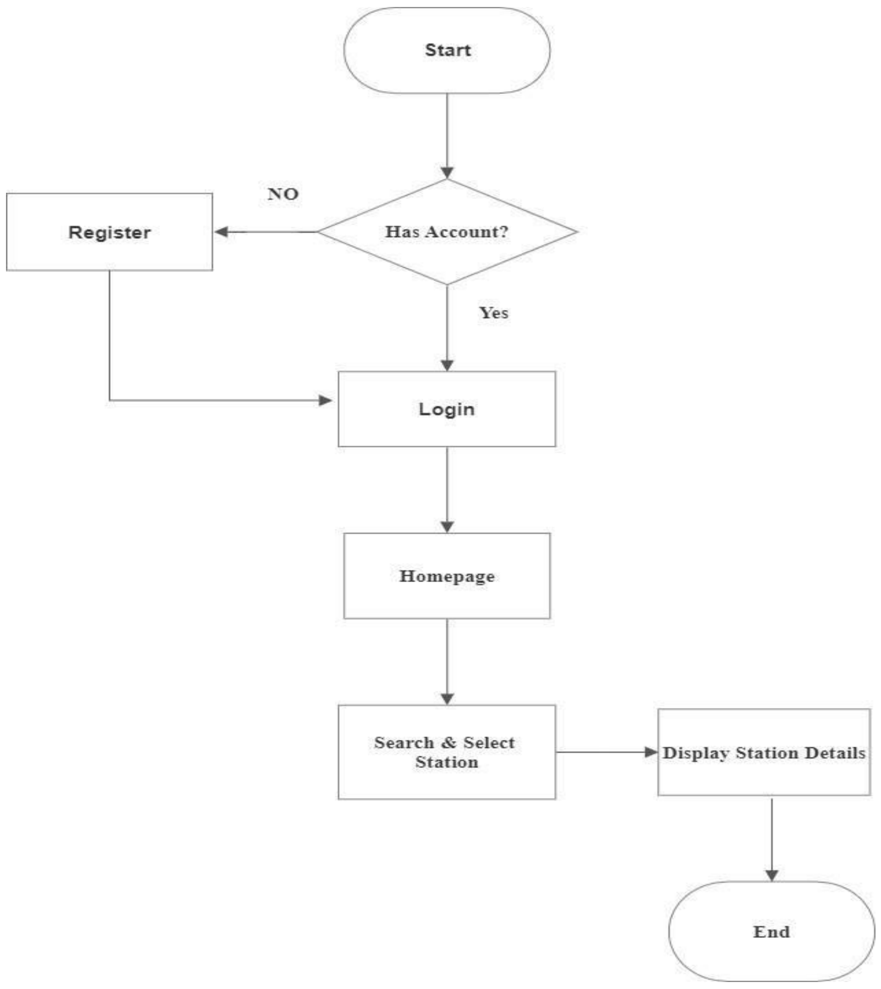


Fig 4.2 Search and Filter Flowchat

4.3 Slot Booking

The user can click the login button to go to the login page if they already have an account or else they can click the register button which will take the user to the registration page. Once registered, the user can login and gets directed to the Homepage.

The user can filter the stations. Once Apply button is clicked, it list the station details based on the filters applied. Once the user selects a station in the Station page it displays the station details. Here the user can book a slot by selecting the date, time and duration. Once chosen, the user click proceed, which will take the user to the payment page.

It will display the amount to be paid and the methods of payment. Here the user should select the payment method and proceed with payment. If the payment is done successfully and if the slot is free then the slot gets booked and shows success message. If the payment is not successful or if the slot is not free then the slot doesn’t get booked and it shows the booking failure page. Once done it will take the user to the homepage.

Fig 4.3 illustrates the flowchat of the slot booking process in the find green project.

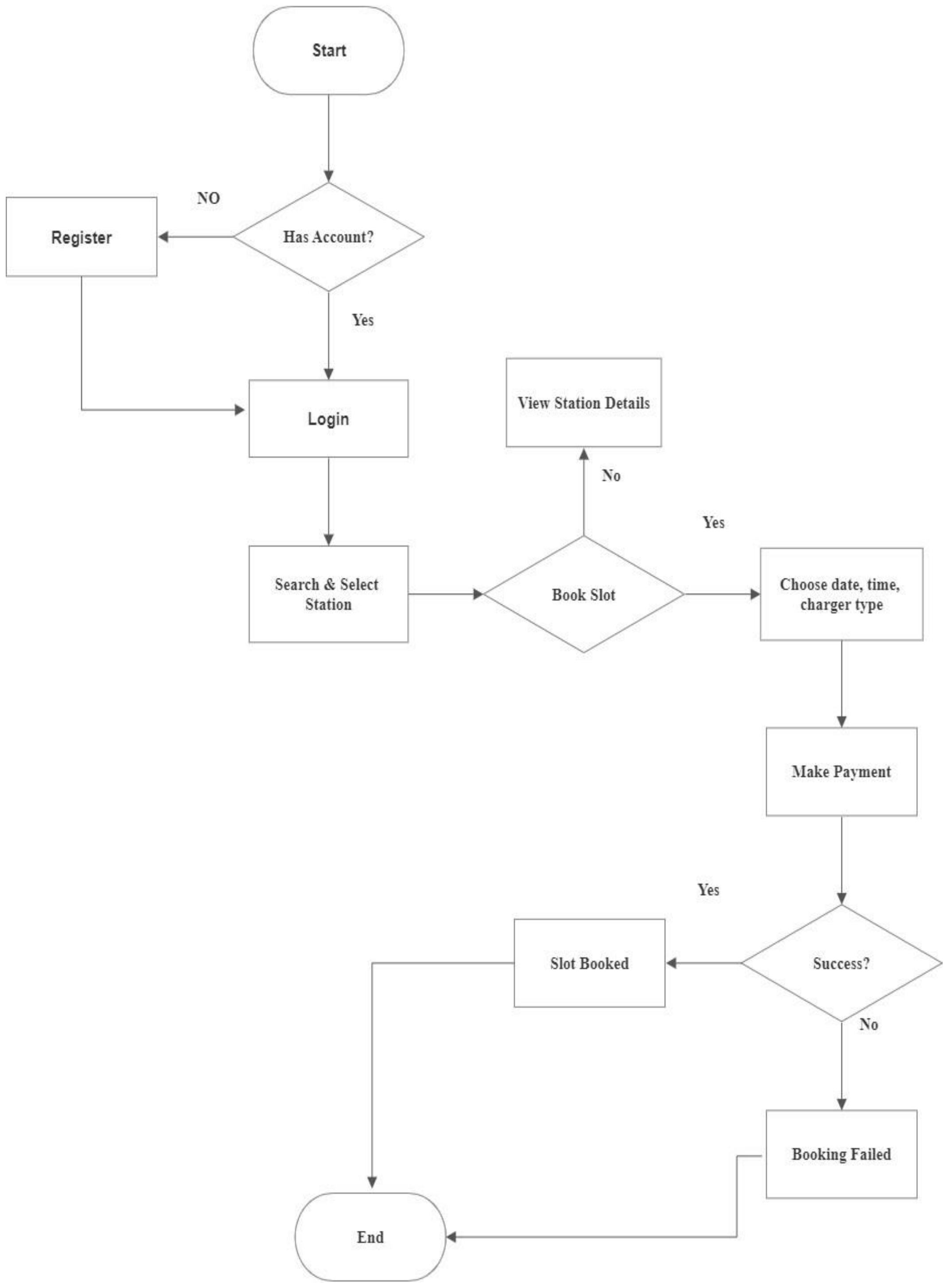


Fig 4.3 Slot Booking Flowchart

4.4 Charging History

It shows the charging history of the user. It shows the details of the completed or cancelled booking of the user. It displays the details like the date and time, station name, user ratings, slot, time, completed or cancelled status etc. It also contains filter option using which the user can view the history based on days, months or years. Fig 4.4 illustrates the flowchat of the charging.

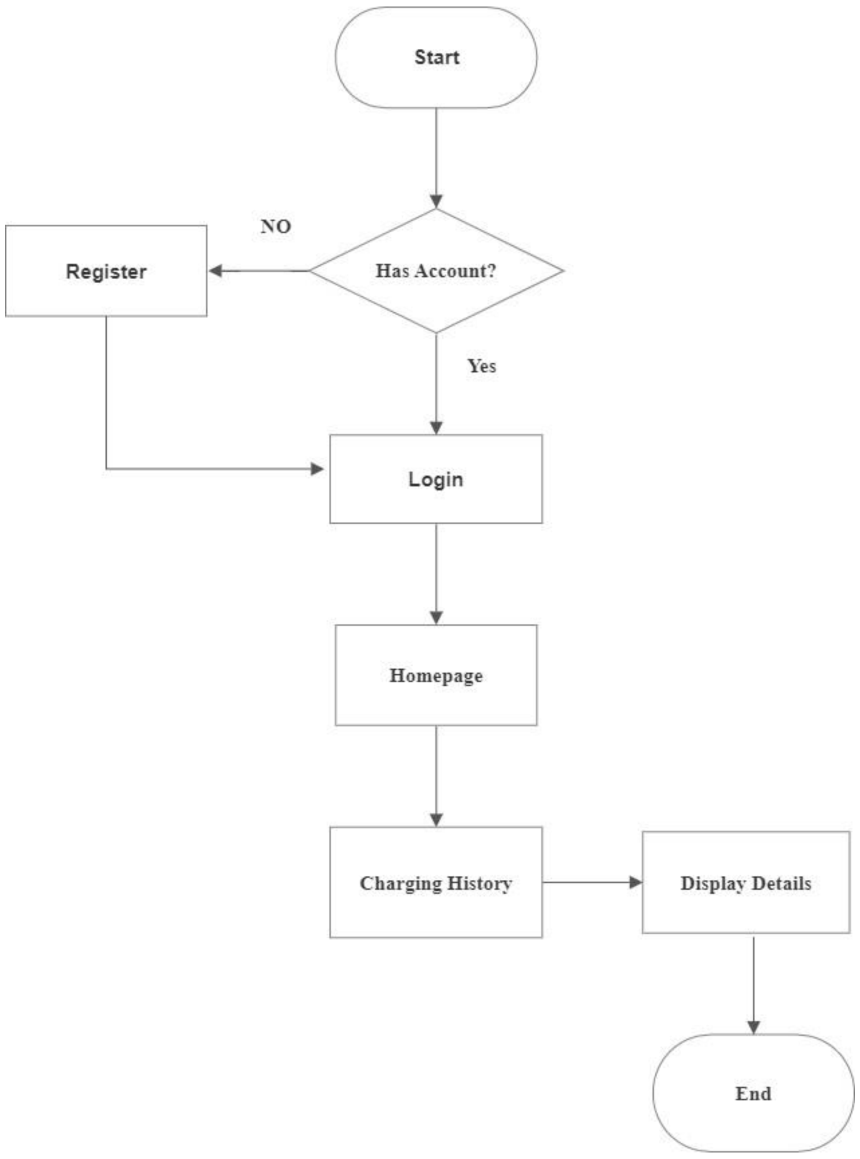


Fig 4.4 Charging History flowchart

4.5 My Bookings

If the user chooses upcoming bookings, it displays the details like the date and time, station name, payment method, slot, time etc.

If the user chooses cancelled bookings, it displays the details like the date and time, station name, payment method, slot, time etc. The user can book the same station by clicking the station name. Fig 4.5 illustrates the flowchat of the my booking.

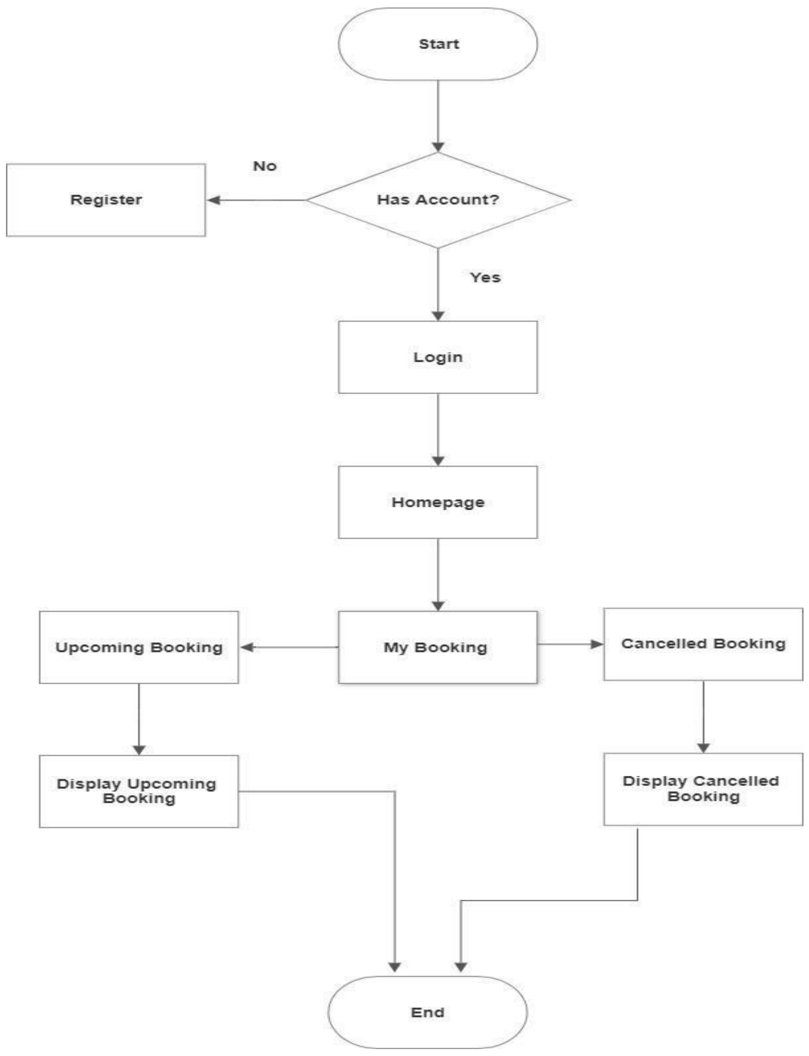


Fig 4.5 My Booking Flowchart

4.6 My Payment

The user can add their payment methods like card details or recharge money to their wallet, add the gpay account etc. The user can also remove the payment methods by clicking the ‘Remove’ button. The users can have options to make payment via app using different payment options like debit/credit card, wallets and others. Fig 4.6 illustrates the flowchat of the my payment.

