**OOPS, Project -**

**Car Parking Slot Booking Web Application Project**

**Group 20 –**

Team Members –

1. Yash Ratnani – 2020A7PS0039H
2. Shubh Badjate – 2020A7PS0028H
3. Ashwin Naveen Pugalia – 2020A7PS1080H
4. Jai Sehgal – 2020A7PS2089H

**Project Name – ParKar Services**

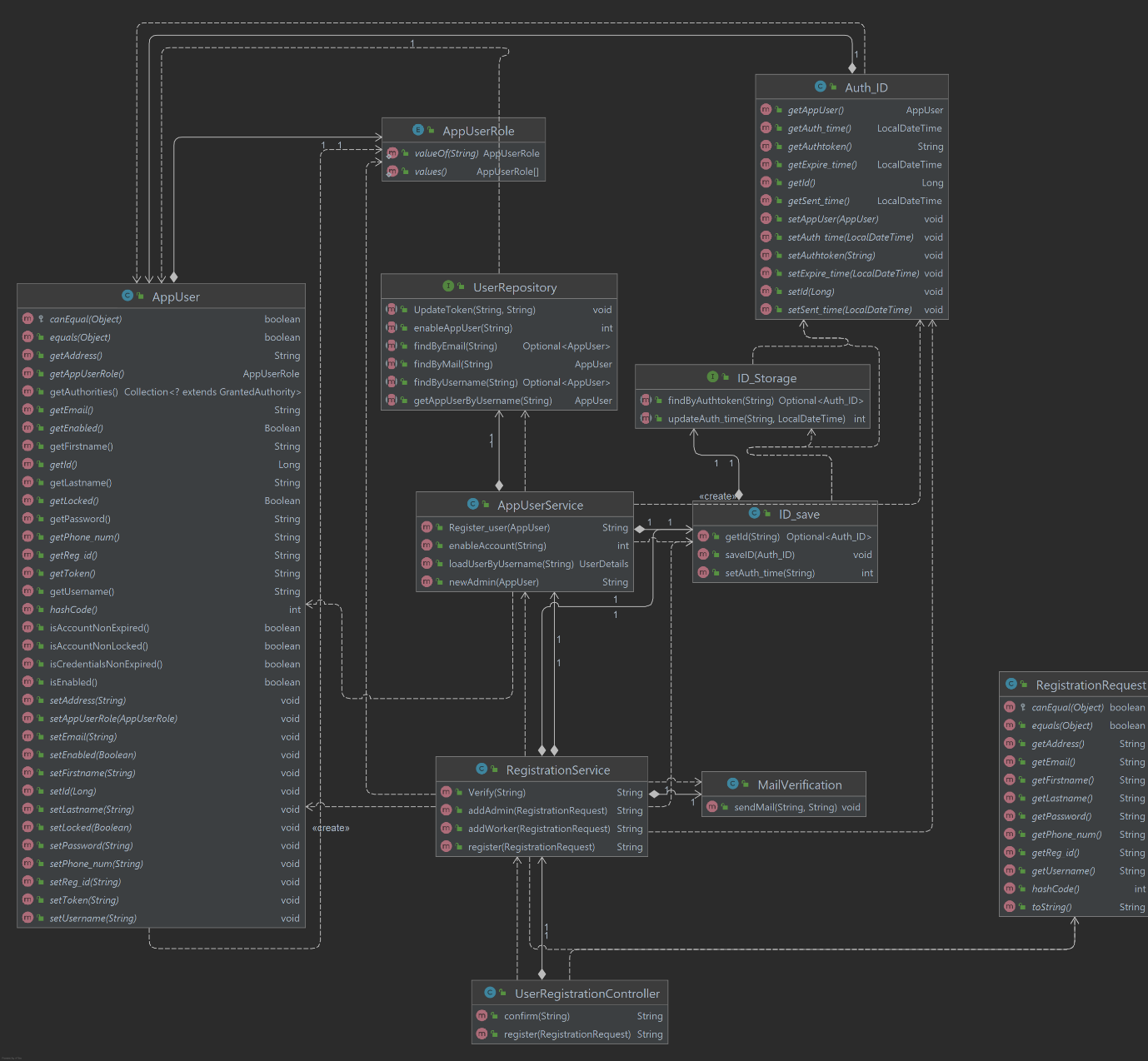
**Frameworks used –**

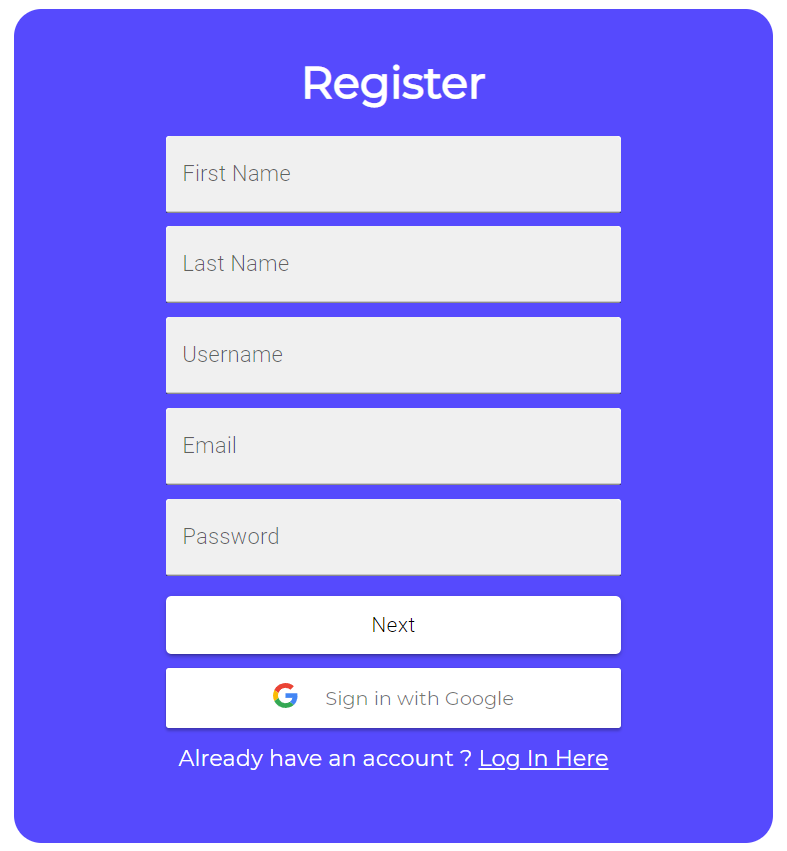
1. **Backend – SpringBoot and REST APIs**
2. **Frontend – ReactJs (User Interface)**
3. **Database – MySql**
4. **Deployed – a. Backend – Tomcat Server (port 8080)**

