GoPlugable Code Documentation

Suhail AK

04 June 2024

Version 1.0

Contents

1 Introduction

1.1 Project Overview

This is a web-based project for finding and renting out EV Chargers. There has been a big increase in the usage of electronic vehicles in recent times. An EV owner would have a charger for charging their vehicles. However, at most times these chargers are not used since the vehicles need to be charged once or twice. So that leads to the idea of sharing their chargers as charging stations whereas other users can also utilize the chargers for a reasonable price. This is the basic idea of this project. The users can register in this application as hosts to rent out their chargers or as users to use other's chargers for charging. The application lists available chargers on the map according to their location.

The mobile Application is developed in the Flutter framework using Dart language. The APIs for the project are done in the Laravel framework using PHP language. There are 2 types of users for mobile applications which are Hostee and Chargee. The web application contains only one user which is the admin who manages, monitors and controls everything regarding the Application.

1.2 Scope of Document

This document contains the complete details of the Laravel admin project. This project has both an admin dashboard and APIs for mobile applications. This document will have the project structure, code comments, feature-wise comparisons, route details, helper class details, model details and more. This document would be a helpful reference document for the clients of what has been done and for the mobile and app developers who will be maintaining this project.

This document is intended to be used by the following audience:

1. Client: To provide a clear understanding of the project
2. Backend Developers: To assist and extend the entire codebase
3. Mobile Developers: To understand API endpoints and their usage
4. Project Managers: To get an overview of the project's architecture and features

1.3 Document Navigation

This document will contain the details about the entire codebase so this document would be pretty lengthy and huge. The document will include a table of contents with page numbers for easy navigation through pages. The document will be code-based so there would be issues in finding the appropriate code, so there will be an index or appendix based on common topics in the last of the document. The user can refer to the topic and corresponding page numbers to find the correct code easily. This document would contain the following sections which will make the navigation easier.

1. **Table of Contents:** The user can locate the major sections and topics within the document. Each will have a corresponding page number.
2. **Sections and Subsections:** The document is organised into sections and subsections for easier navigation.
3. **Index or Appendices:** The list would have an index based on topics in alphabetical order.

2 Installation and setup

2.1 Prerequisite

This project is developed in laravel framework using PHP language. There are some prerequisites and tools for installation and running the project.

1. PHP (Minimum 7.4 version)
2. MySQL
3. Composer
4. Database File (.sql format)
5. Source Code
6. Code Editor
7. Git

2.2 Installation

This project can be installed and run in your environment just by following the step by step instructions mentioned below.

1. Download the source code using git from the master branch for final and production code.
2. Download the database file.
3. Download the xampp from the official website and run the installation according to the operating system of your choice.
4. Start the apache server and mysql server
5. Create a new database by using either phpmyadmin or mysql terminal.
6. Open the project folder and rename .env.example file to .env
7. Install the packages and other files dependent to the project using composer. This can be done using command:
   composer install
   This will create a folder named vendor and there would be all the packages and dependencies needed for this project.
8. Update the database credentials, other credentials and confidential secret keys in the .env file
9. Import the sql file using phpmyadmin if you want tables with existing data or just run the command **php artisan migrate** to migrate and create all the tables from scratch.
10. If there are some pre-defined data, It should be added to the database using the command
    php artisan db:seed
11. There may be uploads or files in the storage folder, this should be made available to public folder by creating system links. This can be done using the following command
    php artisan storage:link
12. The project can be run in 2 ways. First one is to run it continously by placing file inside htdocs (steps 13-15) and the second one is run temporarily using command. (steps 11-12)
13. The project can be run using the command, **php artisan serve**. When this command is executed, a link is generated in the console.
14. The link can be used to run the project in the browser or the API.
15. If you want to run the project continuously other than running using artisan command, place the project folder inside the htdocs folder of the xampp installation directory.
16. After placing the project directory inside the htdocs folder, the project would be completely accessible using the link *http://localhost/foldername*

17. A normal project developed in laravel should be always run or invoked from the public folder inside the project folder, In this case a .htaccess file is included to shift the document root from public/ to the root of the project

2.3 Configuration

It is important to configure all the details needed for running the project. All the credentials for database, payment gateway, messaging and other dependencies related to the project. In laravel, the configuration is done in the .env file which was already mentioned in the installation section.

The configurations that are relevant to this project will be added here one by one for clear understanding.

1. **APP_NAME:** This denotes the name of the laravel project or application.
2. **APP_ENV:** This denotes the environment of the laravel project or application. The values could be dev, staging or production.
3. **APP_KEY:** This is a key that is generated to identify the project. This would be created by default. In case, there are no keys configured you can configure it using running command
   php artisan key:generate
4. **APP_URL:** This denotes the value of the website link in the application. These are used in the application for file system operations.
5. **LOG_CHANNEL:** This denotes the logging channel of the application. This is used for logging the errors, warning and messages to the Log file. This value is not changed much.
6. **LOG_DEPRECATIONS_CHANNEL:** This denotes the warnings that certain features or functionalities within a software system will be removed in future versions. This value is not changed much.
7. **LOG_LEVEL:** This denotes the level of logging in the Application. The logging level could be debug or production. "debug" is used to get all the errors and warning more than "production" level debugging.
8. **DB_CONNECTION:** This key denotes the database connection or query language which is used in the application. This could be Mysql, Posgress, Mongo or Sqlite depending on user's choice.
9. **DB_HOST:** This denotes the value of the location where the database used in the application is hosted or stored. The normal implementation would be implementing the application and database in same server which leads to the usage of "localhost".
10. **DB_PORT:** This denotes the value of the port which allows to connect to the database to the host. The common port would be like 3306 for connecting to the database host
11. **DB_DATABASE:** This denotes the value of name of the database used in the application.
12. **DB_USERNAME:** This denotes the value of user name of the user account associated with the database used in the application.
13. **DB_PASSWORD:** This denotes the value of user password of the user account associated with the database used in the application.
14. **BROADCAST_DRIVER:** This denotes the driver value of broadcast messaging platform in the application.
15. **CACHE_DRIVER:** This denotes the cache mechanism used in the laravel application. Normally, if the value is used as file to ensure caching is done in filesystem.
16. **FILESYSTEM_DISK:** This value denotes the filesystem used in laravel application for storage of uploaded files and the operation of files. Normally, the value local is used for local filesystem and we can change it for use with other filesystems such as s3, etc.
17. **QUEUE_CONNECTION:** This denotes the queueing mechanism used in the laravel application. The tasks can be added to queue to run asyncronously along with the execution, This allows that async operations.
18. **SESSION_DRIVER:** This denotes the session used in the laravel application. A session starts when the user is authenticated using authentication middleware.

19. **SESSION_LIFETIME:** This denotes the lifetime of a valid session. The session would expire after the validity or lifetime of session expires.
20. **MEMCACHED_HOST:** This enables a centralised RAM store which stores the cache of the laravel application. The host value of the MEMCACHE STORE is added here. Common value is Localhost URL
21. **REDIS_HOST:** Redis is a platform to store key-value based storage database. The URL to the database is added here.
22. **REDIS_PASSWORD:** The password to access the Redis key store is added here.
23. **REDIS_PORT:** The port used to access the Redis key store is added here.
24. **MAIL_MAILER:** The email service which is used for the email services and operations is mentioned here.
25. **MAIL_HOST:** The email service host where the email services is available. While sending emails the SMTP connection is established with the host first. (Ezample Host: **smtp.gmail.com**)
26. **MAIL_PORT:** The port used to connect to the email service host is mentioned here. Commonly used email port numbers are 465, 587 etc.
27. **MAIL_USERNAME:** The username of the account associated with the email service is mentioned here. The username would be a email address normally.
28. **MAIL_PASSWORD:** The password of the account associated with the email service is added in this key.
29. **MAIL_ENCRYPTION:** The encryption method to connect to the email service host is added here. This would be either TLS or SSL.
30. **MAIL_FROM_NAME:** The name of the email sender is added here so it would show in from email recieved to a customer. This is either added as The application name or the name of persons/
31. **MAIL_FROM_ADDRESS:** The email address where the email is sent from is added here.
32. **AWS_ACCESS_KEY_ID:** The access key details for the AWS S3 storage bucket is added here.
33. **AWS_SECRET_ACCESS_KEY:** The secret key associated with the access key for the AWS S3 storage bucket is added here.
34. **AWS_DEFAULT_REGION:** The default region for the AWS S3 storage bucket is added here.
35. **AWS_BUCKET:** The AWS S3 storage bucket name is added here.
36. **AWS_USE_PATH_STYLE_ENDPOINT:** The end point details for the aws path style is added here.
37. **PUSHER_APP_ID:** The application id for the pusher service is mentioned here. The pusher is a service to keep running connections between client and server. Also used to send broadcast messages or webhooks.
38. **PUSHER_APP_KEY:** The access key details for the pusher service account is added here.
39. **PUSHER_APP_SECRET:** The secret key associated with app key for the pusher service account is added here.
40. **PUSHER_HOST:** The host details of where the pusher service is accessed or connected is added here.
41. **PUSHER_PORT:** The port to connect to the pusher service account is added here. Common port value will be 443.
42. **PUSHER_SCHEME:** The connection scheme associated with the pusher service account is added here. Common value will be https since the port number is 443.
43. **PUSHER_APP_CLUSTER:** The cluster details associated with the pusher service account is added here.
44. **JWT_SECRET:** The secret key associated with the JWT Authentication token. We need to generate a secret key.

45. **FIREBASE_SERVER_KEY:** The server key of the configured firebase account for the application. This key is acquired after configuring the project in the Firbase console.

46. **STRIPE_KEY:** The key associated with the stripe payment gateway services. This key and secret key is generated from the stripe payment gateway dashboard after completing all necessary verifications and payments.

47. **STRIPE_SECRET:** The secret key associated with the stripe key for the stripe payment gateway. This secret key is generated from the payment gateway dashboard.

48. **TWILIO_PHONE_NUMBER_UK:** The phone number which is associated with the twilio account. Twilio is used for sending verification OTP SMS and other marketing SMS. The number is purchased from twilio dashboard.

49. **TWILIO_SID:** The Service ID associated with our twilio account is added here.

50. **TWILIO_ACCOUNT_SID:** The Service ID associated with our twilio account is added here.

51. **TWILIO_AUTH_TOKEN:** The Authentication token associated with the service ID from twilio is added here.