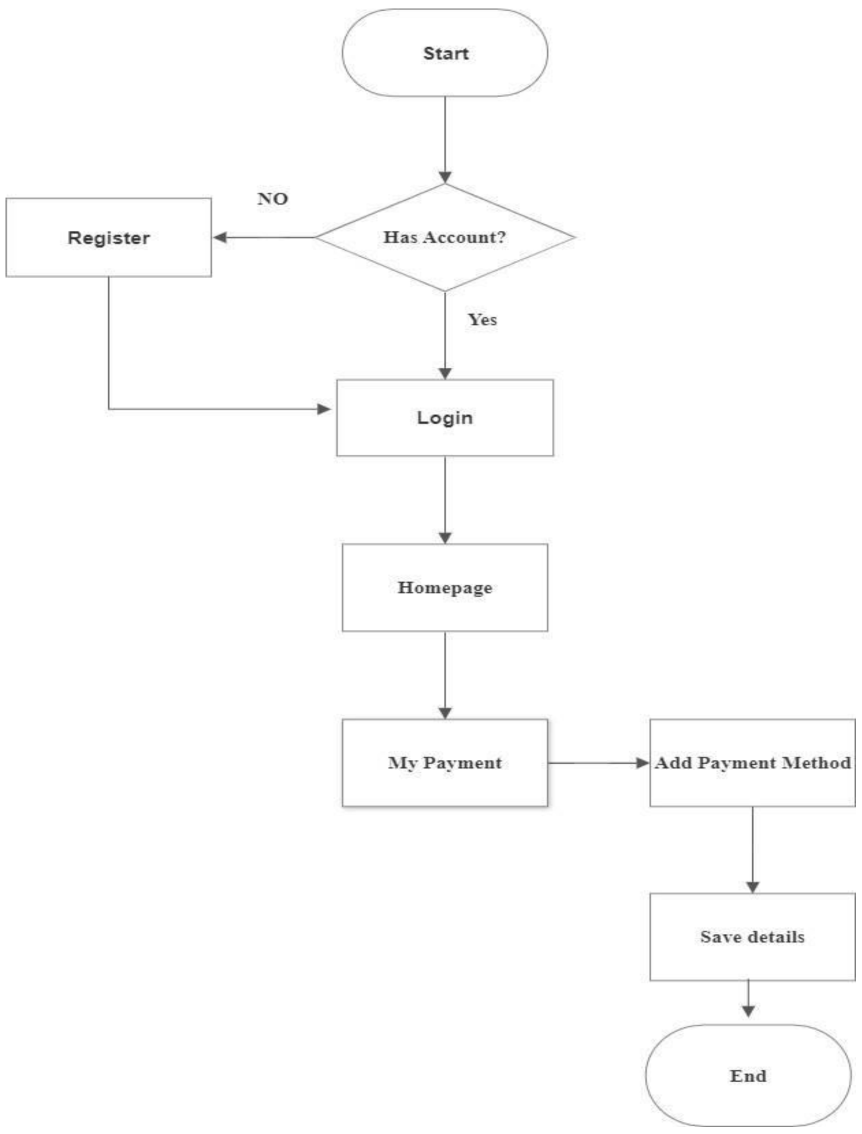


Fig 4.6 My Payment Flowchart

5 USER INTERFACE AND EXPERIENCE

5.1 User Interface

The user interface is the point at which human users interact with a computer, website or application. The goal of effective UI is to make the user's experience easy and intuitive, requiring minimum effort on the user's part to receive the maximum desired outcome.

User interface is important to meet user expectations and support the effective functionality of your site. A well-executed user interface facilitates effective interaction between the user and the program, app or machine through contrasting visuals, clean design and responsiveness.

User interfaces can be visualized in many ways and in many degrees of fidelity. On the web user interfaces are usually rendered as HTML & CSS, and in native applications using native or custom libraries. A design system is often visualised as a collection of UI elements that define a guideline for building a user experience within a given platform.

When designing a UI for the site, it's important to consider the user's expectations in terms of accessibility, visual aesthetic and ease of use. An optimal mix of effective visuals and efficient responsiveness will improve our site's conversion rates, as it anticipates the needs of the user and then satisfies those needs.

In this project, the wireframes created using the whimsical website. All the web design and app design was designed in that site.

5.1.1 Whimsical

Whimsical is a collaborative digital workspace platform that helps design, engineering and product departments facilitate ideation through the creation of project wireframes, flowcharts, mind maps and sticky notes. Multiple stakeholders can collaborate and work on a single document and utilize markdown shortcuts, buttons, checkboxes and inputs to customize user flows.

Designers use Whimsical to create or edit files, grant viewers access to comment on shared files and invite guests to view and collaborate on specific files. Individuals can create a personal or organizational workspace, add new or move the existing board from another workspace, onboard team members and exchange ideas in real-time.

5.1.2 Sample Designs

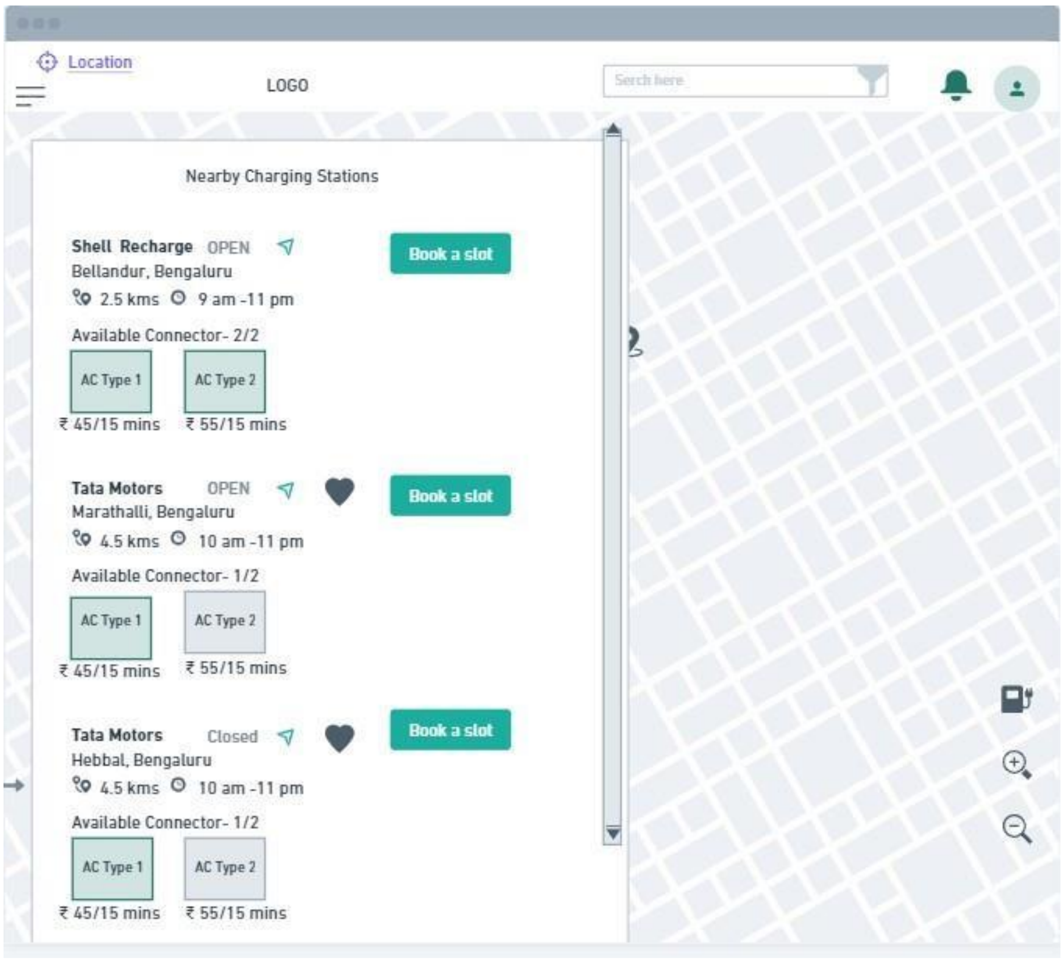


Fig 5.1 User Interface for Web

The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.

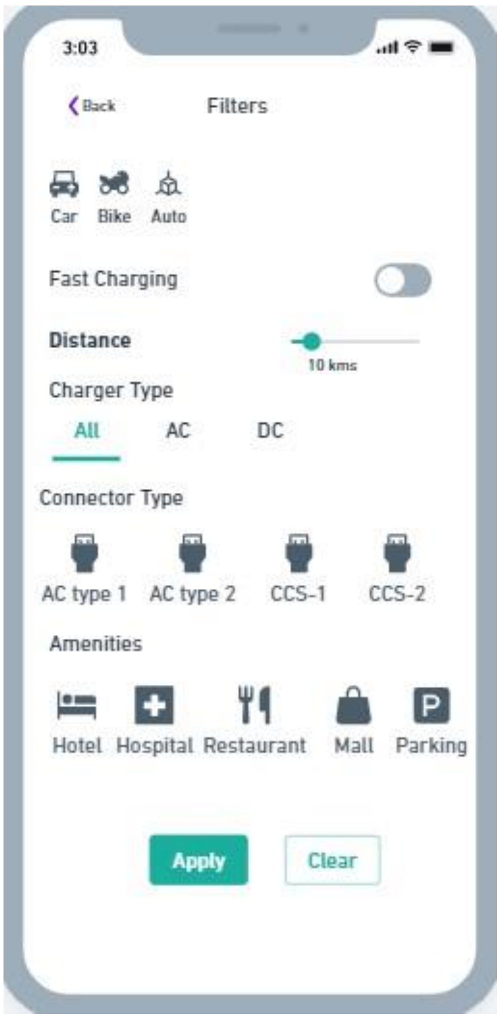


Fig 5.2 User Interface for App

5.2 User Experience

User experience (UX) design is the process design teams use to create products that provide meaningful and relevant experiences to users. UX design involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability and function.

Since UX design encompasses the entire user journey, it's a multidisciplinary field–UX designers come from various backgrounds such as visual design, programming, psychology and interaction design. To design for human users also means working with a heightened scope regarding accessibility and accommodating many potential users' physical limitations, such as reading small text.

A UX designer's typical tasks vary but often include user research, creating personas, designing wireframes and interactive prototypes, and testing designs. These tasks can vary significantly from one organization to the next.

In this project, to design the user experience design for both website and app, the tool called Adobe XD was used.

5.2.1 Adobe XD

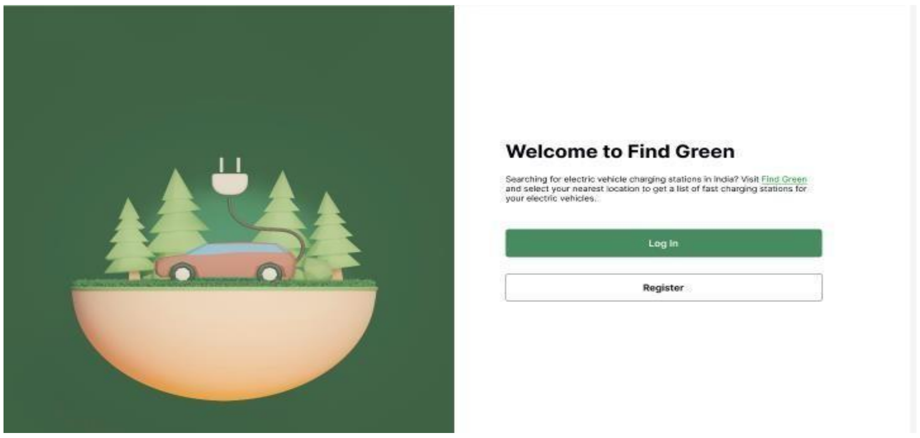
Adobe XD is a powerful, vector-based tool for digital design and prototyping user interfaces (UI) and user experiences (UX). Adobe XD has a wide range of features and applications, making it an essential tool for any UX designer.

Its capabilities include optimized performance, a clean interface, and an expansive set of features. One key feature is the live preview, which allows designers to see changes in real time. Another is the repeat grid, which makes it easy to create complex layouts with multiple elements. Adobe XD also offers support for voice commands and gestures, making it a great option for designing interactive prototypes.

5.3 Screenshots

5.3.1 Welcome Page

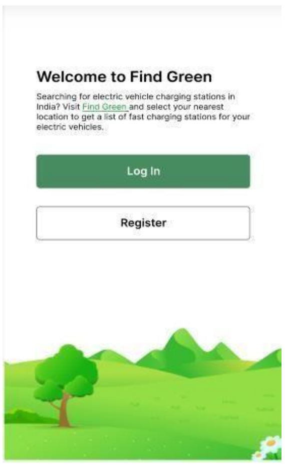
Fig 5.3 illustrates the welcome page of the website and Fig 5.4 illustrates the welcome screen of the app.This page appears once the user opens the app or the website. Here the user can click the login button to go to the login page if they already have an account or else they can click the register button which will take the user to the registration page.



Welcome to Find Green

Searching for electric vehicle charging stations in India? Visit Find Green and select your nearest location to get a list of fast charging stations for your electric vehicles.

Fig 5.3 Welcome Page for Web

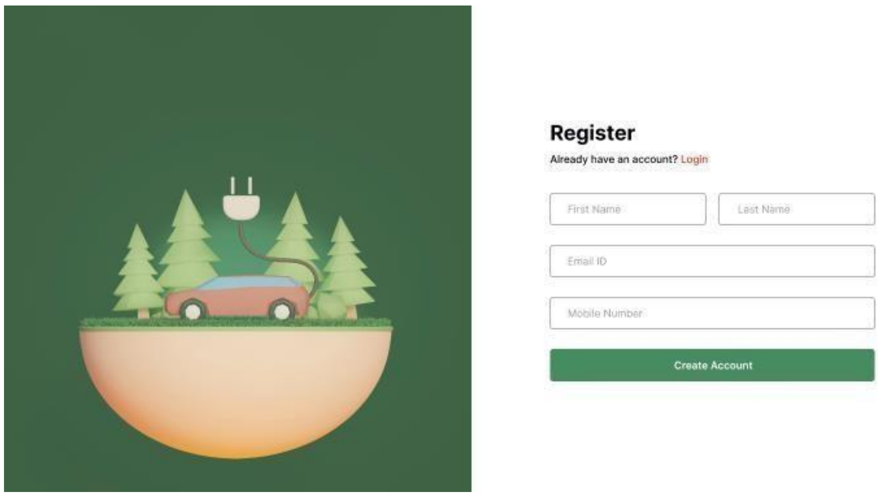

Welcome to Find Green

Searching for electric vehicle charging stations in India? Visit Find Green and select your nearest location to get a list of fast charging stations for your electric vehicles.

Fig 5.4 Welcome Page for App

5.3.2 Registration Page

Fig 5.5 illustrates the registration page of the website, Fig 5.6 illustrates the registration screen of the app.Here the user can register by entering their first and last name, email id and mobile number and click create account button. It will take the user to the OTP page. If the user already has an account, the user the click Login hyperlink to go to the login page.