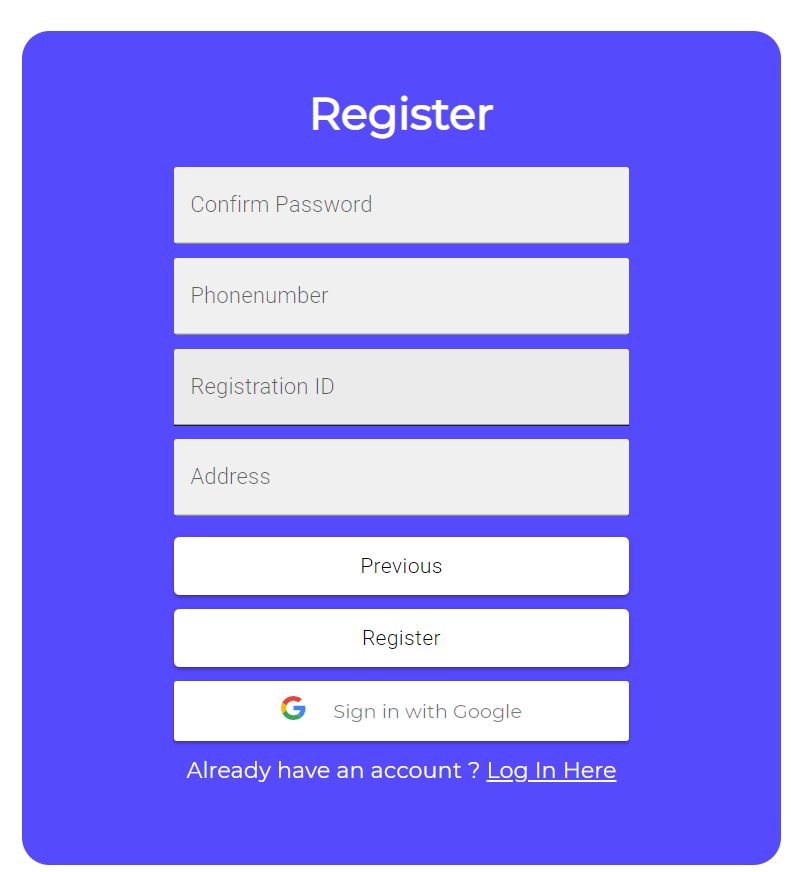
**b. Frontend – React App (port 3000)**

**c. MySql database ( port 3306 )**

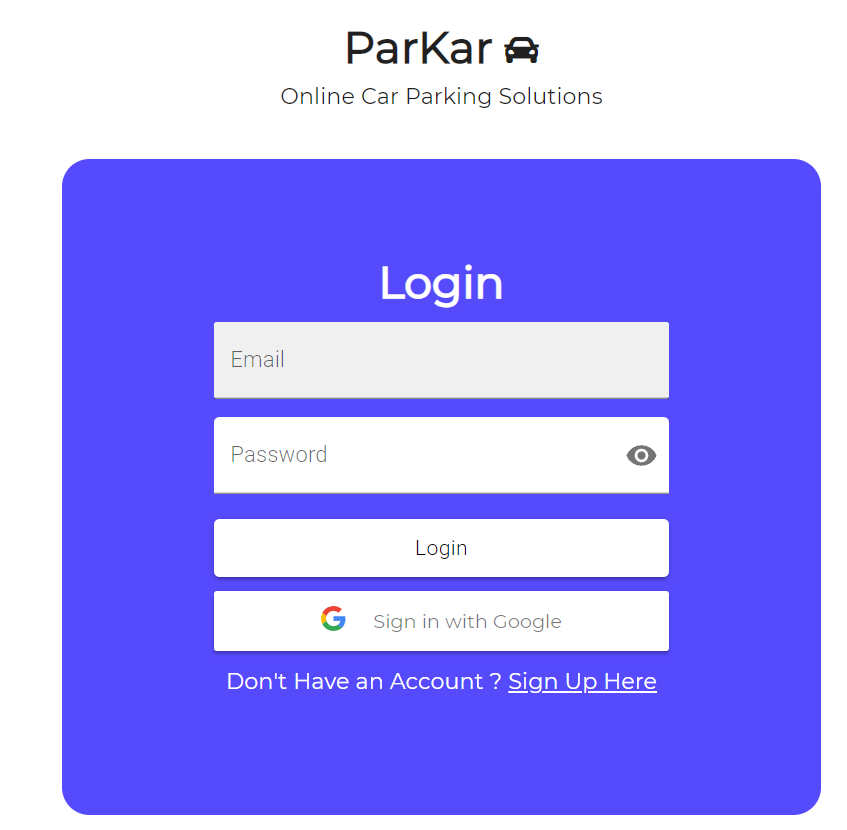
1. **Registration, AuthToken/mail verification and AppUser**

