**OOP Project Report**

Car Parking Slot Booking Web Application

Semester 1 2020-2021

Icon

Description automatically generated

Members:

|  |  |
| --- | --- |
| **Name** | **ID** |
| Anshul Kanodia | 2020A7PS0174H |
| Chirag Gadia | 2020A7PS2050H |
| Kotha Rohit Reddy | 2020A7PS1890H |
| Uday Singh Thakur | 2020A7PS1721H |

Note: The project can also be viewed in github. The link for the repo is https://github.com/rohitkotha10/CarParkingApp.git

**Technologies used:**

Backend:

Spring Boot 2.6.1 was used for the implementing the backend server. It is a java framework. Spring uses Maven for project build tasks. The Backend server runs on Tomcat server in the port 8080. MySQL is used for database implementation. It runs of port 3036.

Frontend:

ReactJS was used for designing the frontend pages. Material UI Framework was widely used for the UI components.

Detailed build instructions and additional technical information is provided in the README.md file in the project folder.

**Class Diagram and Backend Program Structure:**

The Classes can be mainly divided into four categories.

1. Controller
2. Model
3. Service
4. Repository

Classes in controller kind are the ones which take care of sending and receiving data from the frontend.

Classes in model take care of representation of the entity and the tables are created for each class in model in database

Classes in Service are used to implement the functions related to each entity/model and uses the database.

Classes in Repository give available commands to remove, search, or create entities. These are all JPA Repositories.

The class diagrams can be found in the source folder.

**Routing and Page Structure**

Each page is associated with a separate link. These links are created with the help of react-router-dom v6 and the routes and the components in each of the routes are mentioned in routes.js. If a user enters an unknown route, he is redirected to a 404 not found page. If he tries to access a page without login, then an error is displayed on screen.

**Authorization Pages**

Three pages were created in Login, Registration and Verification Page.

Material UI’s components for text fields, Boxes, Cards etc. were used in ReactJS.

Login:

For Login Page, google Login was implemented in the frontend itself using react-google-login. When user clicks on Submit login button, the email and password details are sent to backend for verification and a response code is sent back. If success code is sent, the app navigates to the appropriate dashboard using the useNavigate React Hook. For Google Login, Only the email is sent to database, and if email exists in database, then he is validated, note that the user needs to be registered first in order to use Google login. The sending of data Is a POST request in the appropriate request mapping. This is mentioned in the corresponding controller classes.

***Insert login page***

Registration:

Registration page takes all the details and sends a POST request, this request triggers the Java Mail Sender and sends an email to the user for verification, after the email is sent, a response code is received from backend after which user is automatically redirected to verification page.

***Insert Registration page***

Verification:

In the verification page, the user needs to enter the email and verification code to be validated, a POST request is sent to validate the user and if a correct response is received, then user is automatically navigated to Login Page.

***Insert verification page***

In all these pages, if user enters an invalid email or if there is any problem, then an error message is displayed.

**Dashboards**

After login, the client is redirected to a different webpage, each one is having their own customized dashboard. In our website the dashboard is simply the drawer located in the left side of the page. By default all types of users user/admin/worker are redirected to the orders page by default upon login.

Admin Dashboard:

There are four pages Here:

1. Orders: All the orders can be viewed here in a Table. These are not arranged in any specific order.
2. Workers: All the workers can be viewed here in a Table. There is an option to add a worker which opens a dialog on clicking. After opening, you can enter the worker details and then save worker, the worker data is fetched using the react hook useEffect. It is a get request which return the list of all workers.
3. Users: Table of Users
4. Parking: Table of parking spots. There is also an option to add parking spots.

***Insert admin dashboard here***

User Dashboard:

There are three pages Here:

1. Orders: All the orders related to the user can be viewed here in a Table. These are not arranged in any specific order. There is an option to add comments here. The user may select the order ID and post his rating and comment for the order. The changes will be reflected immediately in the order page.
2. Booking: User has an option to book a slot for parking here.
3. Profile Info: Displays the users personal info

***Insert user dashboard here***

Worker Dashboard:

There two pages Here:

1. Orders: All the orders linked to the worker can be viewed here in a Table. These are not arranged in any specific order. Upon completion of an order the worker clicks on complete order and then enters the order id. Upon clicking submit, an email is sent to the user showing the payment.
2. Profile Info: Displays workers personal info

***Insert worker Dashboard here***

**Booking**

In the booking page of user, he is first asked to enter the date, check in time, duration, car wash option, and air fill option for his car parking. The user can book their parking slot for a minimum of 1 hour and a maximum of 5 hours. After that upon clicking next, a dialog is opened to select the parking location and worker. Upon clicking next, POST requests are sent with date, check in time and duration as parameters upon which the response is the list of available parking slot and available workers. Accordingly, the options for selecting worker and parking slot are displayed in the dialog. After selecting this, upon clicking submit, the order is added to the database and the orders table is updated immediately.

***Insert Booking pages here, date, time picker and slot picker***

**Payment**

The payment is calculated when the worker completes the order in his dashboard. Upon clicking the complete order button, a post request is sent to the database along with the order ID to calculate the payment. The payment Is calculated in this way:

25Rs. Per every hour

10 Rs. For filling air in the tyres

20 Rs. For Car wash.

Let us take a scenario,

The user takes a slot for 3 hours and opts for car wash but not filling air.

So, the amount to be paid is (3 \* 25) + 20 = 95Rs.

An email is sent to the user to pay the amount.

Upon completion of every order, the worker experience is updated. The experience includes the number of hours he worked across all orders during his work.

For Example, if a user opts for a slot with 4 hrs duration, then the experience of worker is incremented by 4 hrs. this is implemented whenever the order completion takes place.

***Insert Worker Order Confirmation pic and email here***

**Comments and Rating**

The user can add reviews to his orders in the orders page of his dashboard. Upon clicking add comment, a dialog is opened to enter the order ID, comment, and rating for his order. A minimum of 1 and a maximum of 5 stars can be given in a review. On submit, a post request is sent to backend and the changes are reflected immediately. We consider that the rating for the order also includes the rating of parking slot and worker.

For calculating the worker rating, we take the averages of all the ratings in the orders linked to him. This is calculated dynamically whenever the user posts a rating. Upon updating the order rating, while posting the review in the database, the workers rating is also updated, the implementation can be found in the **commentAdd** method of the **MyOrdersController** class.

***Insert add comment picture here***