Register

Already have an account? Login

Fig 5.5 Registration Page for Web

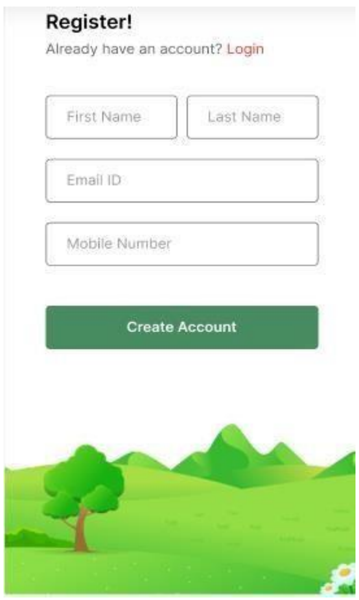
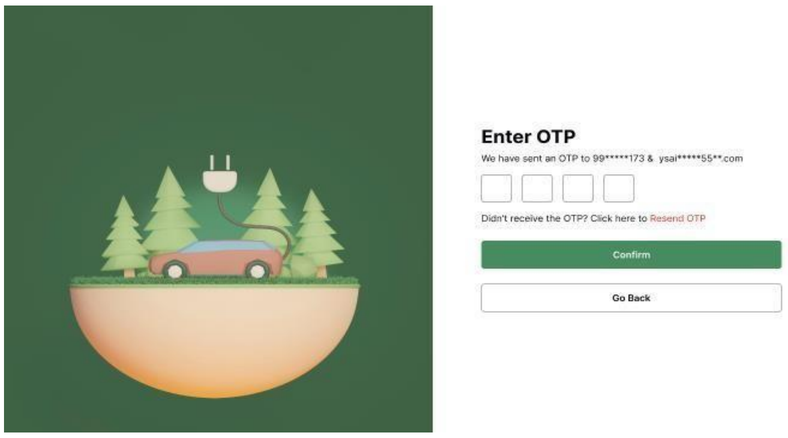


Fig 5.6 Registration Page for App

5.3.3 OTP Page

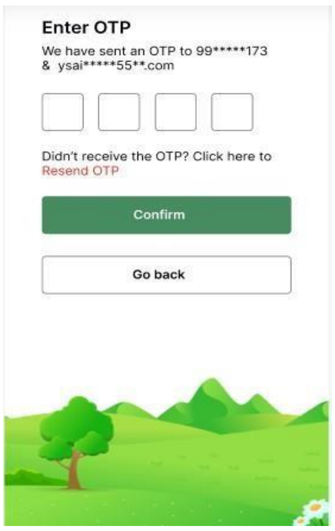
Fig 5.7 and 5.8 illustrates the otp page of the website and app. After clicking Create Account button in the registration page, it gets directed to this otp page. The user gets opt to their mobile number and email id. They should enter the correct opt to confirm their registration. If they didn't receive their otp, they can click Resend OTP hyperlink to get the otp again. If the user wants to change their details the can click Go Back button to go back to the registration page.



We have sent an OTP to 99*****173 & ysai*****55**.com

Didn't receive the OTP? Click here to Resend OTP

Fig 5.7 OTP Page for Web



Enter OTP

We have sent an OTP to 99*****173 & ysai*****55**.com

Fig 5.8 OTP Page for App

5.3.4 Homepage

Fig 5.9 and 5.10 illustrates the homepage of the website and app. Here the user can choose any option. If the user clicks My Profile it gets directed to profile page. If the user clicks 'My Bookings' it shows details about the user bookings. If the user clicks 'Charging history' it shows the user's charging history. If the user clicks 'Payment' it shows the payment details. If the user clicks Favourite station it shows the list of favourite stations. If the user clicks Logout it gets directed to Login page.

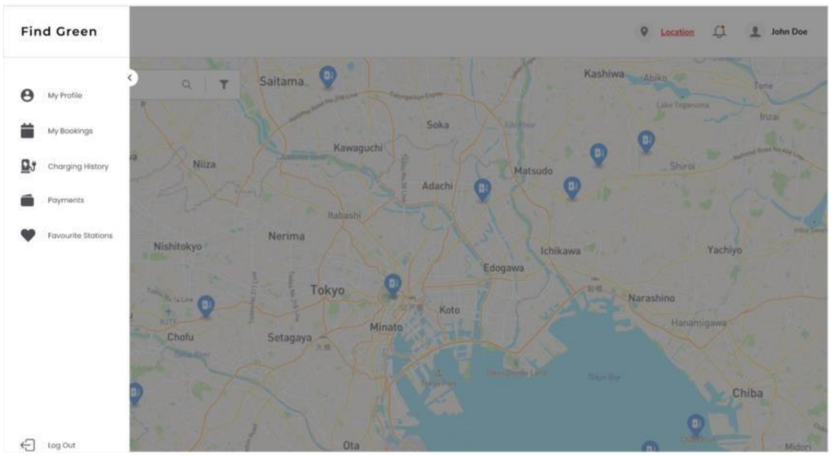


Fig 5.9 Homepage of Web

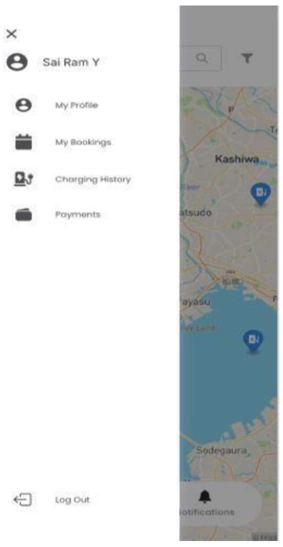


Fig 5.10 Homepage of App

5.3.5 Filter Screen

Fig 5.11 and 5.12 illustrates the filter page of the website and app. Here the user can filter the stations based on the charger type, availability, vehicle type, distance, connector type, favourite station, fast changing etc. Once Apply button is clicked, it list the station details based on the filters applied. If the Clear button is clicked, it will clear all the applied filters.

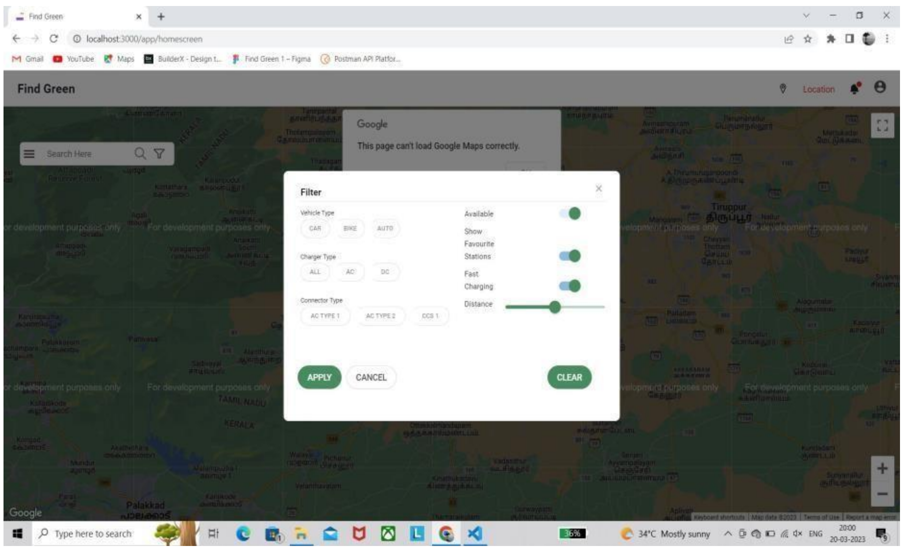


Fig 5.11 Filter Screen for Web

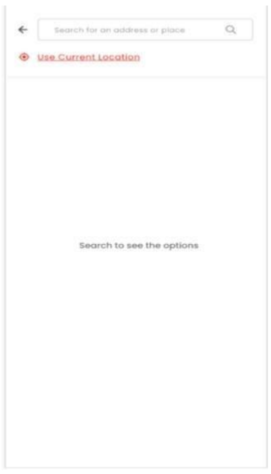
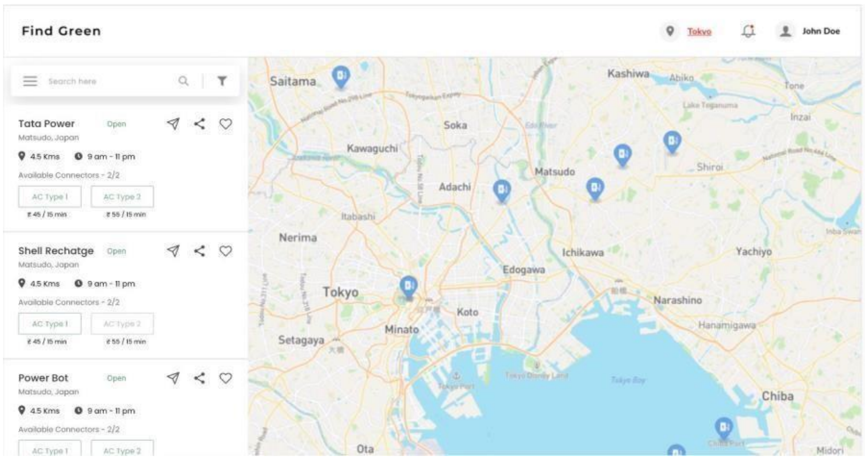


Fig 5.12 Filter Screen for App

5.3.6 Station Page

Fig 5.13 and 5.14 illustrates the station page of the website and app. It displays the station details based on the filters applied. The station details includes the station name, place, city, working hours, status, available connectors, connector types, rate etc. Here the can add the station as their favourite station or share the station to others or navigate to the station. The stations will be viewed either in list format or map view.

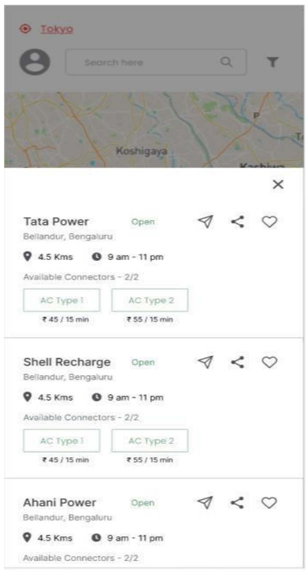


Tata Power Open
Matsudo, Japan
4.5 Kms 9 am - 11 pm
Available Connectors - 2/2
AC Type 1 AC Type 2
₹ 45 / 15 min ₹ 55 / 15 min

Shell Rechatge Open
Matsudo, Japan
4.5 Kms 9 am - 11 pm
Available Connectors - 2/2
AC Type 1 AC Type 2
₹ 45 / 15 min ₹ 55 / 15 min

Power Bot Open
Matsudo, Japan
4.5 Kms 9 am - 11 pm
Available Connectors - 2/2
AC Type 1 AC Type 2

Fig 5.13 Station Screen for Web


Tokyo

Search here

Tata Power Open
Bellandur, Bengaluru
4.5 Kms 9 am - 11 pm
Available Connectors - 2/2
AC Type 1 ₹ 45 / 15 min
AC Type 2 ₹ 55 / 15 min

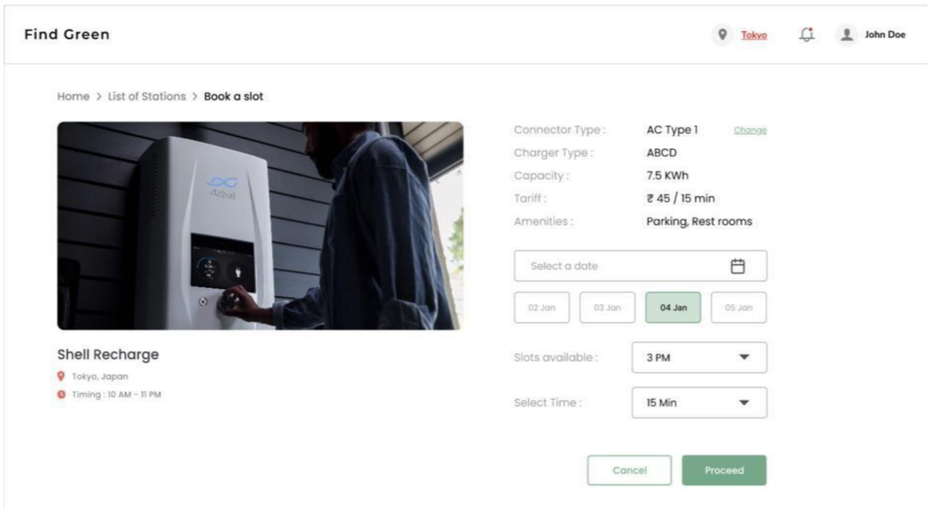
Shell Recharge Open
Bellandur, Bengaluru
4.5 Kms 9 am - 11 pm
Available Connectors - 2/2
AC Type 1 ₹ 45 / 15 min
AC Type 2 ₹ 55 / 15 min

Ahani Power Open
Bellandur, Bengaluru
4.5 Kms 9 am - 11 pm
Available Connectors - 2/2

Fig 5.14 Station Screen for App

5.3.7 Booking Page

Fig 5.15 and 5.16 illustrates the booking page of the website and app. The page appears when the user selects a station in the Station page. This page displays the station name, place, working hours, connector type, charger type, capacity, tariff, amenities, date and time pickers etc. Here the user can book a slot by selecting the date, time and duration. Once chosen, the user click 'Proceed' button, it will take the user to the payment page or the user can cancel using the Cancel button.


Find Green Tokyo John Doe

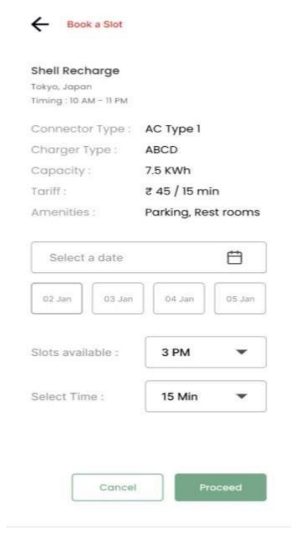
Home > List of Stations > Book a slot

Shell Recharge
Tokyo, Japan
Timing : 10 AM - 11 PM

Connector Type :	AC Type 1
Charger Type :	ABCD
Capacity :	7.5 KWh
Tariff :	₹ 45 / 15 min
Amenities :	Parking, Rest rooms

Select a date
02 Jan 03 Jan 04 Jan 05 Jan
Slots available : 3 PM
Select Time : 15 Min

Fig 5.15 Booking Screen for Web



Shell Recharge
Tokyo, Japan
Timing : 10 AM – 11 PM

Connector Type :	AC Type 1
Charger Type :	ABCD
Capacity :	7.5 KWh
Tariff :	₹ 45 / 15 min
Amenities :	Parking, Rest rooms

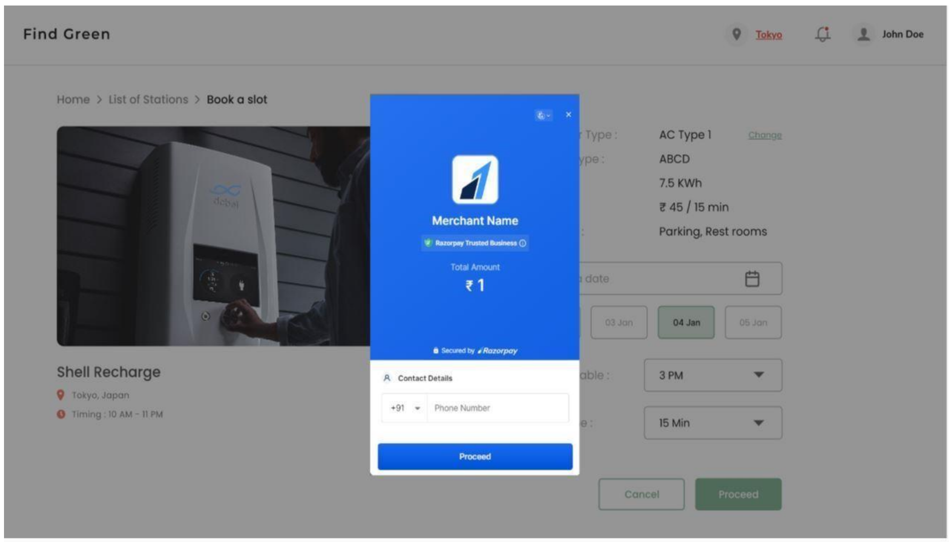
Select a date
02 Jan 03 Jan 04 Jan 05 Jan

Slots available : 3 PM
Select Time : 15 Min

Fig 5.16 Booking Screen for App

5.3.8 Payment Page

Fig 5.17 and 5.18 illustrates the payment page of the website and app. Once the slot is selected it gets directed to the payment page. It will display the amount to be paid and the methods of payment. Here the user should select the payment method and proceed with payment. If the payment is successful it will show success screen or it will show the failure screen.