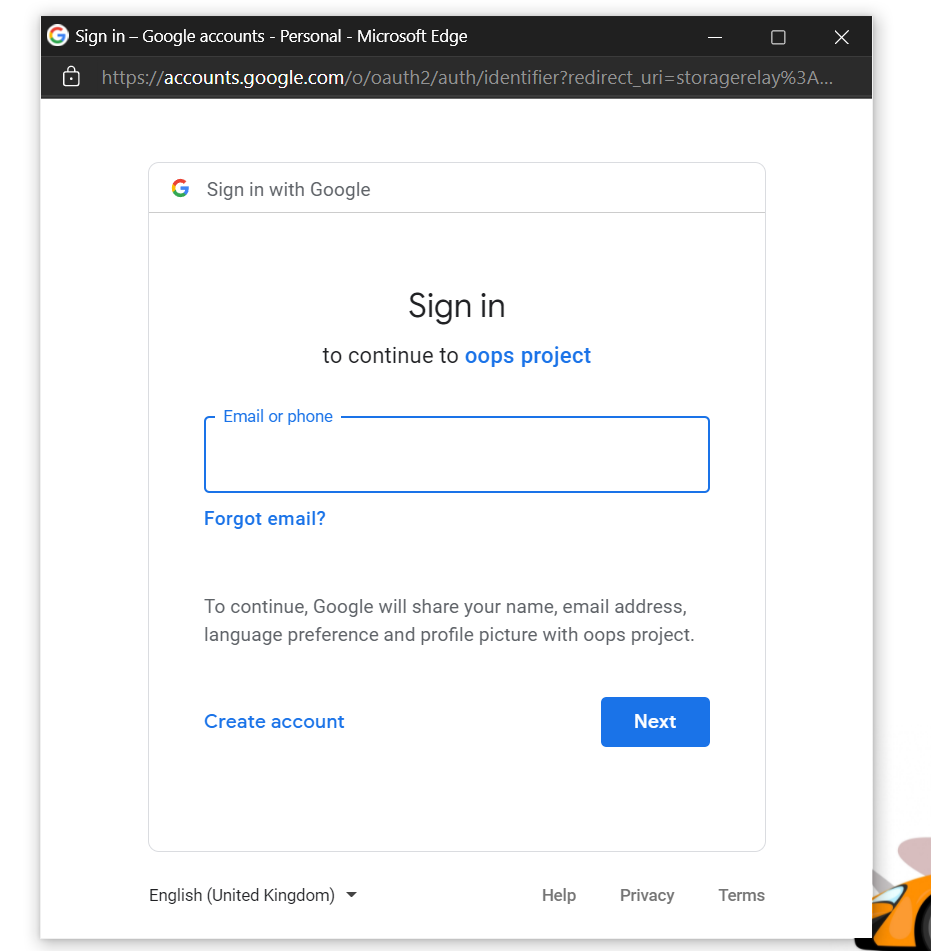


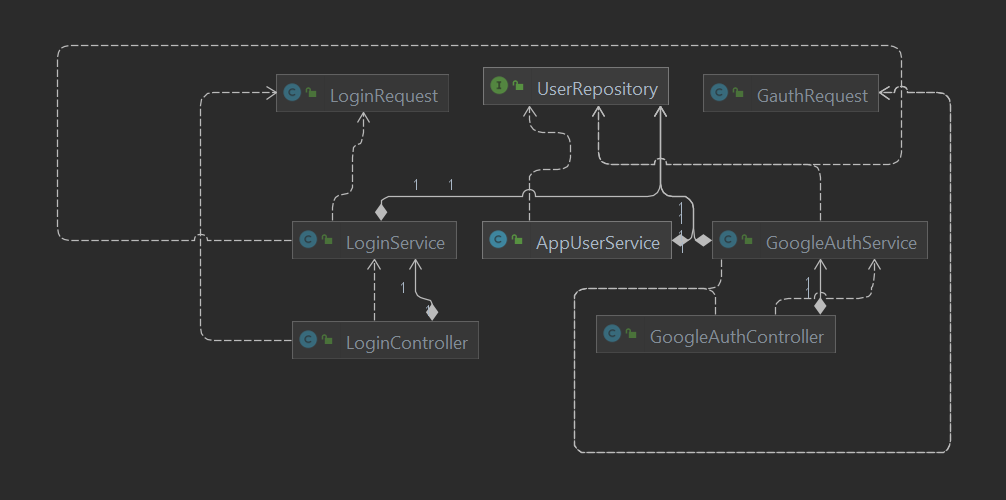




1. **Login and GoogleAuth Controller –**





****

**Implementation details along with User Interface**

We have used ReactJs for user interface and spring boot for java backend.