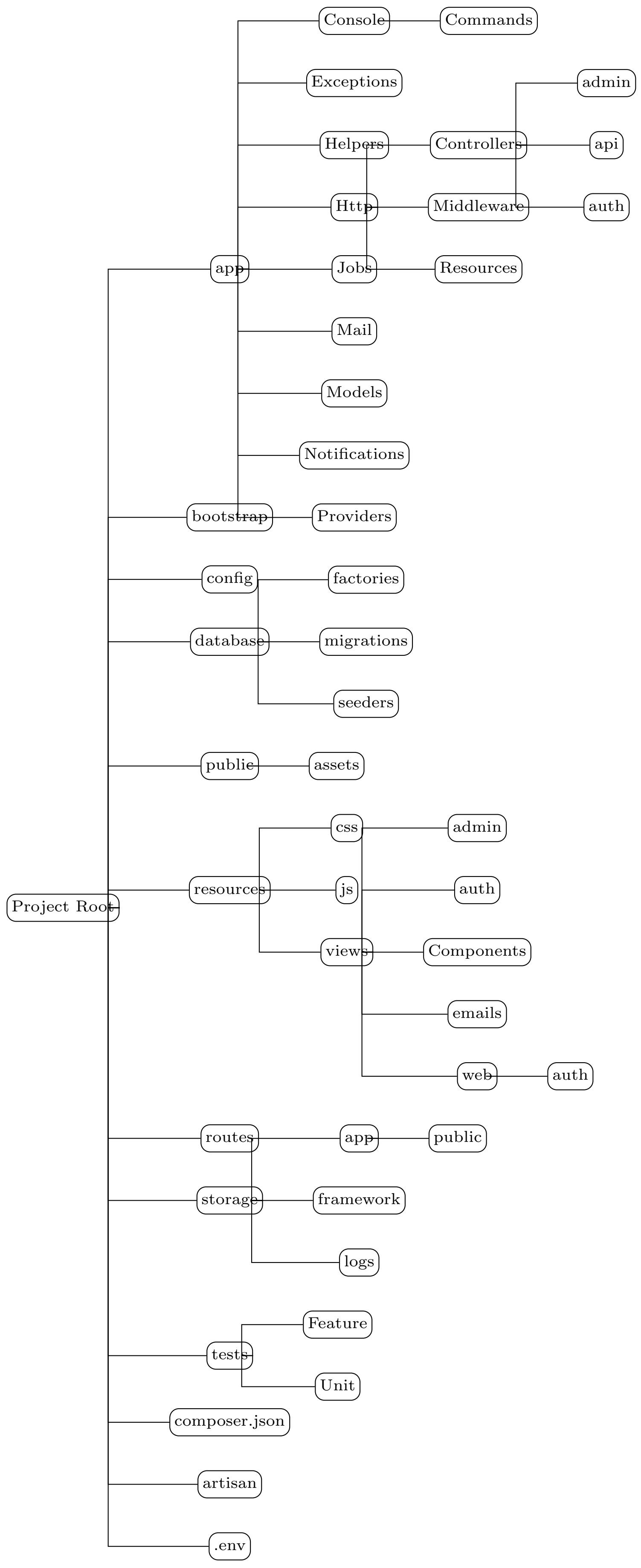
52. **TWILIO_PHONE_NUMBER:** The phone number which is associated with the twilio account.

53. **GOOGLE_MAPS_LOCALE:** The Locale or language associated with the google maps is added here.

54. **GOOGLE_MAPS_API_KEY:** The Api key associated with the maps api which was generated from the google cloud console. This key will be used for google maps integration in our application.

3 Project Structure

3.1 Directory Structure



3.2 Important Files and Directories

3.2.1 app folder

The app folder contains some sub folders related to the application.

1. Console - This folder contains the commands folder, custom created commands, kernel file to execute the cron jobs or scheduled tasks.
2. Exceptions - This folder contains the exception handlers in the application.
3. Helpers - This folder contains commonly used helper classes and functions for static references across the application.
4. Http - This folder contains the main controllers, middleware and resources related to the web. This contains the core part of application.
5. Jobs - This folder contains the cron jobs created for this application. This jobs can be scheduled to run at a specific interval or at a specified time.
6. Mail - This folder contains all the mail classes that was created for this application. The email formats or values passed to the email template is defined here.
7. Models - This folder contains all the models of the application. A model is considered as the basic element which contains properties, relationship and methods of the application.
8. Notifications - This folder contains notifications that are sent to the mobile application or the api.
9. Providers - This folder contains the service Provider details of the application.

3.2.2 bootstrap folder

This folder contains the default design files of the laravel project. But this only comes to use in case of using npm mix. Haven't used it in this project.

3.2.3 config folder

This folder contains all the necessary configurations of the packages, dependencies etc. used in our project. This includes configuration for the application, authentication, cache, cors, hashing, geocoder etc.

3.2.4 database folder

This folder contains all the files related to the database.

1. factories - The factories are used to create bulk fake data for the models. This is mainly used for testing purposes
2. migrations - The migrations are files that are used to define the table structure for the database. These files are migrated to database using the command **php artisan migrate**
3. seeders - The seeders are used to seed or add predefined data to the database. Like the admin login credentials, predefined countries and states etc.

3.2.5 public folder

This folder contains all the assets and other common components used in the application. The document root or main starting point of the application is inside this folder.

3.2.6 resources folder

This folder contains all the resources of the application. This folder contains the style files, views of the application. Views are the screens or designs the user view when navigating in website.

3.2.7 routes folder

This folder contains the routes used to navigate in the application. There are 2 types of routes by default.

1. **api.php** - The API routes that is used for the mobile application.
2. **web.php** - The routes that is used for the web application.

3.2.8 storage folder

This folder contains the files that are uploaded by the users. This folder also contains the cache and logs of the web application.

1. **app folder -** The app/public folder in this directory contains all the uploaded files and newly created files by the user. This can be linked to the public folder for easy access by linking them using command **php artisan storage:link**
2. **framework folder -** This folder is auto-generated by the laravel framework. This contains the cache, sessions and testing data for the application.
3. **logs folder -** This folder is also auto-generated by the laravel framework. This contains the logs of errors occurred in the application. This can also be used for manually logging the data to check by users.

3.2.9 tests folder

This folder contains the test information for the application. This contains testcases for both unit testing and feature wise testing. There won't be any test cases by default.

3.2.10 composer file

The file **composer.json** contains the information about the project and all the dependencies needed for this project. The packages would be installed or updated once the command **php artisan install** is executed. The composer should be installed in the system before running this. The setup and installation of composer is mentioned in section 2.2

3.2.11 artisan file

The **artisan file** is the most important and basic file in the laravel project. It is used to execute all the inline commands in the application. artisan file is auto generated by laravel framework and there is no need to change anything in this file.

3.2.12 .env file

The **.env file** contains all the information, secret keys and configuration regarding the application. All the credentials for database, payment gateway, messaging and other dependencies related to the project are configured in this file.

4 Code Documentation

This section includes the complete documentation of the project. This will include models, Controllers, Routes, Middlewares and Helper Classes. Each of them will have details like File Location, purpose and line numbers.

4.1 Models

4.1.1 BatteryWatts.php

This model contain details about the power of the battery of Electric Vehicle (EV).

File Location: app/Models/

4.1.2 Booking.php

This model contains the information about the booking of slots for EV Charging.

File Location: app/Models/

This include properties like:

1. cs_id: The id of charging station which the booking is made.
2. user_id: The id of user which books the charging station.
3. booking_time: The time for which the booking has been made. (Schedule time)
4. booking_duration: How long should the booking last. (Estimated Time)
5. booking_end_time: The end time of the booking. (Estimated Time)
6. booking_charging_point: The charging point details of the charging station.
7. booking_status: The status of the booking
8. charging_start_time: The actual starting time of the charging
9. charging_end_time: The actual ending time of the charging
10. booking_vehicle_id: The id of the vehicle which is charging
11. deleted_at: The timestamp when the booking information is deleted
12. charging_start_percentage: The actual percentage which was available when starting the charging process
13. charging_end_percentage: The actual percentage when the charging process is stopped or completed
14. charging_start_photo: The filename or path to the image which is taken at the starting of the charging
15. charging_end_photo: The filename or path to the image which is taken at the completion of the charging

This model includes the relationships with other models. They are:

1. user() - This defines the relation with User Model. Defines which user created the booking. The column used for reference is user_id
2. vehicle() - This defines the relation with VehicleModel Model. Defines which vehicle is used for the booking. The column used for reference is booking_vehicle_id
3. chargingStation() - This defines the relation with ChargingStation Model. Defines which charging station is used for the booking. The column used for reference is cs_id
4. chargingPoint() - This defines the relation with CsChargingPoint Model. Defines which charging point inside the charging station is used for the booking. The column used for reference is booking_charging_point
5. details() - This defines the relation with BookingDetail Model. Defines more details about the booking. The column used for reference is booking_id in BookingDetail model and id in this Model

4.1.3 BookingDetail.php

This model contains the detailed information about the booking.

File Location: app/Models/

This model have properties like:

1. booking_id: This defines the id of the booking. Refers to the Model Booking
2. total_amount: This defines the total amount for the corresponding booking.
3. parking_amount: This defines the parking amount in the charging station.
4. minimum_amount: This defines the minimum chargeable amount set by the charging station.
5. discount_amount: This defines the discount amount available for the corresponding booking.
6. total_kw: This defines the total energy consumed for the corresponding booking.
7. per_kw: This defines the amount charged per KW for the corresponding booking.
8. deleted_at: This defines the timestamp where corresponding booking is deleted.
9. vat_amount: This defines the Value Added Tax (VAT) amount for the corresponding booking.
10. host_wallet: This defines the commission amount to the host on the corresponding booking.
11. connection_amount: This defines the amount for the connection or using the charging station for the corresponding booking.
12. vat_percentage: This defines the percentage of VAT applied for the corresponding booking.
13. connection_charge: This defines the minimum chargeable amount for the connection or using the charging station set by the charging station.
14. host_currency: This defines the name of the currency used by the host.
15. host_currency_symbol: This defines the symbol of the currency used by the host.
16. user_currency_symbol: This defines the symbol of the currency used by the user.
17. user_currency: This defines the name of the currency used by the user.
18. conversion_rate: This defines the currency conversion rate of the host and user.

4.1.4 ChargerBrands.php

This model contains the details about the brands of chargers used in charging station.

File Location: app/Models/

This model have following properties:

1. charger_brand_name: This defines the brand name of the charger
2. battery_watts: This defines the power of the battery which can be charged using this charger.
3. charger_brand_status: This defines the current status of the charger
4. deleted_at: This defines the timestamp where corresponding charger is deleted.