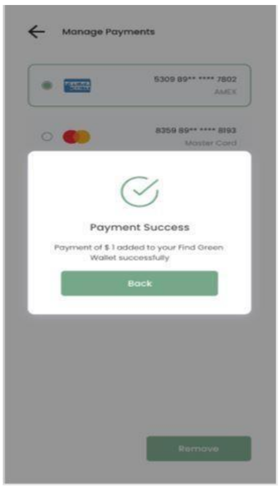
Shell Recharge
Tokyo, Japan
Timing : 10 AM – 11 PM

Merchant Name
Razorpay Trusted Business
Total Amount
₹ 1
Secured by Razorpay
Contact Details
+91 Phone Number

Type : AC Type 1 Change
ype : ABCD
7.5 KWh
₹ 45 / 15 min
: Parking, Rest rooms

date
03 Jan 04 Jan 05 Jan
able : 3 PM
e : 15 Min

Fig 5.17 Payment Screen for Web



Manage Payments

5309 89** **** 7802
AMEX

8359 89** **** 8193
Master Card

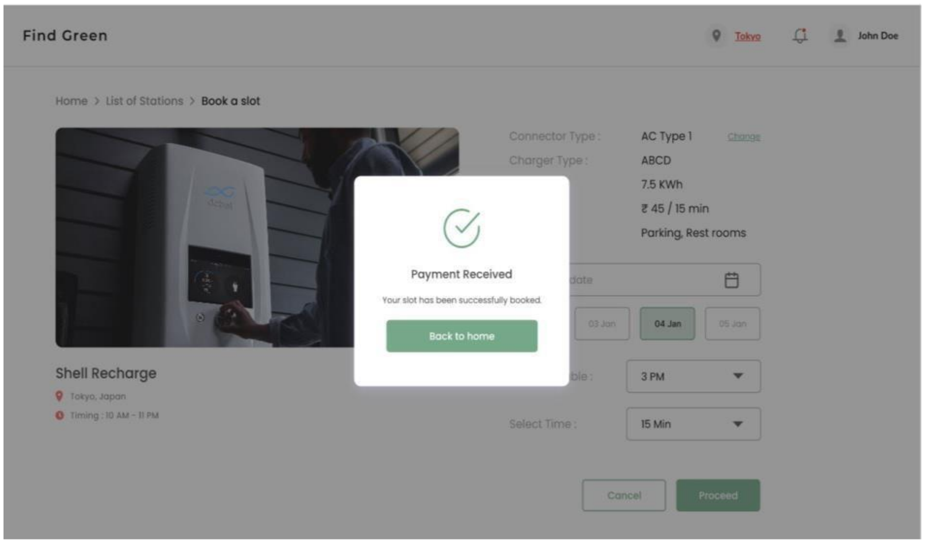
Payment Success

Payment of $ 1 added to your Find Green Wallet successfully

Fig 5.18 Payment Screen for App

5.3.9 Booking Success/Failure Page

Fig 5.19 and 5.20 illustrates the booking success/failure page of the website and app. If the payment is done successfully and if the slot is free then the slot gets booked and shows success message. If the payment is not successful or if the slot is not free then the slot doesn't get booked and it shows the booking failure page. Once payment is done it will take the user to the homepage.



Find Green

Tokyo

John Doe

Home > List of Stations > Book a slot

Connector Type : AC Type 1

Charger Type : ABCD

7.5 KWh

₹ 45 / 15 min

Parking, Rest rooms

date

03 Jan 04 Jan 05 Jan

ble : 3 PM

Select Time : 15 Min

Shell Recharge

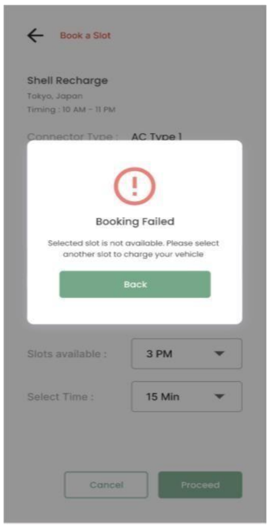
Tokyo, Japan

Timing : 10 AM - 11 PM

Payment Received

Your slot has been successfully booked.

Fig 5.19 Booking Success Screen for Web



Shell Recharge
Tokyo, Japan
Timing : 10 AM – 11 PM

Connector Type : AC Type 1

Booking Failed

Selected slot is not available. Please select another slot to charge your vehicle

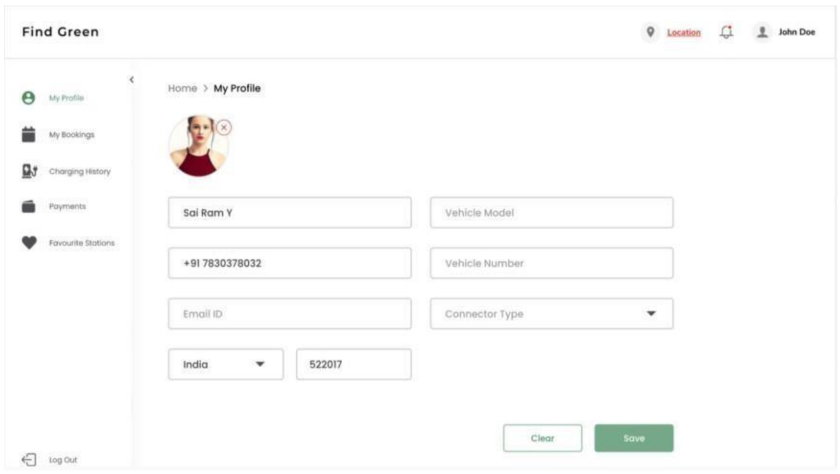
Slots available : 3 PM

Select Time : 15 Min

Fig 5.20 Booking Failure Screen for App

5.3.10 My Profile Page

Fig 5.21 and 5.22 illustrates the My profile page of the website and app. Here the user can edit their details like their first and last name, email id, mobile number, vehicle type, country, connector type etc. The user can add their profile picture by clicking the plus sign or remove the picture by clicking the X sign. The user should click Save to save the user details and it takes the user to the homepage.



Sai Ram Y
Vehicle Model
+91 7830378032
Vehicle Number
Email ID
Connector Type
India
522017

Fig 5.21 My Profile Screen for Web

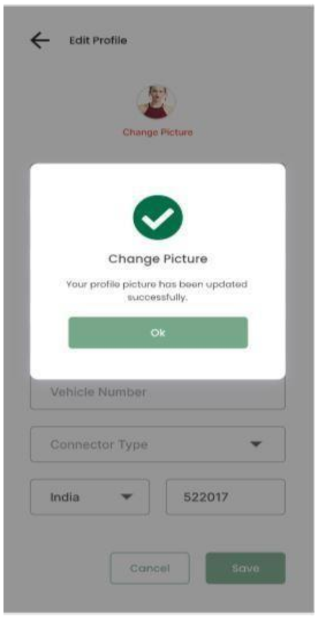
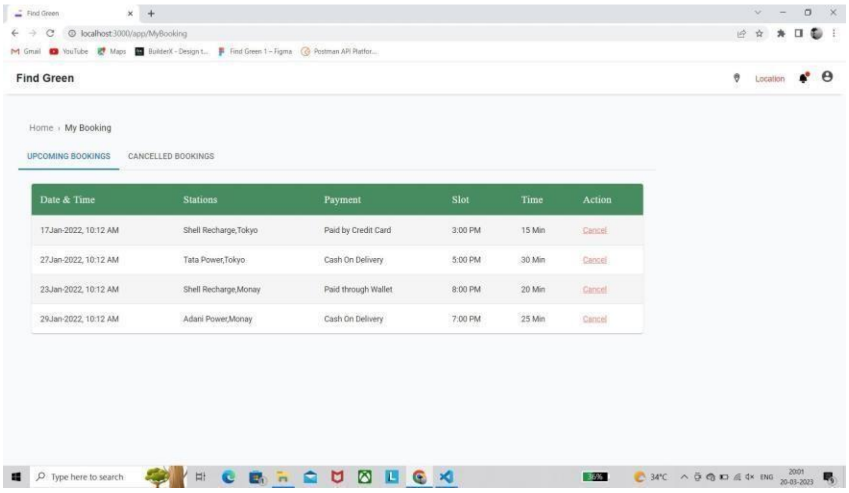


Fig 5.22 My Profile Screen for App

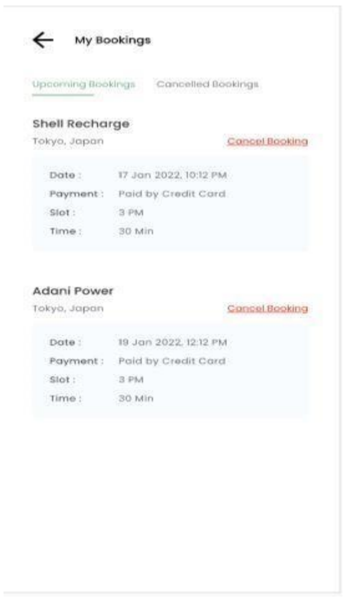
5.3.11 My Booking Page (Upcoming Bookings)

Fig 5.23 and 5.24 illustrates the upcoming booking page of the website and app. This page displays the user’s upcoming bookings. It displays the details like the date and time, station name, payment method, slot, time etc. The user can book the same station by clicking the station name. The user can cancel the booking by clicking the Cancel hyperlink.



Date & Time	Stations	Payment	Slot	Time	Action
17Jan-2022, 10:12 AM	Shell Recharge,Tokyo	Paid by Credit Card	3:00 PM	15 Min	Cancel
27Jan-2022, 10:12 AM	Tata Power,Tokyo	Cash On Delivery	5:00 PM	30 Min	Cancel
23Jan-2022, 10:12 AM	Shell Recharge,Monay	Paid through Wallet	8:00 PM	20 Min	Cancel
29Jan-2022, 10:12 AM	Adani Power,Monay	Cash On Delivery	7:00 PM	25 Min	Cancel

Fig 5.23 Upcoming Booking Screen for Web



Shell Recharge
Tokyo, Japan
Cancel Booking

Date : 17 Jan 2022, 10:12 PM
Payment : Paid by Credit Card
Slot : 3 PM
Time : 30 Min

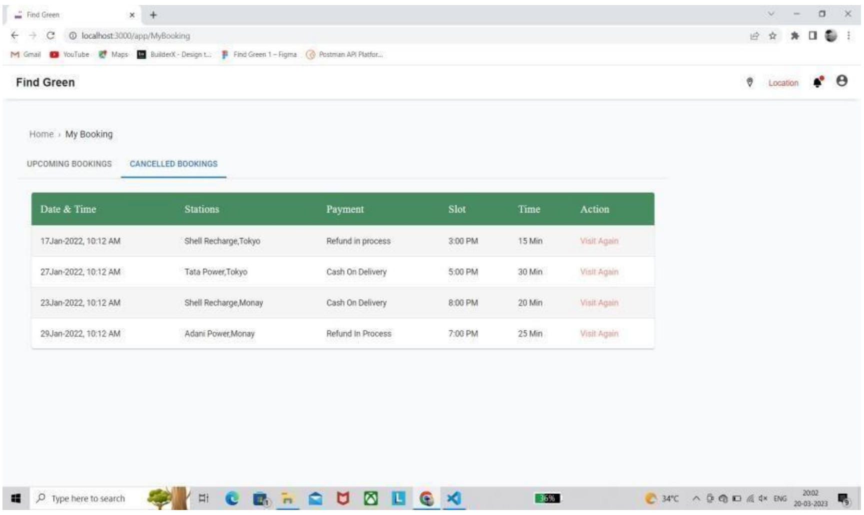
Adani Power
Tokyo, Japan
Cancel Booking

Date : 19 Jan 2022, 12:12 PM
Payment : Paid by Credit Card
Slot : 3 PM
Time : 30 Min

Fig 5.24 Upcoming Booking Screen for App

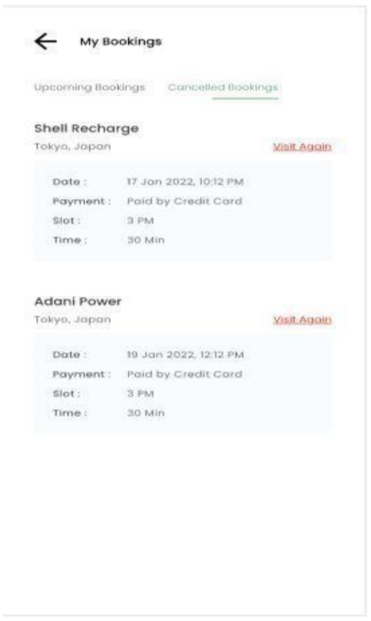
5.3.12 My Booking Page (Cancelled Bookings)

Fig 5.25 and 5.26 illustrates the user's cancelled bookings. It displays the details like the date and time, station name, payment method, slot, time etc. The user can book the same station by clicking the station name. The user can also book the station again by clicking the Visit Again hyperlink.



Date & Time	Stations	Payment	Slot	Time	Action
17Jan-2022, 10:12 AM	Shell Recharge,Tokyo	Refund in process	3:00 PM	15 Min	Visit Again
27Jan-2022, 10:12 AM	Tata Power,Tokyo	Cash On Delivery	5:00 PM	30 Min	Visit Again
23Jan-2022, 10:12 AM	Shell Recharge,Monay	Cash On Delivery	8:00 PM	20 Min	Visit Again
29Jan-2022, 10:12 AM	Adani Power,Monay	Refund In Process	7:00 PM	25 Min	Visit Again

Fig 5.25 Cancelled Booking Screen for Web



My Bookings

Upcoming Bookings | Cancelled Bookings

Shell Recharge
Tokyo, Japan
Visit Again

Date :	17 Jan 2022, 10:12 PM
Payment :	Paid by Credit Card
Slot :	3 PM
Time :	30 Min

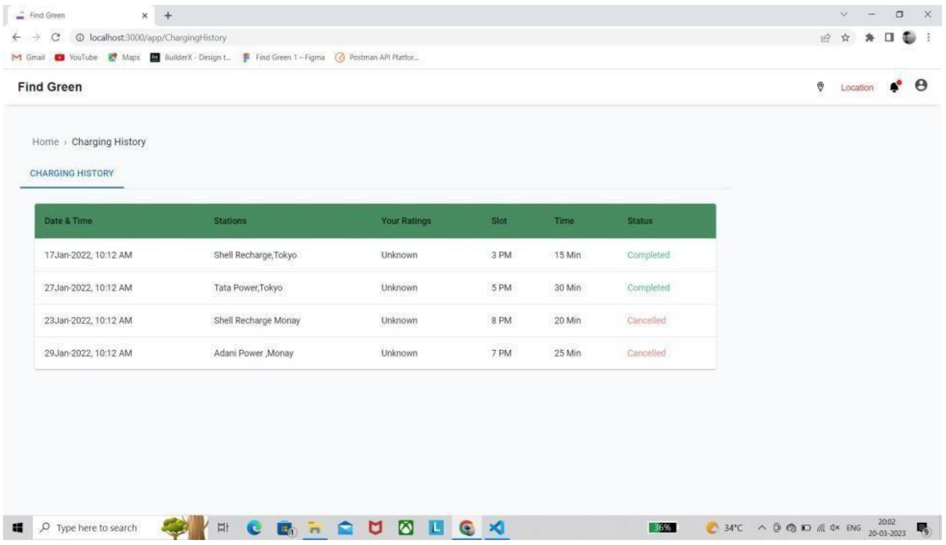
Adani Power
Tokyo, Japan
Visit Again

Date :	19 Jan 2022, 12:12 PM
Payment :	Paid by Credit Card
Slot :	3 PM
Time :	30 Min

Fig 5.26 Cancelled Booking Screen for App

5.3.13 Charging History Page

Fig 5.28 and 5.29 illustrates the charging history page of the website and app. This page shows the charging history of the user. It shows the details of the completed or cancelled booking of the user. It displays the details like the date and time, station name, user ratings, slot, time, completed or cancelled status etc. It also contains filter option using which the user can view the history based on days, months or years.



localhost:3000/app/ChargingHistory

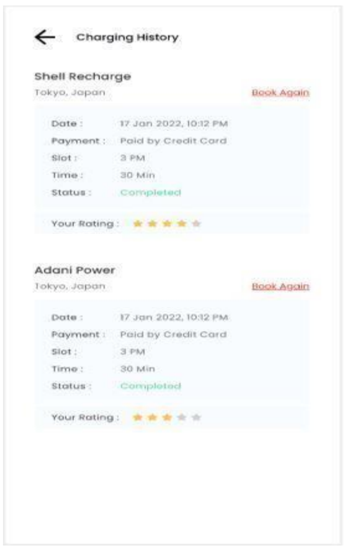
Find Green

Home › Charging History

CHARGING HISTORY

Date & Time	Stations	Your Ratings	Slot	Time	Status
17Jan-2022, 10:12 AM	Shell Recharge,Tokyo	Unknown	3 PM	15 Min	Completed
27Jan-2022, 10:12 AM	Tata Power,Tokyo	Unknown	5 PM	30 Min	Completed
23Jan-2022, 10:12 AM	Shell Recharge Monay	Unknown	8 PM	20 Min	Cancelled
29Jan-2022, 10:12 AM	Adani Power ,Monay	Unknown	7 PM	25 Min	Cancelled

Fig 5.27 Charging History Screen for Web


Charging History

Shell Recharge
Tokyo, Japan
Book Again

Date : 17 Jan 2022, 10:12 PM
Payment : Paid by Credit Card
Slot : 3 PM
Time : 30 Min
Status : Completed

Your Rating : ★★★★★

Adani Power
Tokyo, Japan
Book Again

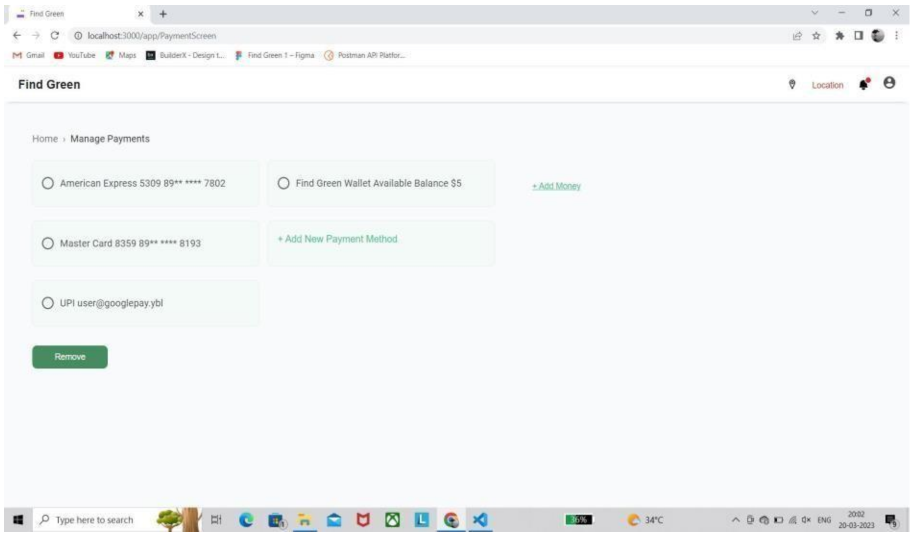
Date : 17 Jan 2022, 10:12 PM
Payment : Paid by Credit Card
Slot : 3 PM
Time : 30 Min
Status : Completed

Your Rating : ★★★★★

Fig 5.28 Charging History Screen for App

5.3.14 Manage Payment Page

Fig 5.29 and 5.30 illustrates the manage payment page of the website and app. Here the user can add their payment methods like card details or recharge money to their wallet, add the gpay account etc. The user can also remove the payment methods by clicking the Remove button.


localhost:3000/app/PaymentScreen

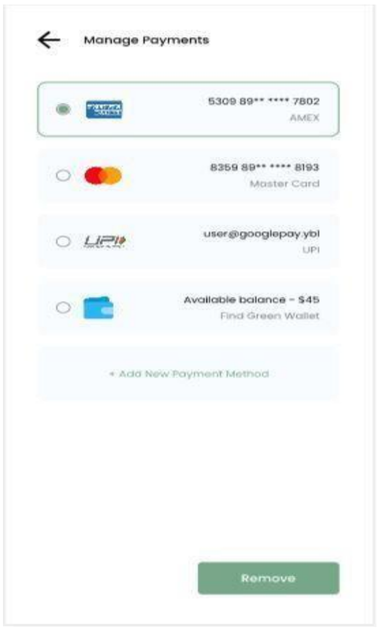
Find Green

Home › Manage Payments

American Express 5309 89** **** 7802
Find Green Wallet Available Balance $5
+ Add Money
Master Card 8359 89** **** 8193
+ Add New Payment Method
UPI user@googlepay.ybl

Remove

Fig 5.29 Manage Payment Screen for Web



5309 89** **** 7802
AMEX

8359 89** **** 8193
Master Card

user@googlepay.ybl
UPI

Available balance - $45
Find Green Wallet

Fig 5.30 Manage Payment Screen for App

5.3.15 Favourite Page

Fig 5.31 illustrates the favourites page of the website and app. This page displays the list of favourite stations added by the user. The station details includes the station name, place, city, working hours, status, available connectors, connector types, rate etc. Here the user can remove favourite station or share the station to others or navigate to the station. The stations will be viewed either in the list format or map view.

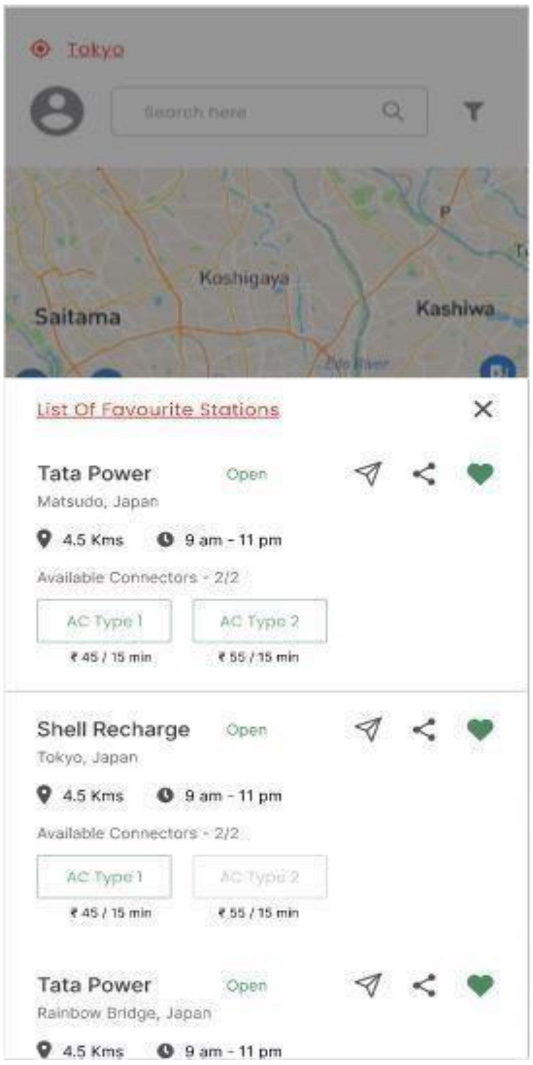
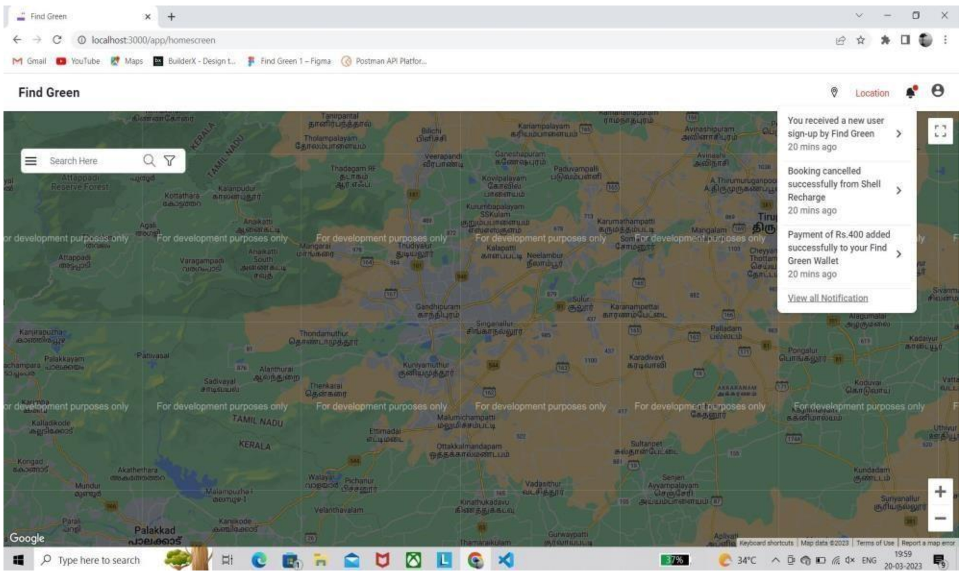


Fig 5.31 Favourite Station Screen for App

5.3.16 Notification Page

Fig 5.32 illustrates the notification page of the website and Fig 5.33 illustrates the notification screen of the app.This page displays the notifications like successful booking or reminding about the booking, payment successful or payment failure, refund amount etc.



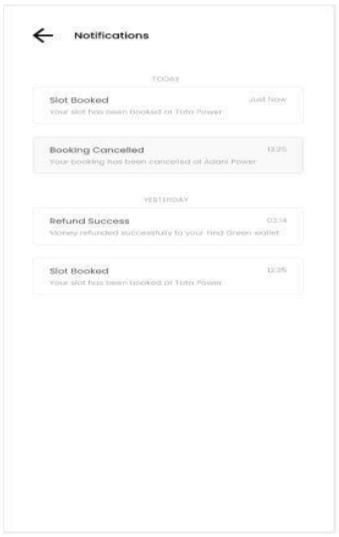
You received a new user sign-up by Find Green
20 mins ago

Booking cancelled successfully from Shell Recharge
20 mins ago

Payment of Rs.400 added successfully to your Find Green Wallet
20 mins ago

View all Notification

Fig 5.32 Notification Screen for Web



Slot Booked — Just Now
Your slot has been booked at Tata Power

Booking Cancelled — 13:25
Your booking has been canceled at Adani Power

Refund Success — 03:14
Money refunded successfully to your Find Green wallet

Slot Booked — 12:35
Your slot has been booked at Tata Power

Fig 5.33 Notification Screen for App

5.3.17 Rating Screen

Fig 5.34 and 5.35 illustrates the rating screen of the website and app. This screen appears after the successful booking a slot in the station. It displays five stars meaning that the user should rate the station in the range of 1 to 5. The user can ignore it by clicking Cancel button. Once the stars are clicked, the user can click the Submit button. It will go the success page.

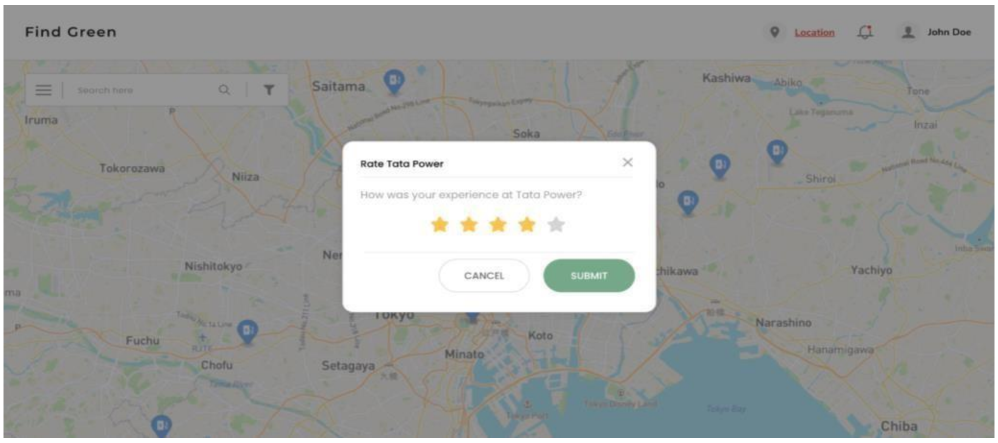


Fig 5.34 Rating Screen for Web

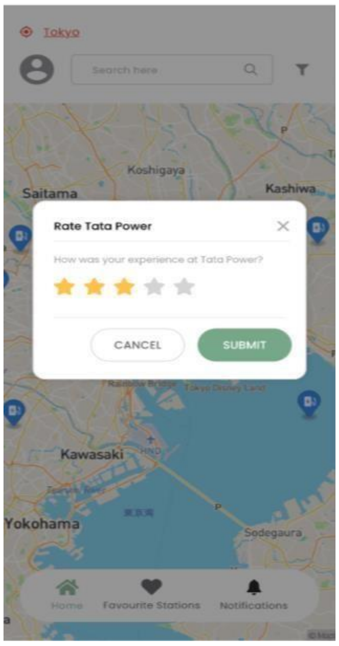


Fig 5.35 Rating Screen for App

5.3.18 Search Screen

Fig 5.36 and 5.37 illustrates the search page of the website and app. Here the user can search the station based on the address, name, area, city, pincode etc. Once Allow button is clicked, it list the station details based on the filters applied. If the user clicks the Cancel button, it will not search the station and goes to the homepage. The user can also search the stations by clicking 'Use current location'. It will show the nearby stations.