This model have relationship with other models. They are:

1. watt() - This defines the relation with BatteryWatts Model. Defines power of battery it can charge. The column used for reference is battery_watts

4.1.5 ChargingPreDetail.php

This model contains the details which is stored for processing before adding to the Booking Details

File Location: app/Models/

This model have following properties:

1. booking_id: This defines the id of the booking. Refers to the Model Booking
2. total_amount: This defines the total amount for the corresponding booking.
3. parking_amount: This defines the parking amount in the charging station.
4. minimum_amount: This defines the minimum chargeable amount set by the charging station.
5. discount_amount: This defines the discount amount available for the corresponding booking.
6. total_kw: This defines the total energy consumed for the corresponding booking.
7. per_kw: This defines the amount charged per KW for the corresponding booking.
8. vat_amount: This defines the Value Added Tax (VAT) amount for the corresponding booking.
9. connection_amount: This defines the amount for the connection or using the charging station for the corresponding booking.
10. vat_percentage: This defines the percentage of VAT applied for the corresponding booking.
11. connection_charge: This defines the minimum chargeable amount for the connection or using the charging station set by the charging station.
12. host_currency: This defines the name of the currency used by the host.
13. host_currency_symbol: This defines the symbol of the currency used by the host.
14. user_currency_symbol: This defines the symbol of the currency used by the user.
15. user_currency: This defines the name of the currency used by the user.
16. conversion_rate: This defines the currency conversion rate of the host and user.

4.1.6 ChargingStation.php

This model contains the details about the charging station

File Location: app/Models/

This model have following properties:

1. host_id: Defines the id of the host or charging station owner
2. cs_name: Defines the name of the charging station
3. cs_location_name: Defines the name of the charging station location
4. cs_eircode: Defines the eir code of the charging station location
5. cs_latitude: Defines the latitude of the charging station location (geocoding of location)
6. cs_longitude: Defines the longitude of the charging station location (geocoding of location)
7. cs_about_us: Defines the about details of the charging station
8. cs_verified: Defines whether the charging station is verified or not
9. cs_featured: Defines whether the charging station is featured or not
10. cs_status: Defines the current status of charging station
11. deleted_at: This defines the timestamp where charging station is deleted.

This model have relationship with other models. They are:

1. host() - This defines relationship with User Model. Defines which user acts as station host or owner. The column used for reference is host_id
2. chargingPoint() - This defines relationship with CsChargingPoint Model. Defines which charging points are owned by the charging station. The column used for reference is cs_id in CsChargingPoint Model and id in this model
3. photos() - This defines relationship with CsPhoto Model. Defines the photo details of the charging station. The column used for reference is cs_id in CsPhoto Model and id in this model
4. proofs() - This defines relationship with CsProofs Model. Defines the proof details of the charging station. The column used for reference is cs_id in CsProofs Model and id in this model
5. highlights() - This defines relationship with CsLocationHighlight Model. Defines the highlights of the charging station location. The column used for reference is cs_id in CsLocationHighlight Model and id in this model
6. parkingInstructions() - This defines relationship with CsParkingInstruction Model. Defines the instructions for parking to follow in the charging station location. The column used for reference is cs_id in CsParkingInstruction Model and id in this model
7. charger() - This defines relationship with CsCharger Model. Defines the charger used charging station. The column used for reference is cs_id in CsCharger Model and id in this model
8. settings() - This defines relationship with CsSetting Model. Defines the settings of the charging station. The column used for reference is cs_id in CsSetting Model and id in this model
9. payments() - This defines relationship with CsPaymentDetails Model. Defines the payments of the charging station. The column used for reference is cs_id in CsPaymentDetails Model and id in this model
10. availabilityDays() - This defines relationship with CsAvailabilityDays Model. Defines the available working days of the charging station. The column used for reference is cs_id in CsAvailabilityDays Model and id in this model
11. parking() - This defines relationship with CsParkingTimingDay Model. Defines the parking timings on each days of the charging station. The column used for reference is cs_id in CsParkingTimingDay Model and id in this model

4.1.7 Chat.php

This model contains the chat between host and the user regarding booking or charging station.

File Location: app/Models/

This model have following properties:

1. chargee_id: The id of the chargee or normal user
2. cs_id: The id of the charging station
3. host_id: The id of the charging station owner or host
4. booking_id: The id of the booking done by user

This model have relationship with other models. They are:

1. cs() - This defines relationship with ChargingStation Model. Defines the chat is regarding with charging station. The column used for reference is cs_id

4.1.8 ChatItem.php

This model contains message of a chat between host and user regarding booking or charging station.

File Location: app/Models/

This model have following properties:

1. chat_id: The id of the chat between user and host
2. chat: The message text
3. user_read: The flag or indicator for checking whether the user read the message
4. user_id: The id of the user who sent the message
5. cs_read: The flag or indicator for checking whether the host read the message
6. booking_id: The id of booking which the chat is related to

4.1.9 Country.php

This mode contains the list of countries. Considered as the master data model.

File Location: app/Models/

This model have following properties:

1. country: The name of country

4.1.10 CountryWiseDistanceUnit.php

This mode contains the distance unit used for different countries.

File Location: app/Models/

4.1.11 County.php

This mode contains the list of counties. Considered as the master data model.

File Location: app/Models/

This model have following properties:

1. country: The name of country
2. state: The name of state
3. county: The name of county

4.1.12 CsAvailabilityDays.php

This model contains the availability of charging station on each days.

File Location: app/Models/

This model have following properties:

1. cs_id: The id of the charging station
2. day: The day of the week
3. time_from: The start time of availability
4. time_to: The end time of availability
5. day_type: The type of day

4.1.13 CsCharger.php

This model contains the information on chargers in charging station.

File Location: app/Models/

This model have following properties:

1. cs_id: The id of charging station
2. watt_id: The id of battery power (Watt)
3. brand_id: The id of charger brand
4. deleted_at: The timestamp when the charger is deleted

This model have relationship with other models. They are:

1. watts() - This defines relationship with BatteryWatts Model. Defines the power of battery. The column used for reference is watt_id
2. battery() - This defines relationship with ChargerBrands Model. Defines the Brand of the charger. The column used for reference is brand_id

4.1.14 CsChargingPoint.php

This model contains the details about the charging point inside the charging station.

File Location: app/Models/

This model have following properties:

1. cs_id: The id of the charging station
2. point_name: The name of the charging point in the charging station
3. point_port: The port of the charging point in the charging station
4. minimum_charge_applicable: Used to check whether the charging point have minimum charge or not
5. point_charge_per_kw: The amount charging point charges for charging per KW
6. point_minimum_charge: The minimum charge for using the charging point
7. deleted_at: The timestamp when the charging point is deleted

4.1.15 CsLocationHighlight.php

This model is used to store the location highlight details of the charging station.

File Location: app/Models/

This model have following properties:

1. cs_id: The id of the charging station
2. highlight_id: The id of the location highlight
3. deleted_at: The timestamp of when the location highlight is deleted

This model have relationship with other models. They are:

1. details() - This defines relationship with LocationHighlights Model. Defines the highlights of the location. The column used for reference is highlight_id

4.1.16 CsParkingInstruction.php

This model contains the instructions about parking thw vehicles in the charging station.

File Location: app/Models/

This model have following properies:

1. cs_id: The id of the charging station
2. parking_instruction: The instruction to park
3. deleted_at: The timestamp of when the instruction is deleted.

4.1.17 CsParkingTimingDay.php

This model contains the timings of parking in charging station on each days in a week.

File Location: app/Models/

This model have following properties:

1. cs_id: The id of the charging station
2. day: The day of the week
3. parking_from: The starting time of parking on the day
4. parking_to: The ending time of parking on the day
5. parking_charge: The amount charged for parking on the day
6. day_type: The type of the day

4.1.18 CsParkingTimings.php

This model contains the timings of parking in charging station

File Location: app/Models/

This model have following properties:

1. cs_id: The id of the charging station
2. cs_settings_id: The id of the charging station settings
3. parking_charge: The amount charged for parking in the charging station
4. parking_from: The starting time for parking in the charging station
5. parking_to: The ending time for parking in the charging station
6. deleted_at: The timestamp of when the parking timings are deleted

4.1.19 CsPaymentDetails.php

This model contains the payment details of the charging station

File Location: app/Models/

This model have following properties:

1. cs_id: The id of the charging station
2. account_holder_name: The name of the account holder of charging station bank account
3. payment_preference: The prefered payment method of charging station
4. account_currency: The currency of the charging station bank account

5. account_country: The country of the charging station bank account
6. account_number: The account number of the charging station bank account
7. account_holder_type: The account type of the charging station bank account
8. account_routing_number: The account routing number of the charging station bank account
9. iban_paypal: The IBAN number of the charging station bank account
10. user_id: The id of user associated with the charging station
11. dob: The date of birth of user associated with the charging station
12. gender: The gender of user associated with the charging station
13. deleted_at: The gender of user associated with the charging station

This model have relationship with other models. They are:

1. preference() - This defines relationship with PaymentPreference Model. Defines the payment preference. The column used for reference is payment_reference

4.1.20 CsPhoto.php

This model contains the photos of the charging station

File Location: app/Models/

This model have following properties:

1. host_id: The id of the host
2. cs_id: The id of the charging station
3. photo: The file name or location to the uploaded photo
4. photo_type: The type of the uploaded photo
5. deleted_at: The timestamp of when the photo is deleted

This model have relationship with other models. They are:

1. chargeStation() - This defines relationship with ChargingStation Model. Defines the charging station which photos belongs to. The column used for reference is cs_id

4.1.21 CsPointTurnOff.php

This model contains the information about turning off charging point for specific days

File Location: app/Models/

This model having following properties:

1. cs_id: The id of the charging station
2. point_id: The id of the charging point of charging station
3. date_from: The starting date for turning off the charging point
4. date_to: The ending date for stop turning off the charging point
5. status: The status for turning off the charging point
6. deleted_at: The timestamp when the turning off charging point

4.1.22 CsProofs.php

This model contains the proofs submitted by charging station

File Location: app/Models/

This model have following properites:

1. cs_id: The id of the charging station
2. proof: The filename or location the uploaded proof document
3. proof_type: The type of the uploaded proof document
4. deleted_at: The timestamp when proof document is deleted

4.1.23 CsRating.php

This model contains the rating information of the charging station from the users.

File Location: app/Models/

This model have following properties:

1. user_id: The id of the user who added rating
2. cs_id: The id of the charging station
3. comment: The comment on the charging station review
4. title: The title on the charging station review
5. rate: The star rating count on the charging station review
6. deleted_at: The timestamp when charging station review is deleted

This model have relationship with other models. They are:

1. user() - This defines relationship with User model. Defines which user added the rating. The column used for reference is user_id

4.1.24 CsSetting.php

This model contains the information about the settings of the charging station

File Location: app/Models/

This model have following properties:

1. cs_id: The id of charging station
2. approval_required: The check whether the user needs approval for charging
3. parking_charge: The check whether parking charge is collected or not
4. parking_charge_ph: The check whether parking charge is collected in hourly rate or not
5. always_on: The check whether charging station is always available or not
6. service_mode: The check whether charging station is in service mode or not
7. deleted_at: The timestamp when the details are deleted

4.1.25 CsTurnOff.php

This model contains the information about the turning off of charging ports in the charging station

File Location: app/Models/

This model have following properties:

1. cs_id: The id of charging station
2. date_from: The starting date of turning off charging station
3. date_to: The ending date of turning off charging station
4. ports: The ports which are turning off in charging station
5. status: The status of charging station
6. deleted_at: The timestamp of when this information is deleted

4.1.26 Currency.php

This model contains the information about the currencies used for transaction.

File Location: app/Models

This model have following properties:

1. currency: The currency used
2. currency_icon: The icon of the currency used
3. country: The country where the currency is used

4.1.27 CurrencyConvertorLog.php

This model contains logging information on the currency conversion API

File Location: app/Models

This model have following properties:

1. user_id: The id of the user doing transaction
2. wallet_from: The id of the sender's wallet
3. wallet_to: The id of the receiver's wallet
4. currency_from: The currency used by the sender
5. currency_to: The currency used by the receiver
6. currency_rate: The currency conversion rate
7. api_response: The response JSON received from the API

4.1.28 Discount.php

This model contains the information about discounts

File Location: app/Models

This model have following properties:

1. user_id: The id of the user who created the discount
2. discount_type: The type of the discount. Can be flat discount or percentage
3. discount: The amount or percentage of the discount

4. discount_code: The code used to avail the discount
5. discount_validity: The check whether the discount have validity or not
6. discount_start: The start time of a discount if it is a time based discount
7. discount_end: The end time of a discount if it is a time based discount
8. discount_status: The current status of the discount
9. discount_number_usage: Defines how many users have used the discount
10. deleted_at: The timestamp of when the discount is deleted
11. limit: Defines how many users can use the discount

4.1.29 DiscountAppliedHistory.php

This model defines the history of the users who have applied the discount

File Location: app/Models

This model have following properties:

1. user_id: The id of the user who applied the discount
2. discount_id: The id of the used discount

4.1.30 EmailSettings.php

This method contains information about the email account for smtp

File Location: app/Models

This model have following properties:

1. smtp_host: The link to email provider host
2. smtp_port: The port used to connect to the email provider host
3. smtp_username: The email id or username associated with the email account
4. smtp_password: The password associated with the email account
5. smtp_encryption: The encryption protocol associated with the email account
6. from_address: The sender email address
7. from_name: The sender name

4.1.31 EmailTemplate.php

This model is used to store the templates used for sending email.

File Location: app/Models

This model have following properties:

1. template_name: The name given to template for identifying
2. subject: The email subject to include in email
3. body: The email body content to include in email

4.1.32 LocationHighlights.php

This model contains information about the location highlights

File Location: app/Models

This model have following properties:

1. lh_name: The name of the location highlight
2. lh_image: The image of the location highlight
3. lh_status: The status of the location highlight
4. deleted_at: The timestamp of when the location highlight is deleted

4.1.33 MyReferral.php

This model contains information about the referral

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. friend_id: The id of the referred person
3. wallet: The id of the wallet
4. history_id: The id of the referral history

4.1.34 Notification.php

This model contains information about the push notifications

File Location: app/Models

This model have following properties:

1. user_id: The id of the user to receive notification
2. booking_id: The id of the booking associated with notification
3. notification_title: The title of the notification
4. notification_description: The description of the notification
5. notification_read: The status whether the notification is read or not
6. deleted_at: The timestamp of when the notification is deleted

4.1.35 Payment.php

This model contains information about the payment

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. payment_id: The id of the payment transaction
3. payment_amount: The amount of the payment transaction
4. payment_status: The status of the payment transaction
5. payment_method: The mode of the payment transaction

6. pg_id: The id of the payment transaction used in payment gateway
7. pg_response: The response of the payment from the payment gateway
8. wallet_before: The balance in the wallet before the payment transaction
9. wallet_after: The balance in the wallet after the payment transaction

4.1.36 PaymentMethod.php

This model have information about the payment methods

File Location: app/Models

This model have following properties:

1. payment_name: The name of payment method
2. deleted_at: The timestamp of when payment method

4.1.37 Property.php

This model contains the details of property

File Location: app/Models

This model have following properties:

1. property_name: The name of the property
2. property_status: The status of the property
3. deleted_at: The timestamp of when the property is deleted

4.1.38 PushNotification.php

This model contains information about the push notification sent to the mobile application users

File Location: app/Models

This model have following properties:

1. title: The title to sent in push notification
2. message: The message to sent in push notification
3. status: The status of push notification
4. image: The file location or file name to the image to sent in push notification
5. type: The type of message sent in push notification

4.1.39 RechargePaymentMethod.php

This model defines the payment method that was used for recharging the wallet

File Location: app/Models

This model have following properties:

1. payment_method: This method defines which payment method is used for recharging the wallet

4.1.40 ReportIssue.php

This model contains the information of issues that was reported by the user

File Location: app/Models

This model have following properties:

1. cs_id: The id of the charging station
2. user_id: The id of the user
3. name: The name of the user
4. email: The email of the user
5. description: The description of the issue
6. issue: The issue details in detail
7. image: The file name or location to the image with issue

4.1.41 Settings.php

This model is used to store the settings. The settings name and corresponding value

File Location: app/Models

This model have following properties:

1. settings_name: The name of the settings
2. settings_value: The value of the settings
3. deleted_at: The timestamp of when the settings is deleted

4.1.42 Sites.php

This model defines the information about the site

File Location: app/Models

This model have following properties:

1. site_name: The name of the site
2. site_status: The status of the site
3. deleted_at: The timestamp of when the site is deleted

4.1.43 SmsSend.php

This model contains the information for sms sending

File Location: app/Models

This model have following properties:

1. message: The message
2. status: The status of the message
3. type: The type of the message

4.1.44 State.php

This model contains information about countries and states

File Location: app/Models

This model have following properties:

1. country: The name of the country
2. state: The state of the country

4.1.45 StripeBankAccount

This model contains the account details in the stripe payment gateway

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. stripe_bank_id: The id of the stripe bank
3. stripe_account_holder_name: The name of the bank account holder
4. stripe_account_holder_type: The type of the bank account holder
5. stripe_fingerprint: The fingerprint key of the stripe gateway
6. stripe_last4: The last 4 digits of card
7. routing_number: The routing number of bank account
8. stripe_account: The stripe account

4.1.46 User.php

This model contains information about the user.

File Location: app/Models

This model have following properties:

1. name: The name of the user
2. email: The email of the user
3. password: The encrypted password of the user
4. type: The type of user. The types are admin, user and host
5. mobile: The mobile number of the user
6. otp: The otp used for SMS verification
7. image: The file name or link to the uploaded user image
8. status: The status of the user
9. last_name: The last name of the user
10. fcm_token: The firebase token used for push notification
11. eircode: The location eir code of the user
12. payment_method: The payment method used by the user
13. last_active: The timestamp of when user was last active

14. referral_code: The unique referral code to the user
15. county: The county of the user
16. state: The state of the user
17. country: The country of the user
18. deleted_at: The timestamp of when user is deleted
19. is_permission: Whether the user have permission
20. user_verification_status: The status of user verification
21. freshwork_id: The id for the freshworks CRM
22. freshwork_log: The log details for the freshworks CRM
23. stripe_account_id: The account id of the stripe payment gateway
24. stripe_account_error: The error occurred for particular user in stripe payment gateway
25. wallet_alert: The alerts for the user wallet
26. hostname: The name if the user type is host
27. chargeename: The name if the user type is chargee or user
28. promocode: The promocode used by user.

This model have relationship with other models. They are:

1. wallet() - This defines relationship with Wallet model. Defines the wallet associated to the user, The column used for reference is user_id in Wallet model and id in this model
2. vehicles() - This defines relationship with UserVehicle model. Defines the vehicles associated with the user, The column used for reference is user_id in UserVehicle model and id in this model
3. booking() - This defines relationship with Booking model. Defines the bookings associated with the user, The column used for reference is user_id in Booking model and id in this model
4. chargingStation() - This defines relationship with ChargingStation model. Defines the charging station associated with the user if the user type is a host, The column used for reference is user_id in ChargingStation model and id in this model
5. ids() - This defines relationship with UserVerificationDoc model. Defines the verification documents associated with the user, The column used for reference is user_id in UserVerificationDocs model and id in this model

4.1.47 UserCategory.php

This model is used for categories of users

File Location: app/Models

This model have following properties:

1. category_name: The name of user category
2. category_type: The type of user category
3. category_value: The user category
4. status: The status of user category

4.1.48 UserFavourites.php

This model is used for adding favourite charging stations of the users.

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. cs_id: The id of the favourite charging station
3. deleted_at: The timestamp of when the favourite charging station is deleted

4.1.49 UserTmp.php

This model is used to store user details temporarily

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. mobile: The mobile number of the user
3. deleted_at: The timestamp of when the temporary user is deleted

4.1.50 UserVehicle.php

This model is used to update the vehicle details of the user

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. vehicle_id: The id of the vehicle
3. vehicle_name: The name of the vehicle
4. deleted_at: The timestamp of when vehicle details is deleted

This model have relationship with other models. They are:

1. vehicle() - This defines relationship with VehicleVariants model. Defines the vehicle variant associated to the user, The column used for reference is id in VehicleVariants model and vehicle_id in this model

4.1.51 UserVerificationDoc.php

This model is used to store the user documents for verification.

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. documents: The filename or location to the uploaded documents of user

4.1.52 VehicleBattery.php

This model defines the battery power (In Watts) of a vehicle.

File Location: app/Models

This model have following properties:

1. battery_watts: This stores the battery power information

4.1.53 VehicleModel.php

This model is used to store information about different vehicle brands and their models.

1. brand_id: The id of the vehicle brand
2. vehicle_model_name: The name of the vehicle model
3. vehicle_model_image: The file name or location to the image of vehicle model
4. vehicle_model_status: The status of the vehicle model
5. vehicle_battery: The battery information of the vehicle model

This model have relationship with other models. They are:

1. vehicle() - This defines relationship with brands model. Defines the brand of the vehicle, The column used for reference is id in brands model and brand_id in this model
2. brand() - This defines relationship with brands model. Defines the brand of the vehicle, The column used for reference is id in brands model and brand_id in this model

4.1.54 VehicleVariants.php

This model is used to store the information of the vehicle variants of the models.