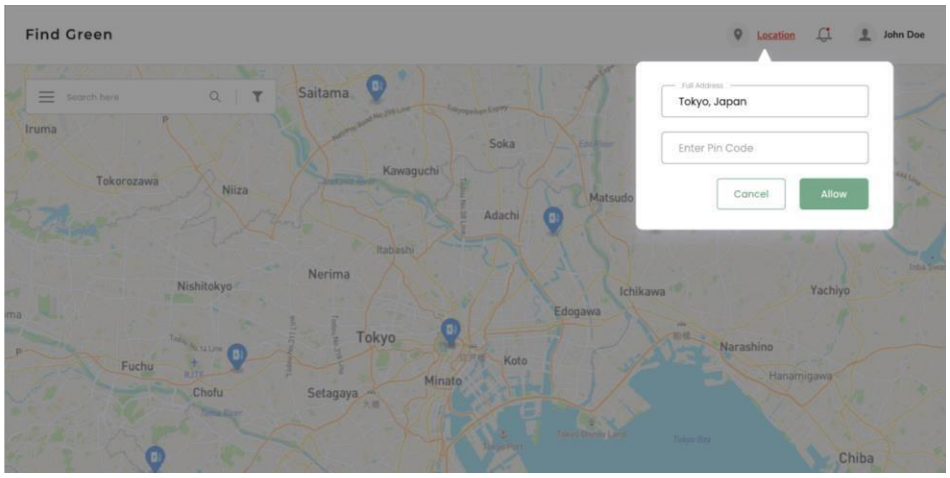


Fig 5.36 Search Screen for Web

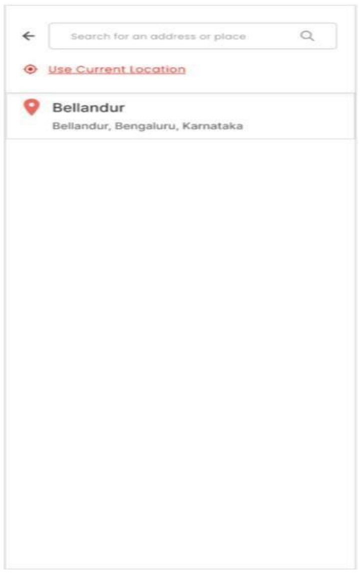
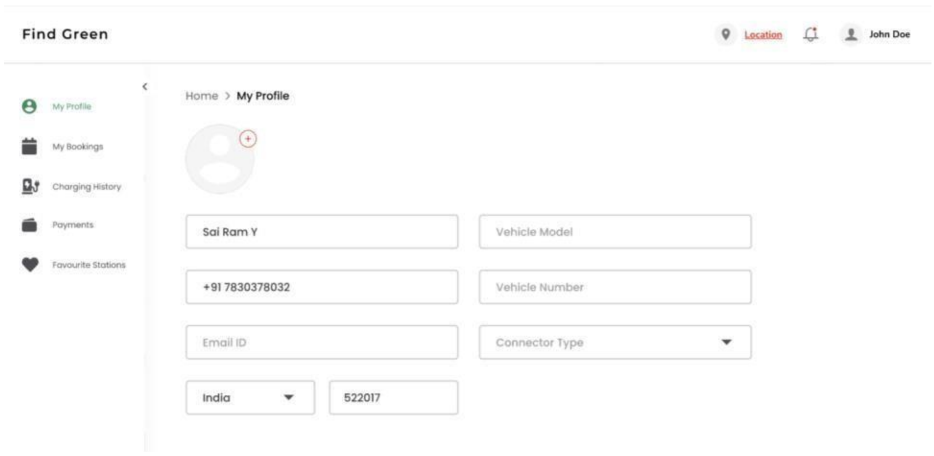

Search for an address or place
Use Current Location
Bellandur
Bellandur, Bengaluru, Karnataka

Fig 5.37 Search Screen for App

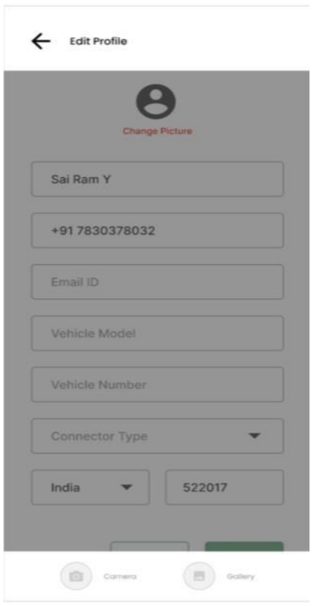
5.3.19 My Profile Page (Profile Picture)

Fig 5.38 and 5.39 illustrates the My profile page of the website and app. The user can add their profile picture by clicking the plus sign or remove the picture by clicking the X sign. When the plus sign is clicked, the user will ask whether to choose the picture from the camera or gallery. Once that is clicked, the user can choose any picture and click Ok. The user should click Save to save the user details and it takes the user to the homepage.



Sai Ram Y
+91 7830378032
Email ID
India
522017
Vehicle Model
Vehicle Number
Connector Type

Fig 5.38 Profile picture Screen for Web



Sai Ram Y
+91 7830378032
Email ID
Vehicle Model
Vehicle Number
Connector Type
India
522017

Fig 5.39 Profile picture Screen for App

6 SYSTEM TESTING

6.1 Testing

System testing is a critical aspect of Software Quality Assurance and represents the ultimate review of specification, design and coding. Testing is a process of executing a program with the intent of finding an error. A good test is one that has a probability of finding an as yet undiscovered error. The purpose of testing is to identify and correct bugs in the developed system. Nothing is complete without testing.

Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to the process of executing a program or application with the intent of finding Software Bugs.

Testing is vital to the success of the system. System testing marks a logical assumption that all the parts of the system are correct; the goal successfully achieved. Special test data is input for processing and the results are examined to locate unexpected results.

NodeJS unit testing refers to testing individual units or components of a Node.js application using specialized automation testing frameworks and libraries. These tests can include testing the functionality of individual functions, modules, or classes and the interactions between different parts of the application. Some popular NodeJS unit testing frameworks include Jest, Mocha, and AVA.

6.2 Unit Testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself. It is correlated with the functional correctness of the independent modules.

Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.

Unit testing is the first level of testing done before integration testing. Unit testing is such a type of testing technique that is usually performed by developers. Although due to the reluctance of developers to test, quality assurance engineers also do unit testing.

NodeJS Unit testing is the method of testing small pieces of code/components in isolation in your NodeJS application. This helps in improving the code quality and helps in finding bugs early on in the development life cycle.

This also provides an added advantage to the users in the way that they can add any new features without breaking any other part of their application. For the NodeJS applications, Jest can be used for Unit Testing.

6.3 JEST

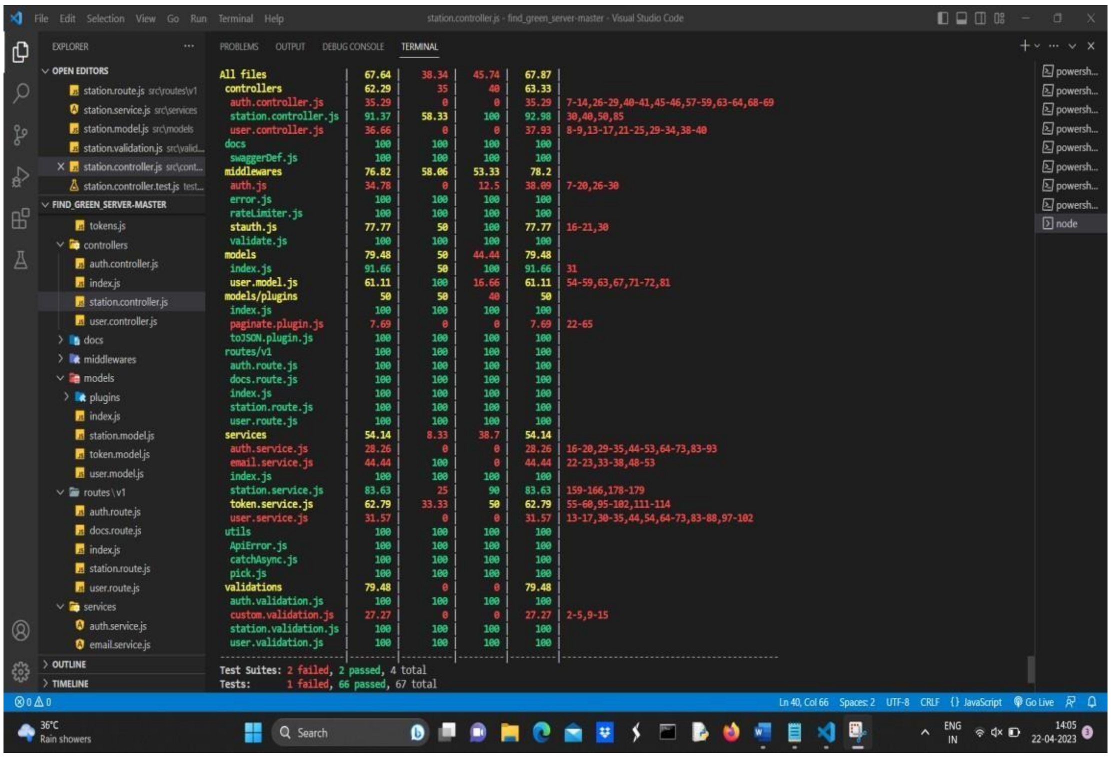
Jest is a *Javascript Testing Framework* by Facebook. It is used most commonly for unit testing. Unit testing is when you provide input to a unit of code(usually, a function) and match the output with the expected output.

Jest Features:

- ***zero config:*** Close to none configuration is required to get started with writing tests and deploying them. However, a config file can be supplied to the test suite as well.
- ***snapshots:*** Jest has the ability to enable snapshot testing as well. Essentially, the snapshots are matched with the saved snapshot and check for matching functionality.

- ***isolated tests:*** Jest tests are run parallelly to improve run time.

Fig 6.1 illustrates the code coverage percentage for testing the project code using the jest in visual studio code teminal.



```
All files               |   67.64 |   38.34 |   45.74 |   67.87 |
 controllers            |   62.29 |      35 |      40 |   63.33 |
  auth.controller.js    |   35.29 |       0 |       0 |   35.29 | 7-14,26-29,40-41,45-46,57-59,63-64,68-69
  station.controller.js |   91.37 |   58.33 |     100 |   92.98 | 30,40,50,85
  user.controller.js    |   36.66 |       0 |       0 |   37.93 | 8-9,13-17,21-25,29-34,38-40
 docs                   |     100 |     100 |     100 |     100 |
  swaggerDef.js         |     100 |     100 |     100 |     100 |
 middlewares            |   76.82 |   58.06 |   53.33 |    78.2 |
  auth.js               |   34.78 |       0 |    12.5 |   38.09 | 7-20,26-30
  error.js              |     100 |     100 |     100 |     100 |
  rateLimiter.js        |     100 |     100 |     100 |     100 |
  stauth.js             |   77.77 |      50 |     100 |   77.77 | 16-21,30
  validate.js           |     100 |     100 |     100 |     100 |
 models                 |   79.48 |      50 |   44.44 |   79.48 |
  index.js              |   91.66 |      50 |     100 |   91.66 | 31
  user.model.js         |   61.11 |     100 |   16.66 |   61.11 | 54-59,63,67,71-72,81
 models/plugins         |      50 |      50 |      40 |      50 |
  index.js              |     100 |     100 |     100 |     100 |
  paginate.plugin.js    |    7.69 |       0 |       0 |    7.69 | 22-65
  toJSON.plugin.js      |     100 |     100 |     100 |     100 |
 routes/v1              |     100 |     100 |     100 |     100 |
  auth.route.js         |     100 |     100 |     100 |     100 |
  docs.route.js         |     100 |     100 |     100 |     100 |
  index.js              |     100 |     100 |     100 |     100 |
  station.route.js      |     100 |     100 |     100 |     100 |
  user.route.js         |     100 |     100 |     100 |     100 |
 services               |   54.14 |    8.33 |    38.7 |   54.14 |
  auth.service.js       |   28.26 |       0 |       0 |   28.26 | 16-20,29-35,44-53,64-73,83-93
  email.service.js      |   44.44 |     100 |       0 |   44.44 | 22-23,33-38,48-53
  index.js              |     100 |     100 |     100 |     100 |
  station.service.js    |   83.63 |      25 |      90 |   83.63 | 159-166,178-179
  token.service.js      |   62.79 |   33.33 |      50 |   62.79 | 55-60,95-102,111-114
  user.service.js       |   31.57 |       0 |       0 |   31.57 | 13-17,30-35,44,54,64-73,83-88,97-102
 utils                  |     100 |     100 |     100 |     100 |
  ApiError.js           |     100 |     100 |     100 |     100 |
  catchAsync.js         |     100 |     100 |     100 |     100 |
  pick.js               |     100 |     100 |     100 |     100 |
 validations            |   79.48 |       0 |       0 |   79.48 |
  auth.validation.js    |     100 |     100 |     100 |     100 |
  custom.validation.js  |   27.27 |       0 |       0 |   27.27 | 2-5,9-15
  station.validation.js |     100 |     100 |     100 |     100 |
  user.validation.js    |     100 |     100 |     100 |     100 |
------------------------|---------|---------|---------|---------|-----------------------------------------
Test Suites: 2 failed, 2 passed, 4 total
Tests:       1 failed, 66 passed, 67 total
```

Fig 6.1 Code Coverage

6.4 Integration Testing

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration testing is a software testing technique that focuses on verifying the interactions and data exchange between different components or modules of a software application. The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other. Integration testing is typically performed after unit testing and before system testing. It helps to identify and resolve integration issues early in the development cycle, reducing the risk of more severe and costly problems later on.

Integration testing can be done by picking module by module. This can be done so that there should be a proper sequence to be followed. And also if the developer don't want to miss out on any integration scenarios then the developer have to follow the proper sequence. Exposing the defects is the major focus of the integration testing and the time of interaction between the integrated units.

Since testing is the common thing, there are a number of libraries and tools such as JSUnit, Unit.js, QUnit, Jasmine, Karma, Mocha, Chai etc…

6.5 Mocha and Chai

Mocha and Chai are two JavaScript frameworks commonly used together for unit testing. Mocha is a testing framework that provides functions that are executed according in a specific order, and that logs their results to the terminal window.

When the tester read tests written in Mocha, the tester will see regular use of the keywords describe and it. These keywords, provided by Mocha, provide structure to the tests by batching them into test suites and test cases.

A test suite is a collection of tests all relating to a single functionality or behavior. A test case or a unit test is a single description about the desired behavior of the code that either passes or fails. Test suites are batched underneath the describe keyword, and test cases are batched under the it keyword.

Additionally, Mocha provides tools for cleaning the state of the software being tested in order to insure that test cases are being run independently of each other. The tester might end up using other tools, to stub or mock the desired behaviors of other units that a given unit of code might interact with. The independence of test cases is a key principle of unit testing, as it allows the cause of errors to be pinpointed more specifically if a test case fails, thereby speeding up the debugging process.