File Location: app/Models

This model have following properties:

1. model_id: The id of the vehicle model
2. variant_name: The name of the vehicle model variant
3. variant_range: The range of the vehicle model variant
4. battery_watt: The battery power of the vehicle model variant

This model have relationship with other models. They are:

1. model() - This defines relationship with VehicleModel model. Defines the model of the variant, The column used for reference is id in VehicleModel model and model_id in this model

4.1.55 Wallet.php

This model is used to store the wallet information of the user

File Location: app/Models

This model have following properties:

1. user_id: The id of the user who owns the wallet
2. wallet: The balance available in the wallet

4.1.56 WalletHistory.php

This model is used to store the history information of the user wallet

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. type: The type of history - Credit or Debit
3. booking_id: The id of booking made by user

4. wallet: The id of the wallet owned by user
5. wallet_before: The balance in wallet before transaction
6. wallet_after: The balance in wallet after transaction
7. wallet_for: The wallet usage person type
8. wallet_withdraw_id: The id of wallet withdrawal information
9. deleted_at: The timestamp of when wallet history is deleted
10. payment_id: The id of payment transaction
11. type_text: The text to display the user

4.1.57 WalletOffer.php

This model contains information about the wallet offers

File Location: app/Models

This model have following properties:

1. user_category: The category of the user
2. wallet: The id of the user's wallet
3. limit: The limit of the offer
4. offer_applied: The count of how many applied the offer

4.1.58 WalletWithdrawLog.php

This model is used to store information on withdrawals from wallet.

File Location: app/Models

This model have following properties:

1. user_id: The id of the user
2. transfer_response: The response received from api after payment transfer
3. transfer_id: The id of the payment transfer
4. payout_response: The response received from api after payout transfer
5. payout_id: The id of the payout transfer
6. transferred_amount: The amount of the transaction
7. transfer_status: The status of the transfer
8. payout_status: The status of the payout
9. withdrawal_status: The status of the withdrawal
10. withdrawal_id: The id of the withdrawal

This model have relationship with other models. They are:

1. user() - This defines relationship with User model. Defines the User who done the wallet transaction, The column used for reference is id in User model and user_id in this model

4.1.59 WalletWithdrawRequest.php

This model contains information about the requests to withdraw amount from wallet

File Location: app/Models

This model have following properties:

1. user_id: The id of the user.
2. wallet: The amount to withdraw.
3. withdraw_status: The status of the wallet withdrawal request.
4. withdraw_note: The additional note for wallet withdrawal request.
5. payment_method: The payment method used for withdrawal.
6. payment_id: The if of payment.

This model have relationship with other models. They are:

1. user() - This defines relationship with User model. Defines the User who done the wallet transaction, The column used for reference is id in User model and user_id in this model

4.1.60 WebPages.php

This model is used for storing webpage information

File Location: app/Models

This model have following properties:

1. page_name: The name of webpage
2. page_content: The content of webpage

4.1.61 brands.php

This model is used to store the vehicle brands inforamtion

File Location: app/Models

This model have following properties:

1. brand_name: The name of the vehicle brand
2. brand_logo: The file name or location to the image of the vehicle brand logo
3. brand_status: The current status of the vehicle brand
4. deleted_at: The timestamp of when the vehicle brand is deleted

This model have relationship with other models. They are:

1. vehiclemodels() - This defines relationship with VehicleModel model. Defines the brand for each vehicle model, The column used for reference is brand_id in VehicleModel model and id in this model

4.2 Controllers

Controllers are the main part of the application. The logic, working, storing, processing information for the application is written inside these controllers. The controllers define which data to take, process and store. The controllers used in our web application are as follows:

4.2.1 AnalyticsController.php

This controller is used to get data from the database and process information for the analytics dashboard. This gives analytics data for users, bookings, hosts, sales, locations etc.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.1.1 bookings()

- Line Numbers: 15 - 61
- This method takes 3 inputs - Start date, End date and Status (Optional)
- If there is no start or end date passed as input, The current date will be the start date and end date
- If there is no status as input, The data with all status will be considered
- The booking details is retrieved from the database using the start date, end date and status
- The retrieved data will be ordered by id in descending order
- The data will be ordered by id in descending order
- The booking data will be passed to the view admin.analytics.booking along with the start date and end date

4.2.1.2 users()

- Line Numbers: 63 - 94
- The input parameters would be start date, end date and status
- The list of users is retrieved from the database checking the start date, end date, status and type as normal user
- If the status parameters are not available, The list of whole users will be retrieved ordering latest as first
- After retrieving data, the user list will be paginated or divided down into groups of pages for easy listing and access
- This paginated users list will be passed to the view admin.analytics.users along with start date and end date

4.2.1.3 hosts()

- Line Numbers: 96 - 151
- The input parameters would be start date, end date and status
- The list of users is retrieved from the database checking the start date, end date, status and type as host user
- If the status parameters are not available, The list of whole hosts will be retrieved ordering latest as first
- After retrieving data, the host list will be paginated or divided down into groups of pages for easy listing and access
- This paginated hosts list will be passed to the view admin.analytics.hosts along with start date, end date and status

4.2.1.4 locations()

- Line Numbers: 153 - 188
- The input parameters would be start date, end date and status
- The list of charging stations is retrieved from the database checking the start date and end date of creation, status of the charging station
- If the parameters are not available, The list of whole charging stations will be retrieved ordering latest as first
- This list of locations will be then passed to the view admin.analytics.locations along with start date and end date

4.2.1.5 sales()

- Line Numbers: 190 - 217
- The input parameters would be start date and end date
- The list of bookings is retrieved from the database checking the start date and end date of the booking
- If the parameters are not available, The list of whole bookings will be retrieved ordering latest as first
- This list of bookings will be then passed to the view admin.analytics.sales along with start date and end date

4.2.2 BookingController.php

This controller is used to manage the user bookings.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.2.1 index()

- Line Numbers: 15 - 19
- The booking list is retrieved from the database as ordering latest as first
- The list is then paginated to divide into pages of 25
- The list is then passed to the view admin.booking.index

4.2.2.2 approve()

- Line Numbers: 24 - 30
- This takes id of the booking as the input
- The corresponding booking is retrieved from the database and update the status to 1. Which means approved

4.2.2.3 show()

- Line Numbers: 62 - 66
- This takes id of the booking as the input
- The corresponding booking is retrieved from the database
- The booking details is passed to the view admin.bookings.details

4.2.2.4 reject()

- Line Numbers: 35 - 41
- This takes id of the booking as the input
- The corresponding booking is retrieved from the database and update the status to 5. Which means rejected

4.2.3 BrandController.php

This controller is used to manage the vehicle brands.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.3.1 index()

- Line Numbers: 14 - 18
- The list of complete non deleted brands is retrieved from the database and ordered by latest
- This brands list is then passed to the view admin.brands.index

4.2.3.2 index()

- Line Numbers: 23 - 26
- This method redirects to the view admin.brands.create

4.2.3.3 store()

- Line Numbers: 35 - 53
- This method first validates the input parameters.
- It validates whether the `name` is available, and the `image` parameter should be an image with a maximum size of 2MB and types: jpg, png, jpeg, gif, or svg.
- It initializes the `image_path` variable as an empty string.
- If the `image` parameter is present in the request, it stores the uploaded image in the `public/image` directory and updates the `image_path` variable with the stored image's path.
- It creates a new `brands` record with the `brand_logo` set to `image_path` and `brand_name` set to the provided `name`.
- Finally, it redirects to the `admin/brands` route with a success message indicating that the brand was successfully created.

4.2.3.4 edit()

- Line Numbers: 66 - 70
- This method retrieves a brand with the specified `id` that has not been deleted.
- It assigns the retrieved brand to the `brand` variable.
- It then returns the `admin.brands.edit` view, passing the `brand` variable to the view using the `compact` function.

4.2.3.5 update()

- Line Numbers: 75 - 94
- This method first validates the input parameters.
- It ensures that the `name` parameter is present in the request.
- It updates the `brand_name` of the brand with the specified `id` using the value provided in the request.
- If the `image` parameter is present in the request, it stores the uploaded image in the `public/image` directory and updates the `brand_logo` of the brand with the new image path.
- Finally, it redirects to the `admin/brands` route with a success message indicating that the brand was successfully updated.

4.2.3.6 destroy()

- Line Numbers: 100 - 105
- This method soft deletes a brand with the specified `id`.
- It updates the `deleted_at` field of the brand to the current date and time, indicating the brand has been deleted.

4.2.4 ChargerBrandController.php

This controller is used to manage the charger brands.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.4.1 index()

- Line Numbers: 15 - 19
- This method retrieves all charger brands that have not been deleted.
- It fetches the charger brands from the `ChargerBrands` model, orders them by `id` in descending order, and assigns them to the `charger` variable.
- It then returns the `admin.settings.charger_brands.index` view, passing the `charger` variable to the view using the `compact` function.

4.2.4.2 create()

- Line Numbers: 24 - 28
- This method retrieves all battery watts that have not been deleted.
- It fetches the battery watts from the `BatteryWatts` model and assigns them to the `watts` variable.
- It then returns the `admin.settings.charger_brands.create` view, passing the `watts` variable to the view using the `compact` function.

4.2.4.3 store()

- Line Nuumbers: 33 - 46
- This method first validates the input parameters.
- It ensures that the `name` and `watts` parameters are present in the request.
- It creates a new `ChargerBrands` record with the `charger_brand_name` set to the lowercase value of the provided `name` and the `battery_watts` set to the provided `watts`.
- Finally, it redirects to the `admin/charger_brand` route with a success message indicating that the charger brand was successfully created.

4.2.4.4 edit()

- Line Numbers: 59 - 70
- This method attempts to retrieve data required for editing a charger brand.
- It fetches all battery watts that have not been deleted and assigns them to the `watts` variable.
- It fetches the charger brand with the specified `id` and assigns it to the `charger` variable.
- It then returns the `admin.settings.charger_brands.edit` view, passing the `charger` and `watts` variables to the view using the `compact` function.
- If an `ErrorException` occurs, it redirects to the `admin/charger_brand` route with an error message indicating invalid charger brand details.

4.2.4.5 update()

- Line Numbers: 75 - 98
- This method first validates the input parameters.
- It ensures that the `name` parameter is present, unique among `charger_brand_name`s excluding the current record, and that the `watts` parameter is present in the request.
- It then attempts to find the charger brand with the specified `id`. If found, it updates the `charger_brand_name` and `battery_watts` fields with the provided values.
- If the update is successful, it redirects to the `admin/charger_brand` route with a success message indicating that the charger brand was successfully updated.
- If an `ErrorException` occurs, it redirects to the `admin/charger_brand` route with an error message indicating invalid charger brand details.

4.2.4.6 destroy()

- Line Numbers: 103 - 117
- This method attempts to soft delete a charger brand with the specified `id`.
- It tries to find the charger brand using the `id`. If found, it updates the `deleted_at` field to mark it as deleted, using the current date and time.
- If the deletion is successful, it returns a JSON response with status `'success'` and a message indicating that the charger brand was successfully deleted.
- If an `ErrorException` occurs during the process, it returns a JSON response with status `'error'` and a generic error message indicating that something went wrong.

4.2.5 ChargingStationController.php

This controller is used to manage the charging stations.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.5.1 index()

- Line Numbers: 34 - 38
- This method retrieves all charging stations that have not been deleted.
- It fetches the charging stations from the `ChargingStation` model, orders them by `id` in descending order, and assigns them to the `stations` variable.
- It then returns the `admin.stations.index` view, passing the `stations` variable to the view using the `compact` function.

4.2.5.2 approve()

- Line Numbers: 43 - 48
- This method updates the status of a charging station with the specified `id` to `1`, indicating approval.
- It directly updates the `status` field of the charging station using the `ChargingStation` model.

4.2.5.3 reject()

- Line Numbers: 53 - 58
- This method updates the status of a charging station with the specified `id` to `0`, indicating rejection.
- It directly updates the `status` field of the charging station using the `ChargingStation` model.

4.2.5.4 unfeatured()

- Line Numbers: 60 - 65
- This method updates the `cs_featured` field of a charging station with the specified `id` to `0`, indicating that it is no longer featured.
- It directly updates the `cs_featured` field of the charging station using the `ChargingStation` model.

4.2.5.5 featured()

- Line Numbers: 67 - 72
- This method updates the `cs_featured` field of a charging station with the specified `id` to `1`, indicating that it is now featured.
- It directly updates the `cs_featured` field of the charging station using the `ChargingStation` model.

4.2.5.6 unverified()

- Line Numbers: 74 - 79
- This method updates the `cs_verified` field of a charging station with the specified `id` to `0`, indicating that it is no longer verified.
- It directly updates the `cs_verified` field of the charging station using the `ChargingStation` model.

4.2.5.7 verified()

- Line Numbers: 81 - 86
- This method updates the `cs_verified` field of a charging station with the specified `id` to `1`, indicating that it is now verified.
- It directly updates the `cs_verified` field of the charging station using the `ChargingStation` model.

4.2.5.8 create()

- Line Numbers: 91 - 100
- This method retrieves necessary data for creating a new charging station.
- It fetches location highlights where `lh_status` is `0` and assigns them to the `locationHighlights` variable.
- It fetches all battery watts and assigns them to the `batteryWatts` variable.
- It fetches charger brands where `charger_brand_status` is `0` and assigns them to the `chargerBrands` variable.
- It fetches all payment preferences and assigns them to the `paymentPreference` variable.
- It fetches users of type `'host'` and with `status 1` and assigns them to the `users` variable.
- Finally, it returns the `admin.stations.create` view, passing all fetched variables using the `compact` function.

4.2.5.9 store()

- Line Numbers: 105 - 444
- This method handles the creation of a new charging station based on the validated request parameters.
- It begins by validating various required fields and their formats, including user ID, station name, location details, descriptions, and multiple file uploads such as ID proofs, address proofs, photos, and charging point photos.
- Upon successful validation, it creates a new entry in the `ChargingStation` model with the provided data.
- It iterates over the uploaded photos and creates entries in the `CsPhoto` model for both location photos and charging point photos associated with the charging station.
- It iterates over selected location highlights and creates entries in the `CsLocationHighlight` model linking them to the charging station.
- Parking instructions are split into individual entries and stored in the `CsParkingInstruction` model.
- Charging details, including battery watts and charger brands, are stored in the `CsCharger` model.
- Charging points data, including their names, charges per kW, minimum charges, and numbers, are stored in the `CsChargingPoint` model.
- Settings related to the charging station, such as approval requirements, parking charges, and operational modes, are stored in the `CsSetting` model.
- Availability timings and parking charges for different days and day types are stored in the `CsParkingTimings` and `CsParkingTimingDay` models, respectively.
- Payment details such as account holder name, payment preferences, IBAN or PayPal details, and additional personal details are stored in the `CsPaymentDetails` model, and if required, a Stripe account is created for the user.
- Proof documents (ID proofs and address proofs) uploaded during station creation are stored in the `CsProofs` model.
- Finally, upon successful creation of the charging station and associated records, it redirects to the `admin/stations` route with a success message.

4.2.5.10 show()

- Line Numbers: 449 - 464
- This method retrieves details of a charging station with the specified `id`.
- It defines an array `$days` mapping numeric day indices to their respective day names.
- Using the `ChargingStation` model, it fetches the station details from the database based on the provided `id`.
- Finally, it renders the `admin.stations.details` view, passing the fetched `station` data and the `days` array for display.

4.2.5.11 edit()

- Line Numbers: 469 - 495
- This method prepares data for editing a charging station with the specified `id`.
- It fetches the charging station details using the `ChargingStation` model based on the provided `id`.
- Additionally, it retrieves related data such as charger details (`CsCharger`), parking instructions, settings (`CsSetting`), parking timings (`CsParkingTimings`), and payment details (`CsPaymentDetails`) associated with the charging station.
- It retrieves and formats parking instructions into a comma-separated string for easier display.
- User-related data such as hosts (`User` models) and payment preferences are also fetched to populate the edit form.
- Finally, it renders the `admin.stations.edit` view, passing all fetched data using the `compact` function.

4.2.5.12 update()

- Line Numbers: 500 - 957
- This method handles the update operation for a charging station based on the provided `id` and `Request` data.
- It begins by validating the incoming `Request` data using Laravel's validation rules, ensuring required fields are present and file uploads meet specified criteria.
- The charging station details are updated using the `ChargingStation` model based on the provided `id`, including core information such as name, location, about us, etc.
- It processes and stores new photos related to the charging station using the `CsPhoto` model for both general location photos and charging point photos.
- Updates or inserts parking instructions (`CsParkingInstruction`) based on the provided input.
- Updates charger details (`CsCharger`) such as wattage and brand.
- Manages charging points (`CsChargingPoint`) by either updating existing points or creating new ones based on the form input.
- Updates or creates settings (`CsSetting`) related to the charging station's operational preferences like approval requirements, parking charges, and service modes.
- Manages parking timings (`CsParkingTimings`) for both regular and night hours.
- Manages availability days (`CsAvailabilityDays`) and parking timings per day (`CsParkingTimingDay`) based on the form input.
- Updates or creates payment details (`CsPaymentDetails`) associated with the charging station host, handling bank account information and uploading proofs if provided.
- Finally, redirects to the admin stations page with a success message upon successful update.

4.2.5.13 destroy()

- Line Numbers: 962 - 968
- This method handles the logical deletion of a charging station identified by the provided `id`.
- It updates the `ChargingStation` model where the `id` matches the provided parameter.
- Sets the `status` field to `0`, indicating that the charging station is no longer active.
- Sets the `deleted_at` timestamp to the current date and time using PHP's `date` function to mark it as deleted.

4.2.5.14 deletePhoto()

- Line Numbers: 970 - 974
- This method handles the deletion of a photo associated with a charging station.
- It deletes the photo record from the `CsPhoto` model where the photo's `id` matches the provided parameter.
- Responds with a JSON object indicating success upon successful deletion, including a status and message.

4.2.6 DiscountController.php

This controller is used to manage the discounts.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.6.1 index()

- Line Numbers: 16 - 27
- This method retrieves discounts and user categories from their respective database tables for display in the admin panel.
- Discounts are fetched from the `Discount` model, ordered by descending `id`, and paginated to display 25 entries per page.
- User categories are fetched from the `UserCategory` model where the status is 0 (assuming active categories).
- It prepares the fetched user categories into an associative array `$category`, where keys are category IDs and values are category names.
- Passes the retrieved discounts (`$discounts`), user categories (`$userCategory`), and the prepared category associative array (`$category`) to the view `admin.discount.index`.

4.2.6.2 create()

- Line Numbers: 32 - 37
- This method prepares data required for creating a new discount entry.
- Fetches users (`$users`) from the `User` model ordered alphabetically by name.
- Retrieves active user categories (`$userCategory`) from the `UserCategory` model where `status` equals 0.
- Passes the fetched users and user categories to the view `admin.discount.create` using the `compact` function.

4.2.6.3 store()

- Line Numbers: 42 - 83
- This method validates and stores a new discount entry based on the incoming `Request` data.
- Validates the `Request` data using Laravel's validation rules, ensuring required fields are present and discount values are within appropriate limits.
- Additional validation is applied if `discount_validity` is set to 0, ensuring the `discount_to` date follows `discount_from` using the `date_format` and `after` rules.
- Creates a new discount entry (`Discount`) in the database with fields including user ID, discount type, discount amount, coupon code, validity period, usage limit, and start/end dates.
- Redirects to the admin discounts page with a success message upon successful creation.

4.2.6.4 edit()

- Line Numbers: 96 - 103
- This method retrieves a specific discount entry (`Discount`) identified by the provided `id` for editing purposes.
- Fetches a collection of users ordered by name.
- Retrieves the discount entry corresponding to the given `id`.
- Fetches user categories (`UserCategory::where('status', 0)->get()`) for potential category-based filtering or display.
- Passes the fetched users, discount entry, and user categories to the view `admin.discount.edit` for rendering.

4.2.6.5 update()

- Line Numbers: 108 - 149
- This method handles the update operation for a discount based on the provided `id` and `Request` data.
- It begins by validating the incoming `Request` data using Laravel's validation rules, ensuring required fields such as `user_id`, `discount_type`, `discount`, `discount_coupon`, `discount_validity`, `discount_number_usage`, and `limit` are present and valid.
- If `discount_validity` is zero, additional validation ensures that `discount_to` follows the correct date format and comes after `discount_from`.
- The discount details are updated using the `Discount` model based on the provided `id`, updating fields such as `user_id`, `discount_type`, `discount`, `discount_coupon`, `discount_validity`, `discount_start`, `discount_end`, `discount_number_usage`, and `limit`.
- Finally, it redirects to the admin discounts page with a success message upon successful update.

4.2.7 EmailSettingsController.php

This controller is used to manage the email settings.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.7.1 create()

- Line Numbers: 22 - 26
- This method handles the creation process for email settings.
- It retrieves the latest email settings entry from the `EmailSettings` model.
- Passes the retrieved settings data to the `admin.email_settings.create` view using compact().
- This view is responsible for displaying a form to create or update email settings.