6.5.1 Sample Test Cases

The following are the few test cases for Create station API. Here in the first test condition, it is written for successful creation and it returns 201 code meaning that the station is created. In the second test condition, it is written for invalid access token. When an invalid token is passed or if token is missing, it will throw the 400 error. In the third test condition, it is written for invalid input like pincode has less than 6 characters. So if user gives pincode with less than 6 characters, it will throw the 400 error. The remaining test cases are in Appendix 1 under station.controller.test.js

// test cases

```
describe('createStation', () => {

  let newStation;

  test('should return 201 and successfully create new station if data is ok', async ()
=> {

    newStation = {
      station_name: "shell ev charging station",
      station_address: "professor colony",
      station_area: "seliyur", station_city:
      "chennai", station_pincode: "620082",
      station_latitude:"27.28",
      station_longitude:"29.6",
      station_working_time: "5-11",
      station_available_connector: "3",
      station_connector_type: "AC Type 1",
      station_charger_type: "ac",
      station_charging: "fast",
      station_vehicle_type: "bike",
      station_status: "available"
    };

    const res = await request(app)
      .post('/v1/station/add-station')
      .set('x-access-token', `${token}`)
      .send(newStation)
      .expect(201);
  });

  test('should return 400 error if access token is missing', async () => {

    await request(app).post('/v1/station/add-station')
    .set('x-access-token', `${token}`)
    .send().expect(400);

  });

    test('should return 400 error if pincode length is less than 6 characters', async ()
=> {

    newStation = {
      station_name: "shell ev charging station", station_address:
      "professor colony", station_area: "seliyur", station_city:
      "chennai", station_pincode: "89065",
      station_latitude:"27.28", station_longitude:"29.6",
      station_working_time: "5-11",
      station_available_connector: "3", station_connector_type:
      "AC Type 1", station_charger_type: "ac",
      station_charging: "fast", station_vehicle_type: "bike",
      station_status: "available"
    };

    await request(app)
      .post('/v1/station/add-station')
      .set('x-access-token', `${token}`)
      .send(newStation)
      .expect(201);

  });
});
```

7 SYSTEM INTEGRATION AND DEPLOYMENT

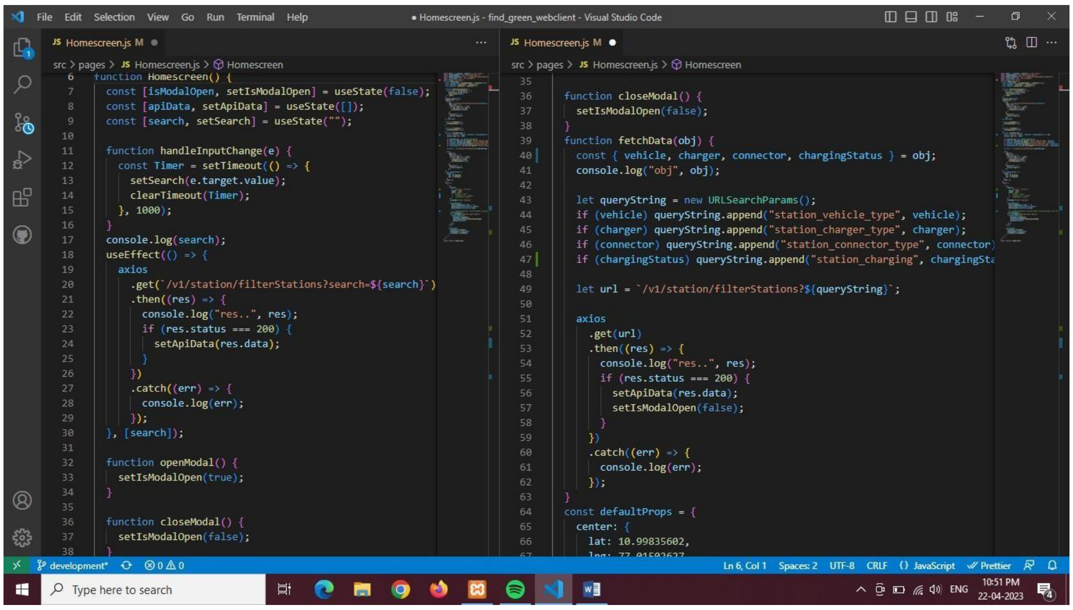
7.1 Integration

System Integration refers to the process by which multiple individual subsystems or sub-components are combined into one all-encompassing larger system thereby allowing the subsystems to function together. In other words, the symbiosis created through system integration allows the main system to achieve the overarching functionality required by the organisation.

In most organisations that use system integration there is a need to improve efficiency and thereby productivity and quality of their operations. The objective is usually to get the company's various IT systems to communicate with each other in the background so as to avoid the time and effort spent manually sharing information with other departments/components of the organisation including upper management. Through system integration, the organisation will experience increase in information flow speeds as well as reduced operational costs.

Furthermore, system integration connects the organisation with third parties such as suppliers, customers and shareholders. Each of which have their own unique interests in information generated by your company. System integration allows suppliers keep track of raw material levels, allows customers to keep track of finished goods inventory and shareholders to view the company position at a glance in a dashboard style in real time. All of these conditions can be met very simply through the use of system integration supplied by a reliable systems integrator.

Fig 7.1 illustrates the integration code of this project using reactjs in the visual studio code.


File Edit Selection View Go Run Terminal Help
• Homescreen.js - find_green_webclient - Visual Studio Code
Homescreen.js M
src > pages > Homescreen.js > Homescreen
development*
Ln 6, Col 1 Spaces: 2 UTF-8 CRLF JavaScript Prettier
Type here to search
10:51 PM
22-04-2023

```javascript
function Homescreen() {
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [apiData, setApiData] = useState([]);
  const [search, setSearch] = useState("");

  function handleInputChange(e) {
    const Timer = setTimeout(() => {
      setSearch(e.target.value);
      clearTimeout(Timer);
    }, 1000);
  }
  console.log(search);
  useEffect(() => {
    axios
      .get(`/v1/station/filterStations?search=${search}`)
      .then((res) => {
        console.log("res..", res);
        if (res.status === 200) {
          setApiData(res.data);
        }
      })
      .catch((err) => {
        console.log(err);
      });
  }, [search]);

  function openModal() {
    setIsModalOpen(true);
  }

  function closeModal() {
    setIsModalOpen(false);
  }
```

```javascript

  function closeModal() {
    setIsModalOpen(false);
  }
  function fetchData(obj) {
    const { vehicle, charger, connector, chargingStatus } = obj;
    console.log("obj", obj);

    let queryString = new URLSearchParams();
    if (vehicle) queryString.append("station_vehicle_type", vehicle);
    if (charger) queryString.append("station_charger_type", charger);
    if (connector) queryString.append("station_connector_type", connector)
    if (chargingStatus) queryString.append("station_charging", chargingSta

    let url = `/v1/station/filterStations?${queryString}`;

    axios
      .get(url)
      .then((res) => {
        console.log("res..", res);
        if (res.status === 200) {
          setApiData(res.data);
          setIsModalOpen(false);
        }
      })
      .catch((err) => {
        console.log(err);
      });
  }
  const defaultProps = {
    center: {
      lat: 10.99835602,
```

Fig 7.1 Integration Code

7.2 Deployment

Deployment in software and web development means pushing changes or updates from one deployment environment to another. When setting up a website the developer will always have the live website, which is called the live environment or production environment.

If the developer want the ability to make changes without these affecting your live website, then you can add additional environments. These environments are called development environments or deployment environments. The additional development environments will typically be a local environment, a development environment, and a staging environment (also known as a staging site). Once this deployment process has been completed the new changes will be visible in the live environment.

8 CONCLUSION

8.1 Working of the system

The EV Charging station finder app is successfully developed for electric vehicle owners, allowing them to easily locate and navigate to charging stations in their area. The app’s user-friendly interface and real-time data will make it easy for users to find the best charging station for their needs, while the payment options and user reviews will enhance the overall user experience.

8.2 Implementation

Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on the new system for the users, what it will workefficient and effectively. It involves careful planning, investing of the currentsystem, and its constraints on implementation, design of methods to achieve the change over methods.

This system has to be integrated with the website and hosted in the internet.

8.3 Future Enhancement

- ***Station Maintenance and Repair Notifications*****:** The app can notify users when charging stations are undergoing maintenance or are out of service, helping them plan their charging needs accordingly.
- ***Personalized Charging Recommendations:*** The app can use data on users' driving habits and vehicle battery status to provide personalized charging recommendations and alerts.

- ***Integration with vehicle systems:*** Another potential enhancement would be to integrate the app with the vehicle's systems, allowing users to see the battery charge level and estimated range, and to receive notifications when their vehicle is fully charged.

APPENDICES

Appendix I (Sample Codes)

//station_model.js

```
const   bcrypt   =   require('bcryptjs');   const
Station = (sequelize, DataTypes) => {
  const station = sequelize.define(
  'findstation',
    { station_name: { type:
      DataTypes.STRING,
      unique: false, allowNull:
      false, validate: { notEmpty:
      true,
        },
      }, station_address: { type:
      DataTypes.STRING,
      unique: false, allowNull:
      false, validate: { notEmpty:
      true,
        }, },
      station_area:
      {
        type: DataTypes.STRING,
        unique: false,
        allowNull: false,
        validate: { notEmpty:
        true,
        }, }, station_city: { type:
      DataTypes.STRING,
      unique: false, allowNull:
```

```
false, validate: { notEmpty:
true,
  },
},    station_pincode:    {
type:   DataTypes.REAL,
unique:  false,  allowNull:
false, validate: {
    notEmpty: true,
  },  },  station_latitude:  {
type:   DataTypes.REAL,
unique:  false,  allowNull:
false, validate: {
    notEmpty: true,
  },
}, station_longitude:
{ type:
DataTypes.REAL,
unique: false,
allowNull: false,
validate: {
    notEmpty: true,
  },
},
station_working_time: {
  type: DataTypes.STRING,
  unique: false, allowNull:
  false, validate: {
  notEmpty: true,
  },
}, station_available_connector:
{
  type: DataTypes.STRING,
  unique: false, allowNull:
```

```
  false, validate: {
  notEmpty: true,
  },
}, station_connector_type:
{
  type: DataTypes.STRING,
  unique: false, allowNull:
  false, validate: {
  notEmpty: true,
  },
},
station_charger_type: {
  type: DataTypes.STRING,
  unique: false, allowNull:
  false, validate: {
  notEmpty: true,
  },
}, station_charging: { type:
DataTypes.STRING,
unique: false, allowNull:
false, validate: { notEmpty:
true,
  },
}, station_vehicle_type:
{
  type: DataTypes.STRING,
  unique: false, allowNull:
  false, validate: {
  notEmpty: true,
  }, }, station_status: { type:
DataTypes.STRING,
unique: false, allowNull:
false, validate: { notEmpty:
true,
  },
      },
    },
    {         timestamps:
      false,
    }
  );
  return station;
};
module.exports = Station;
```

//station_controller.js

```
const httpStatus = require('http-status'); const
ApiError = require('../utils/ApiError'); const
catchAsync = require('../utils/catchAsync');
const { stationService } = require('../services');
const { tokenService } = require('../services');
const logger = require('../config/logger');
const createStation = catchAsync(async (req, res) => {
    const station = await stationService.createStation(req.body); const
    tokens = await tokenService.generateAuthTokens(station); res
    .cookie('refreshToken', tokens.refresh.token, {
      maxAge: tokens.refresh.maxAge, httpOnly:
      true, sameSite: 'none', secure: true,
    })
    .status(httpStatus.CREATED).send({    station,    token:    tokens.access    });
    logger.info(station);
  });

  const searchStations = catchAsync(async (req, res) => {
    const station = await stationService.queryStations(req.query.stationSearch,
req.query.station_pincode); if
    (!station) {
      throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
    } logger.info(station);
    res.send(station);
  });

  const filterStations = catchAsync(async (req, res) => {
    const station = await
stationService.filterStations(req.query.station_vehicle_type,
req.query.station_charger_type, req.query.station_connector_type,
req.query.station_status, req.query.station_charging);

    if (!station) {
      throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
    }
    logger.info(station); res.send(station);
  });

  const findStations = catchAsync(async (req, res) => {
    const station = await stationService.findStations(req.query.stationSearch,
req.query.station_pincode,req.query.station_vehicle_type,
req.query.station_charger_type, req.query.station_connector_type,
req.query.station_status, req.query.station_charging);
    if (!station) {
      throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
    } logger.info(station);
    res.send(station);
  });

  const nearStation = catchAsync(async (req, res) => {
   const lat = req.query.latitude; const
   lng = req.query.longitude; const
   maxDistance = 50;
  const station=await stationService.nearStations(lat, lng,
  maxDistance); logger.info("nearby station") logger.info(station);
   res.send(station);
```

```
});

const getStation = catchAsync(async (req, res) => {
  const station = await stationService.getStationById(req.params.stationId); if
  (!station) {
    throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
  } logger.info(station);
  res.send(station);
});

const deleteStation = catchAsync(async (req, res) => {
  const station = await stationService.getStationById(req.params.stationId); if
  (station)
   await stationService.deleteStationById(req.params.stationId);
  logger.info(station); res.status(httpStatus.NO_CONTENT).send("Deleted");
});

const updateStation = catchAsync(async (req, res) => {
  const stationOld = await stationService.getStationById(req.params.stationId); if
  (!stationOld) {
    throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
  } const station = await stationService.updateStationById(stationOld,
  req.body); logger.info(station);
     res.status(httpStatus.CREATED).send(
    {                              station_name:
     station.station_name,
    station_address:         station.station_address,         station_area:
    station.station_area,          station_city:          station.station_city,
    station_pincode:       station.station_pincode,       station_latitude:
    station.station_latitude,                              station_longitude:
    station.station_longitude,                        station_working_time:
    station.station_working_time,          station_available_connector:
    station.station_available_connector,        station_connector_type:
    station.station_connector_type,                 station_charger_type:
```

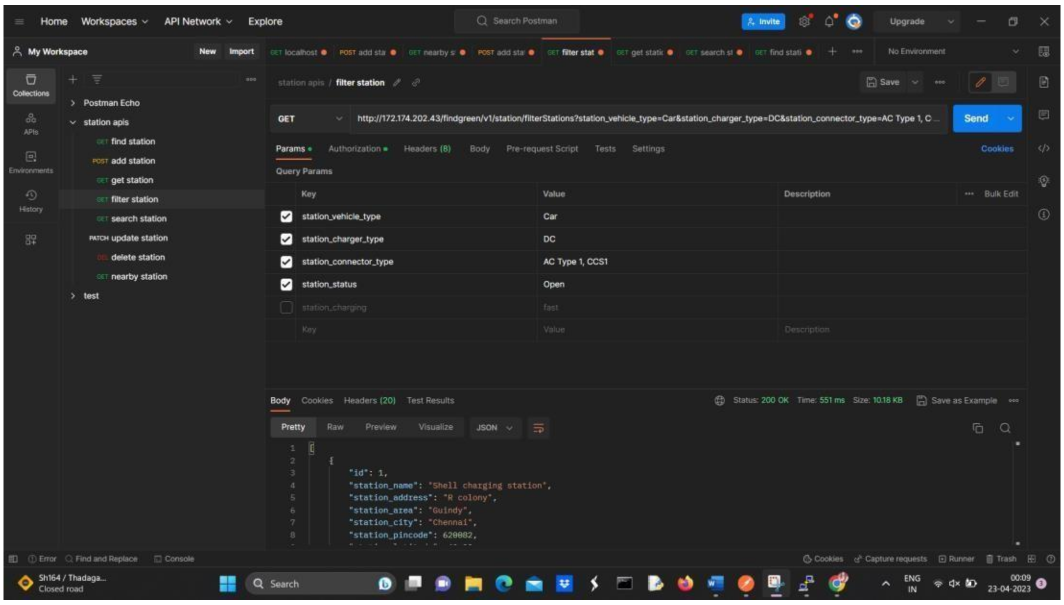
```
    station.station_charger_type,                    station_charging:
    station.station_charging,                     station_vehicle_type:
    station.station_vehicle_type,                        station_status:
    station.station_status,
    }
   );
 });

 module.exports = {
   createStation,
   searchStations,
   filterStations,
   getStation,
   deleteStation,
   updateStation,
   nearStation,
   findStations,
 };
```