4.2.7.2 store()

- Line Numbers: 31 - 64
- This method handles the storage of email settings based on the provided `Request` data.
- It begins by validating the incoming `Request` data using Laravel's validation rules, ensuring required fields such as `smtp_host`, `smtp_port`, `smtp_username`, `smtp_password`, `smtp_encryption`, `from_address`, and `from_name` are present and valid.
- All request data is captured into the `$data` variable.
- It retrieves the latest email settings entry from the `EmailSettings` model.
- If settings exist (`$settings`), it updates them with `$data` and stores the update status in `$status`.
- If no settings exist (`$settings` is null), it creates new settings using `EmailSettings::create($data)` and stores the creation status in `$status`.
- Upon successful storage (`$status`), environment variables related to email configuration (e.g., `MAIL_HOST`, `MAIL_PORT`) are updated using `$this->updateEnvVariable()`.
- Finally, it redirects to the email settings creation page (`admin/email_settings/create`) with a success message upon successful creation or update of email settings.

4.2.7.3 updateEnvVariable

- Line Numbers: 98 - 116
- This method updates or adds a key-value pair in the '.env' file based on the provided `$key` and `$value`.
- It first retrieves the path to the '.env' file using `base_path()`.
- Reads the current content of the '.env' file into `$currentEnvContent` using `file_get_contents()`.
- Checks if the `$key` already exists in the '.env' file:
  - If the `$key` exists, it updates the corresponding value by replacing the existing line with the new `$value`.
  - If the `$key` does not exist, it appends a new key-value pair to the end of `$currentEnvContent`.
- Updates the '.env' file with the modified content using `file_put_contents()`.
- Returns `true` upon successful update of the '.env' file.

4.2.8 EmailTemplateController.php

This controller is used to manage the email templates.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.8.1 index()

- Line Numbers: 15 - 19
- This method retrieves a paginated list of email templates for display in the admin panel.
- It fetches email templates from the `EmailTemplate` model, ordering them by `id` in descending order.
- Paginates the retrieved templates for easier navigation and viewing.
- Passes the paginated templates data to the `admin.email_templates.index` view using `compact()`.
- The view is responsible for displaying the list of email templates in the admin interface.

4.2.8.2 create()

- Line Numbers: 24 - 27
- This method returns the view responsible for creating new email templates in the admin panel.
- It directs to the `admin.email_templates.create` view, which contains the form for creating new email templates.
- Users can input details such as template name, subject, body content, and other relevant information through this view.

4.2.8.3 store()

- Line Numbers: 32 - 43
- This method handles the storage of a new email template based on the provided `Request` data.
- It begins by validating the incoming `Request` data using Laravel's validation rules, ensuring required fields such as `template_name`, `subject`, and `body` are present and valid.
- All request data is captured into the `$data` variable.
- It creates a new email template using the `EmailTemplate::create($data)` method and stores the creation status in `$status`.
- Upon successful storage (`$status`), it redirects to the admin email templates page (`admin/email_templates`) with a success message indicating the template was successfully added.

4.2.8.4 edit()

- Line Numbers: 56 - 60
- This method retrieves an email template for editing based on the provided `$id`.
- It uses `EmailTemplate::findOrFail($id)` to find the email template by its unique identifier (`$id`).
- Passes the retrieved template data to the `admin.email_templates.edit` view using `compact()`, allowing the view to display the template details for editing.
- Users can modify fields such as template name, subject, body content, etc., through this view.

4.2.8.5 update()

- Line Numbers: 65 - 71
- This method handles the update operation for an email template based on the provided `$id` and `Request` data.
- It retrieves the email template using `EmailTemplate::findOrFail($id)` to ensure the template exists.
- Captures all incoming `Request` data into the `$data` variable.
- Updates the retrieved template with the new data using `$template->update($data)` and stores the update status in `$status`.
- Upon successful update (`$status`), it redirects to the admin email templates page (`admin/email_templates`) with a success message indicating the template was successfully updated.

4.2.9 HomeController.php

This controller is used to manage the dashboard.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.9.1 index()

- Line Numbers: 22 - 25
- This method directs to the admin dashboard view.
- It returns the `admin.dashboard` view, which typically serves as the main interface for administrative tasks and displays an overview of the system's current status.
- This view may include various widgets, charts, or summaries relevant to the admin user's role and responsibilities.

4.2.9.2 home()

- Line Numbers: 33 - 49
- This method determines the appropriate dashboard to redirect a logged-in user based on their user type.
- It first retrieves the authenticated user using `Auth::user()`.
- Checks the user's type (`$user->type`) to determine if they are a 'host' or 'admin'.
- If the user is a 'host', it redirects them to the host dashboard using `redirect()->intended('host/dashboard')`.
- For an 'admin' user or any other type not explicitly handled, it redirects to the admin dashboard using `redirect()->intended('admin/dashboard')`.
- The `intended()` method ensures the user is redirected to their originally intended destination if available, or defaults to the specified dashboard path.

4.2.10 HostController.php

This controller is used to manage the hosts.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.10.1 index()

- Line Numbers: 16 - 65
- This method handles the index operation for displaying a list of hosts based on the provided `Request` data.
- It begins by initializing the `dateFrom` and `dateTo` variables with the current date in `m/d/Y` format.
- It retrieves the `status` parameter from the request.
- A new query builder instance for the `User` model is created.
- The method checks if the request contains a `date_from` parameter.
- If present, it retrieves the `date_from` and `date_to` parameters from the request.
- Converts the `date_from` string to a Unix timestamp.
- Formats the Unix timestamp to `m/d/Y` format and assigns it to `dateFrom`.
- Converts the Unix timestamp to `Y-m-d 00:00:00` format for the start of the day.
- Converts the `date_to` string to a Unix timestamp.
- Formats the Unix timestamp to `m/d/Y` format and assigns it to `dateTo`.
- Converts the Unix timestamp to `Y-m-d 00:00:00` format for the start of the day.
- Parses the `date_from` to a Carbon instance and gets the start of the day as a date string.
- Parses the `date_to` to a Carbon instance and gets the end of the day as a date string.
- Adds a condition to the query to filter results where the `created_at` column is between `from` and `to`.
- Checks if the request has a `status` parameter, and if it is not equal to -1 and less than or equal to 1.
- Adds a condition to the query to filter results by the `status` parameter.
- Checks if the request has a `status` parameter and if it is equal to 2.
- Adds a condition to the query to filter results where the `deleted_at` column is not null.
- Checks if the request has a `status` parameter and if it is equal to 3.
- Adds a condition to the query to filter results where the `deleted_at` column is null and does not have a related `chargingStation`.
- Executes the query with additional conditions to filter results by `type` as 'host', order by `id` in descending order, and where `deleted_at` is null. The results are stored in the `users` variable.
- Returns the `admin.hosts.index` view, passing the `users`, `dateFrom`, `dateTo`, and `status` variables to the view.

4.2.10.2 create()

- Line Numbers: 70 - 73
- This method handles the creation operation for a new host.
- It simply returns the `admin.hosts.create` view.
- The `admin.hosts.create` view typically contains a form for creating a new host, including fields for necessary host details.

4.2.10.3 store()

- Line Numbers: 78 - 98
- This method handles the store operation for saving a new host based on the provided `Request` data.
- It begins by validating the incoming `Request` data using Laravel's validation rules.
- Ensures required fields such as `name`, `last_name`, `email`, and `mobile` are present and valid.
- The `email` field must be unique in the `users` table.
- The `mobile` field must be unique in the `users` table and must be between 10 and 14 characters.
- Creates a new user with the type `host` using the validated data.
- Fields `name`, `last_name`, `email`, `mobile`, and `type` are populated from the request.
- The `password` field is set to an empty string.
- Saves the new user to the `users` table.
- Finally, it redirects to the admin hosts page with a success message upon successful creation.

4.2.10.4 paymentRequests()

- Line Numbers: 111 - 115
- This method handles the retrieval and display of payment requests.
- It fetches all `WalletWithdrawRequest` records from the database, ordered by `id` in descending order.
- The fetched payment requests are stored in the `requests` variable.
- Returns the `admin.hosts.requests` view, passing the `requests` variable to the view.
- The `admin.hosts.requests` view typically displays a list of payment requests, including details such as request amounts, dates, and statuses.

4.2.10.5 edit()

- Line Numbers: 120 - 124
- This method handles the edit operation for a host based on the provided `id`.
- It retrieves the `User` record where the `id` matches the provided `id`.
- The retrieved user record is stored in the `user` variable.
- Returns the `admin.hosts.edit` view, passing the `user` variable to the view.
- The `admin.hosts.edit` view typically contains a form pre-filled with the user's current details, allowing for editing and updating the user's information.

4.2.10.6 update()

- Line Numbers: 129 - 147
- This method handles the update operation for a host based on the provided `id` and `Request` data.
- It retrieves the `User` record where the `id` matches the provided `id`.
- The retrieved user record is stored in the `user` variable.
- If the user exists, it updates the `User` record with the new data from the `Request`.
- The fields `name`, `email`, `mobile`, and `last_name` are updated with the values from the `Request`.
- After updating, it redirects to the admin hosts page with a success message indicating the host details were successfully updated.
- If the user does not exist, it redirects to the admin hosts page with an error message indicating that the host details were not found.

4.2.11 LocationHighlightsController.php

This controller is used to manage the highlights of charging station location.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.11.1 index()

- Line Numbers: 14 - 18
- This method handles the index operation for displaying a list of location highlights.
- It retrieves all `LocationHighlights` records from the database where the `deleted_at` column is null.
- The retrieved location highlights are ordered by `id` in descending order and stored in the `location_highlights` variable.
- Returns the `admin.settings.location_highlights.index` view, passing the `location_highlights` variable to the view.
- The `admin.settings.location_highlights.index` view typically displays a list of location highlights, including details such as names, descriptions, and other relevant information.

4.2.11.2 create()

- Line Numbers: 23 - 26
- This method handles the creation operation for a new location highlight.
- It returns the `admin.settings.location_highlights.create` view.
- The `admin.settings.location_highlights.create` view typically contains a form for creating a new location highlight, including fields for details such as name, description, and any other relevant information.

4.2.11.3 store()

- Line Numbers: 31 - 49
- This method handles the store operation for saving a new location highlight based on the provided `Request` data.
- It begins by validating the incoming `Request` data using Laravel's validation rules.
- Ensures the `name` field is required and unique in the `location_highlights` table under the `lh_name` column.
- Validates the `image` field, ensuring it is an image file with allowed extensions (`jpg`, `png`, `jpeg`, `gif`, `svg`) and a maximum size of 2048 KB.
- Initializes an empty string `$image_path` to store the path of the uploaded image.
- Checks if the request has an `image` field and stores the uploaded image in the `public` disk under the `image` directory.
- Creates a new `LocationHighlights` record using the validated data.
- Converts the `name` to lowercase and assigns it to the `lh_name` field.
- Stores the path of the uploaded image in the `lh_image` field.
- Redirects to the `admin/location_highlights` page with a success message upon successful creation of the location highlight.