//station.controller.test.js

```
const request = require('supertest'); const faker =
require('faker'); const httpStatus = require('http-
status'); const app = require('../../src/app'); const
db = require('../../src/models'); const config =
require('../../src/config/config'); const { invalid }
= require('joi'); const { ne } =
require('faker/lib/locales'); const ApiError =
require('../../src/utils/ApiError');

const { station_master } = db;

describe('Station Controllers', () => {
  let token =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOjksImlhdCI6MTY3ODIx
MTk2NCwiZXhwIjoxNjgwODAzOTY0LCJ0eXBlIjoicmVmcmVzaCJ9.dR5gfE
vc7TjyBghK3gBvfdGgs7TWD9ajZklK7hqPUiA';

 describe('createStation', () => {
  let newStation;

  test('should return 201 and successfully create new station if data is ok', async
() => { newStation
    = {
     station_name: "shell ev charging station",
     station_address: "professor colony",
     station_area: "seliyur", station_city:
     "chennai", station_pincode: "620082",
     station_latitude:"27.28",
     station_longitude:"29.6",
     station_working_time: "5-11",
     station_available_connector: "3",
     station_connector_type: "AC Type 1",
     station_charger_type: "ac",
     station_charging: "fast",
     station_vehicle_type: "bike",
     station_status: "available"
    };
    const res = await request(app)
     .post('/v1/station/add-station')
     .set('x-access-token', `${token}`)
     .send(newStation)
     .expect(201);
  });
 });

 describe('getStation', () => {
  test('should return 200 and the station object if data is ok', async () => {
      let stationId=450;
      const res = await request(app)
        .get(`/v1/station/getStation/${stationId}`)
        .set('x-access-token', `${token}`)
        .send()
        .expect(() => { if
          (!stationId) {
            throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
          } else
          expect(404);
        });
    });
  });

describe('deleteStation', () => {


    test('should return 204 if data is ok', async () => {
      let stationId = 5; const res =
      await request(app)
        .get(`/v1/station/deleteStation/${stationId}`)
        .set('x-access-token', `${token}`)
        .send()
        .expect(() => { if
          (stationId) {
          expect(204);
          }
          else

            throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
    });
  });
  describe('updateStation', () => {
    test('should return 200 and successfully update station if data is ok', async () =>
  { const updateBody = {
        station_name: "find ev charging station",
        station_address: "professor colony",
     station_area: "tambaram", station_city:
     "chennai", station_pincode: "620081",
     station_latitude:"23.28",
     station_longitude:"25.6",
     station_working_time: "6-10",
     station_available_connector: "2",
     station_connector_type: "AC Type 2",
     station_charger_type: "dc",
     station_charging: "slow",
     station_vehicle_type: "car",
     station_status: "busy",
    }; let stationId=10; const res
    = await request(app)
    .get(`/v1/station/getStation/${stationId}`)
    .set('x-access-token', `${token}`)
    .send()
    .expect(() => { if
       (!stationId) {
        throw new ApiError(httpStatus.NOT_FOUND, 'Station not found');
       } else
       { const re =
        request(app)
        .patch(`/v1/station/upd
        ateStation/${stationId
        }`)
       .set('x-access-token', `${token}`)
       .send(updateBody)
       .expect(201);
       }
     });
  });
});

describe('searchStations', () => {
  test('should return 200 and the station object if data is ok', async () => {


    let stationSearch="tambaram"; let
    station_pincode=620081;

    const res = await request(app)

.get(`/v1/station/searchStations?stationSearch=${stationSearch}&station_pincode
=${station_pincode}`)
      .set('x-access-token', `${token}`)
      .send()
      .expect(200);
  });
 });

 describe('findStations', () => {
  test('should return 200 and the station object if data is ok', async () => {
    let  stationSearch="tambaram";
    let station_pincode=620081; let
    station_vehicle_type= "car"; let
    station_charger_type="dc";  let
    station_connector_type=   "AC
    Type  2";  let  station_status=
    "busy";  let  station_charging=
    "slow";  const  res  =  await
    request(app)
.get(`/v1/station/findStations?stationSearch=${stationSearch}&station_pincode=$
{station_pincode}&station_vehicle_type=${station_vehicle_type}&station_charg
er_type=${station_charger_type}&station_connector_type=${station_connector_t
ype}&station_status=${station_status}&station_charging=${station_charging}`)
.set('x-access-token', `${token}`)
      .send()
      .expect(200);
  });
  });

  describe('filterStations', () => {
    test('should return 200 and the station object if data is ok', async () => {

      let station_vehicle_type= "car"; let
      station_charger_type="dc"; let
      station_connector_type= "AC Type 2"; let
      station_status= "busy"; let
      station_charging= "slow"; const res =
      await request(app)
.get(`/v1/station/filterStations?station_vehicle_type=${station_vehicle_type}&stat
ion_charger_type=${station_charger_type}&station_connector_type=${station_c
onnector_type}&station_status=${station_status}&station_charging=${station_ch
arging}`)
        .set('x-access-token', `${token}`)
        .send()
        .expect(200);
    });
  });

  describe('nearStations', () => {

    let nearby;

    test('should return 200 and the station object if data is ok', async () => {

      let latitude="20.28"; let
      longitude="24.6";

      nearby={
        latitude:"20.28",
        longitude:"24.6",
        maxDistance: 50,
      }

      const res = await request(app)
        .get(`/v1/station/nearStation?latitude=${latitude}&longitude=${longitude}`)
        .set('x-access-token', `${token}`)
        .send(nearby)
        .expect(200);
    });
  });
});
```

Appendix II (Screenshots)

Postman Screen



GET http://172.174.202.43/findgreen/v1/station/filterStations?station_vehicle_type=Car&station_charger_type=DC&station_connector_type=AC Type 1, C ...

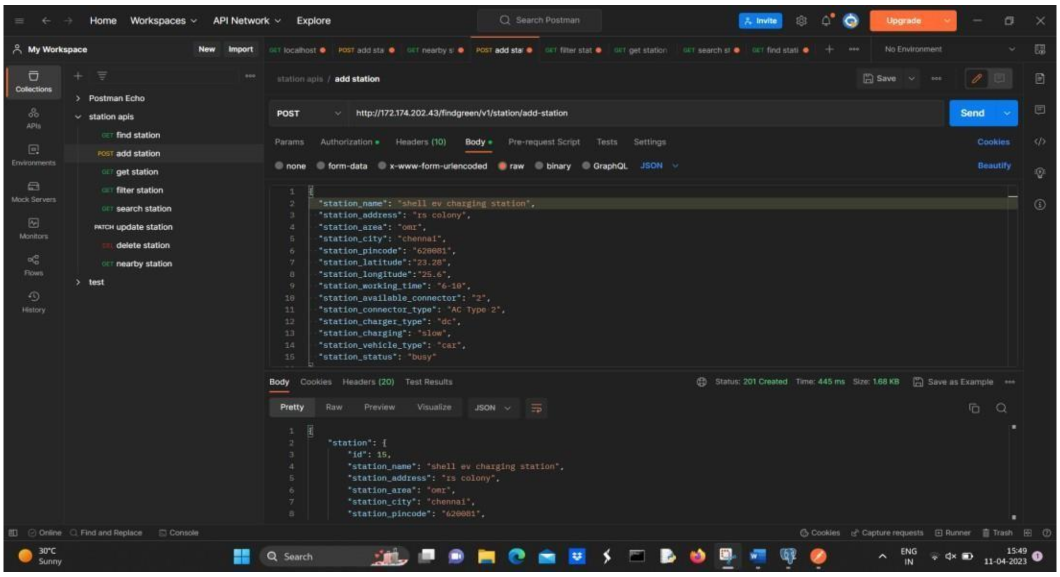
Query Params

	Key	Value	Description
✓	station_vehicle_type	Car	
✓	station_charger_type	DC	
✓	station_connector_type	AC Type 1, CCS1	
✓	station_status	Open	
	station_charging	fast	

Body Cookies Headers (20) Test Results — Status: 200 OK Time: 551 ms Size: 10.18 KB

```
[
    {
        "id": 1,
        "station_name": "Shell charging station",
        "station_address": "R colony",
        "station_area": "Guindy",
        "station_city": "Chennai",
        "station_pincode": 620082,
```

Fig A-2.1 Filter station API in Postman



```
POST  http://172.174.202.43/findgreen/v1/station/add-station
```

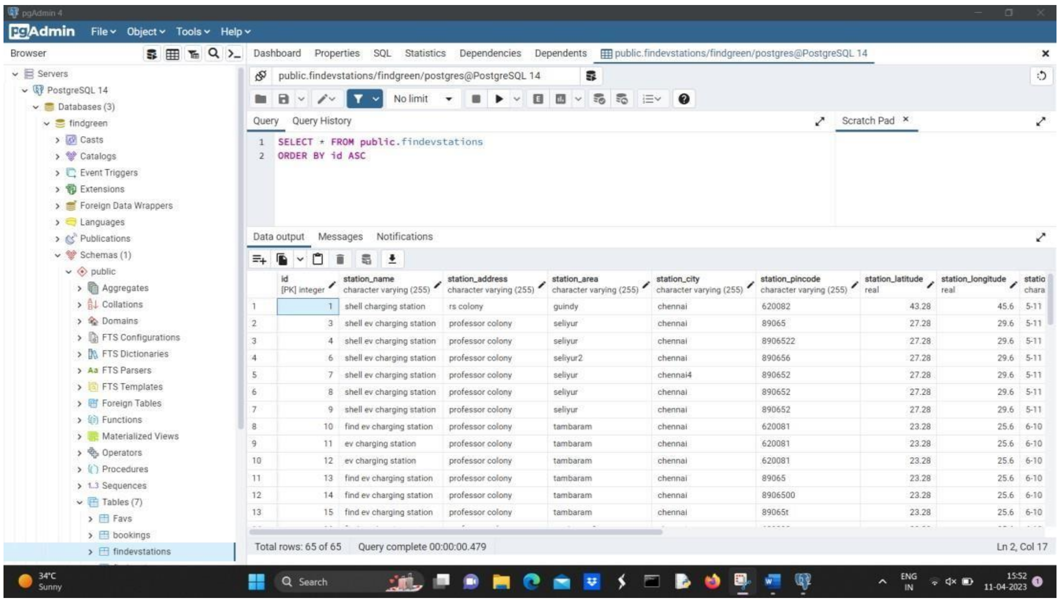
```
{
 "station_name": "shell ev charging station",
 "station_address": "rs colony",
 "station_area": "omr",
 "station_city": "chennai",
 "station_pincode": "620081",
 "station_latitude":"23.28",
 "station_longitude":"25.6",
 "station_working_time": "6-10",
 "station_available_connector": "2",
 "station_connector_type": "AC Type 2",
 "station_charger_type": "dc",
 "station_charging": "slow",
 "station_vehicle_type": "car",
 "station_status": "busy"
```

```
Status: 201 Created   Time: 445 ms   Size: 1.68 KB
```

```
{
    "station": {
        "id": 15,
        "station_name": "shell ev charging station",
        "station_address": "rs colony",
        "station_area": "omr",
        "station_city": "chennai",
        "station_pincode": "620081",
```

Fig A-2.2 Add station API in Postman

pgAdmin Screen



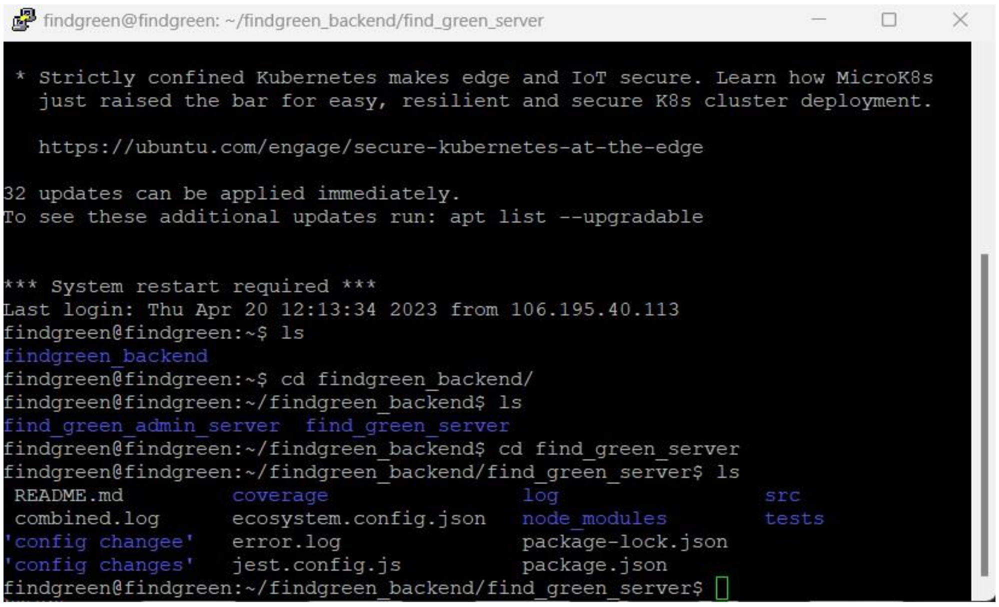
```
SELECT * FROM public.findevstations
ORDER BY id ASC
```

	id [PK] integer	station_name character varying (255)	station_address character varying (255)	station_area character varying (255)	station_city character varying (255)	station_pincode character varying (255)	station_latitude real	station_longitude real	statio chara
1	1	shell charging station	rs colony	guindy	chennai	620082	43.28	45.6	5-11
2	3	shell ev charging station	professor colony	seliyur	chennai	89065	27.28	29.6	5-11
3	4	shell ev charging station	professor colony	seliyur	chennai	8906522	27.28	29.6	5-11
4	6	shell ev charging station	professor colony	seliyur2	chennai	890656	27.28	29.6	5-11
5	7	shell ev charging station	professor colony	seliyur	chennai4	890652	27.28	29.6	5-11
6	8	shell ev charging station	professor colony	seliyur	chennai	890652	27.28	29.6	5-11
7	9	shell ev charging station	professor colony	seliyur	chennai	890652	27.28	29.6	5-11
8	10	find ev charging station	professor colony	tambaram	chennai	620081	23.28	25.6	6-10
9	11	ev charging station	professor colony	tambaram	chennai	620081	23.28	25.6	6-10
10	12	ev charging station	professor colony	tambaram	chennai	620081	23.28	25.6	6-10
11	13	find ev charging station	professor colony	tambaram	chennai	89065	23.28	25.6	6-10
12	14	find ev charging station	professor colony	tambaram	chennai	8906500	23.28	25.6	6-10
13	15	find ev charging station	professor colony	tambaram	chennai	89065t	23.28	25.6	6-10

```
Total rows: 65 of 65   Query complete 00:00:00.479
```

Fig A-2.3 Station Details in pgAdmin

Putty Server



```
 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

32 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


*** System restart required ***
Last login: Thu Apr 20 12:13:34 2023 from 106.195.40.113
findgreen@findgreen:~$ ls
findgreen_backend
findgreen@findgreen:~$ cd findgreen_backend/
findgreen@findgreen:~/findgreen_backend$ ls
find_green_admin_server  find_green_server
findgreen@findgreen:~/findgreen_backend$ cd find_green_server
findgreen@findgreen:~/findgreen_backend/find_green_server$ ls
 README.md          coverage                  log                src
 combined.log       ecosystem.config.json     node_modules       tests
'config changee'    error.log                 package-lock.json
'config changes'    jest.config.js            package.json
findgreen@findgreen:~/findgreen_backend/find_green_server$
```

Fig A-2.4 Putty Server

REFERENCES

[1] https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

[2] https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm

[3] https://www.baeldung.com/postman-upload-file-json

[4] https://www.tutorialspoint.com/postgresql/postgresql_expressions.htm [5] https://www.digitalocean.com/community/tutorials/how-to-use-sequelize- with-node-js-and-mysql