4.2.11.4 edit()

- Line Numbers: 62 - 66
- This method handles the edit operation for a location highlight based on the provided `id`.
- It retrieves the `LocationHighlights` record where the `id` matches the provided `id`.
- The retrieved location highlight record is stored in the `lc` variable.
- Returns the `admin.settings.location_highlights.edit` view, passing the `lc` variable to the view.
- The `admin.settings.location_highlights.edit` view typically contains a form pre-filled with the location highlight's current details, allowing for editing and updating the location highlight's information.

4.2.11.5 update()

- Line Numbers: 71 - 101
- This method handles the update operation for a location highlight based on the provided `id` and `Request` data.
- Validates the incoming `Request` data using Laravel's validation rules.
- Ensures the `name` field is required and must be unique in the `location_highlights` table under the `lh_name` column, excluding the current record with `id` equal to the provided `$id`.
- Attempts to retrieve the `LocationHighlights` record where the `id` matches the provided `$id`. If not found, throws an `ErrorException`.
- Updates the `lh_name` field of the retrieved location highlight record with the new `name` from the `Request`.
- Initializes an empty string `$image_path` to store the path of the uploaded image.
- Checks if the request has an `image` field.
- If an image is uploaded, stores the uploaded image in the `public` disk under the `image` directory and updates the `lh_image` field of the location highlight record with the new image path.
- Redirects to the `admin/location_highlights` page with a success message upon successful update.
- If an `ErrorException` occurs (location highlight not found), redirects to the `admin/location_highlights` page with an error message indicating invalid location highlight details.

4.2.11.6 destroy()

- Line Numbers: 106 - 111
- This method handles the soft delete operation for a location highlight based on the provided `id`.
- It updates the `LocationHighlights` record where the `id` matches the provided `id`.
- Sets the `deleted_at` column to the current date and time in `Y-m-d H:i:s` format using PHP's `date()` function.
- Soft deletion is typically used in Laravel to mark a record as deleted by setting a value in the `deleted_at` column instead of physically removing the record from the database.
- This approach allows for easy recovery of deleted records and maintains data integrity.

4.2.12 MailController.php

This controller is used to manage the emails.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.12.1 index()

- Line Numbers: 16 - 28
- This method handles the index operation for displaying a view related to email templates in the admin panel.
- It retrieves all `UserCategory` records where the `status` is 0 (assuming `UserCategory` represents categories of users).
- The retrieved user categories are stored in the `$userCategory` variable.
- Initializes an empty array `$category` to store category names keyed by their IDs.
- Iterates over each `$userCategory` record and populates the `$category` array where the key is the category's `id` and the value is the category's `category_name`.
- Fetches the `EmailTemplate` record with an `id` of 13 using `findOrFail()`, assuming it retrieves an existing template record or throws an exception if not found.
- Stores the fetched `$template` and `$category` arrays in the `admin.mail.index` view using `compact()`.
- The `admin.mail.index` view typically displays information related to email templates, possibly allowing administrators to edit or manage email templates and select user categories for targeted emails.

4.2.12.2 store()

- Line Numbers: 41 - 63
- This method handles the store operation for saving an email message to be sent later via a job queue.
- Validates the incoming `Request` data using Laravel's validation rules.
- Ensures the `user_id`, `subject`, and `message` fields are required.
- Constructs an array `$mail_data` containing the subject, message, recipient email (`to`), and type of user (`type`) from the `Request`.
- Initiates a new job instance of `App/Jobs/SendEmail` and passes `$mail_data` as a parameter.
- Sets a delay for the job execution using `delay()`, scheduling it to run 2 seconds from the current time.
- Dispatches (`dispatch()`) the job to the queue system for asynchronous processing.
- Redirects to the `admin/mail` route with a success message indicating that the email message was successfully saved to the mail queue.
- This approach ensures that the email sending process is handled asynchronously via job queues, improving application responsiveness and scalability.

4.2.13 PaymentPreferenceController.php

This controller is used to manage the payment preferences.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.13.1 index()

- Line Numbers: 14 - 18
- This method handles the index operation for displaying a list of payment preferences in the admin settings.
- It retrieves all `PaymentPreference` records from the database where the `deleted_at` column is null, indicating active preferences.
- The retrieved payment preferences are stored in the `$location_highlights` variable.

- Returns the `admin.settings.payment_preference.index` view, passing the `$location_highlights` variable to the view.
- The `admin.settings.payment_preference.index` view typically displays a list of payment preferences, including details such as payment methods, account information, and other relevant settings.

4.2.13.2 create()

- Line Numbers: 23 - 26
- This method handles the creation operation for a new payment preference.
- It returns the `admin.settings.payment_preference.create` view.
- The `admin.settings.payment_preference.create` view typically contains a form for creating a new payment preference, including fields for details such as payment method, account information, and any other relevant settings.

4.2.13.3 store()

- Line Numbers: 31 - 42
- This method handles the store operation for saving a new payment preference based on the provided `Request` data.
- Validates the incoming `Request` data using Laravel's validation rules.
- Ensures the `name` field is required and must be unique in the `payment_preferences` table under the `preference_name` column.
- Creates a new `PaymentPreference` record using the validated data.
- Converts the `name` to lowercase and assigns it to the `preference_name` field.
- Stores the newly created `PaymentPreference` record in the database.
- Redirects to the `admin/payment_preference` route with a success message upon successful creation of the payment preference.
- This approach ensures that payment preferences are stored with unique names and are easily retrievable for administrative purposes.

4.2.13.4 edit()

- Line Numbers: 55 - 59
- This method handles the edit operation for a payment preference based on the provided `id`.
- It retrieves the `PaymentPreference` record where the `id` matches the provided `id`.
- The retrieved payment preference record is stored in the `$lc` variable.
- Returns the `admin.settings.payment_preference.edit` view, passing the `$lc` variable to the view.
- The `admin.settings.payment_preference.edit` view typically contains a form pre-filled with the payment preference's current details, allowing for editing and updating the payment preference's information.

4.2.13.5 update()

- Line Numbers: 64 - 85
- This method handles the update operation for a payment preference based on the provided `id` and `Request` data.
- Validates the incoming `Request` data using Laravel's validation rules.
- Ensures the `name` field is required and must be unique in the `payment_preferences` table under the `preference_name` column, excluding the current record with `id` equal to the provided `$id`.

- Attempts to retrieve the `PaymentPreference` record where the `id` matches the provided `$id`. If not found, throws an `ErrorException`.
- Updates the `preference_name` field of the retrieved payment preference record with the new `name` from the `Request`.
- Redirects to the `admin/payment_preference` route with a success message upon successful update.
- If an `ErrorException` occurs (payment preference not found), redirects to the `admin/payment_preference` route with an error message indicating invalid payment preference details.

4.2.14 PromotionController.php

This controller is used to manage the promotions.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

4.2.14.1 sms()

- Line Numbers: 16 - 27
- This method handles the index operation for displaying a view related to SMS promotions in the admin panel.
- Retrieves all `User` records from the database, ordered by `name` in ascending order.
- Retrieves all `UserCategory` records where the `status` is 0 (assuming `UserCategory` represents categories of users).
- Initializes an empty array `$category` to store category names keyed by their IDs.
- Iterates over each `$userCategory` record and populates the `$category` array where the key is the category's `id` and the value is the category's `category_name`.
- Returns the `admin.promotions.sms.index` view, passing the `$users`, `$category`, and `$userCategory` variables to the view.
- The `admin.promotions.sms.index` view typically displays a form or interface for sending SMS promotions to users based on selected categories or individual users.

4.2.14.2 notifications()

- Line Numbers: 29 - 36
- This method handles the index operation for displaying a view related to push notifications in the admin panel.
- Retrieves all `UserCategory` records from the database where the `status` is 0 (assuming `UserCategory` represents categories of users).
- Initializes an empty array `$category` to store category names. However, in the provided code, the `$category` array is empty and not populated with any values.
- Returns the `admin.promotions.push_notification.index` view, passing the `$userCategory` and `$category` variables to the view.
- The `admin.promotions.push_notification.index` view typically displays a form or interface for sending push notifications to users based on selected categories or other criteria.

4.2.14.3 sendSMS()

- Line Numbers: 38 - 52
- This method handles the creation and queueing of SMS messages for promotions based on the provided `Request` data.
- Validates the incoming `Request` data using Laravel's validation rules.
- Ensures the `user_id` and `message` fields are required for sending the SMS.
- Creates a new `SmsSend` record in the database using the validated data.
- Sets the `type` field to the `user_id`, assuming it identifies the recipient or type of SMS.
- Sets the `message` field to the SMS message content provided in the `Request`.
- Sets the `status` field to 0, indicating that the SMS is queued and pending.
- Redirects to the `admin/promotions/sms` route with a success message indicating that the SMS has been successfully queued for sending soon.
- This approach ensures that SMS messages for promotions are queued and sent asynchronously, improving application responsiveness and scalability.

4.2.14.4 sendNotification()

- Line Numbers: 54 - 123
- This method handles the creation and queueing of push notifications based on the provided `Request` data.
- Validates the incoming `Request` data using Laravel's validation rules. It ensures that `user_id`, `message`, and `subject` fields are required.
- Creates a new `PushNotification` record in the database using the validated data. It sets the `type` field to the `user_id`, `title` to the `subject`, `message` to the `message`, and `status` to 1, assuming 1 represents a queued status.
- Constructs an SQL query to select `user_categories` where `id` matches `user_id` and `status` is 0.
- Executes the SQL query to retrieve `user_categories` based on the `user_id`.
- Checks if `$userCategories` is not empty and extracts `category_type` and `category_value` from the first result.
- Initializes `$where` as an empty string for further conditional checks.
- Depending on the `category_type`, retrieves `fcm_token` for users matching the `category_value` in `country`, `state`, or `county` fields. Also fetches user IDs matching these criteria into `$users`.
- Iterates through `$users` and creates a new `Notification` record for each user, using `subject` and `message` from the request.
- If `$fcmTokenUser` (array of FCM tokens) is not empty, sends a push notification using the `SendPushNotification` class with `subject`, `message`, and `$fcmTokenUser`.
- Redirects to the `admin/promotions/push_notifications` route with a success message indicating that the notification has been successfully queued for sending soon.

4.2.15 PropertyController.php

This controller is used to manage the properties.

File Location: app/Http/Controllers/admin/

Here are the methods used in this controller

5 Index