Server Sided logic implementation –

Terminologies in Classes –

Interfaces are named as “\_\_Repositories” since they extend JpaRepository for database linking purposes.

Classes with name as “\_\_Controller” are the classes that handle POST, GET, PUT, DELETE and various Rest API requests. These classes directly talk with frontend.

Classes with the name as “\_\_Services” are the classes with main backend business logic which are called on by these controllers.

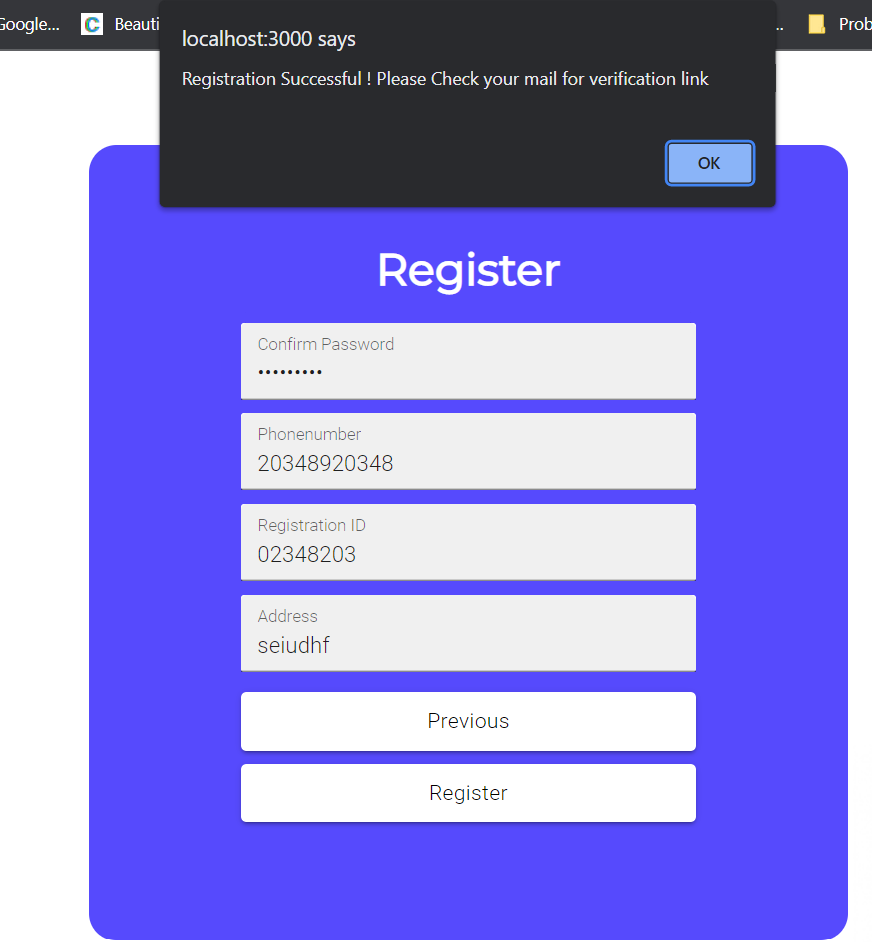
Classes with name as “\_\_Request/Model” govern the format and content of request backend asks from frontend in POST/GET requests (JSON Format)

1. Registration-

We have taken the required details from the user from the frontend which is sent to backend via POST requests. RegisterationController Class gets the user details from the frontend at the endpoint “/register” and calls the register method from the RegisterationService Class. RegisterationService class checks the UserRepository if registration from similar credentials has been done earlier and throws errors with appropriate messages for frontend to catch and display.

If not the User is saved into the database by calling Register\_user function from AppUserService class and a Verification link is sent to User Email via JavaMailSender API, after the verification the account is enabled.

Endpoint in which verification is done is “register/confirm/?id={unique\_token}”

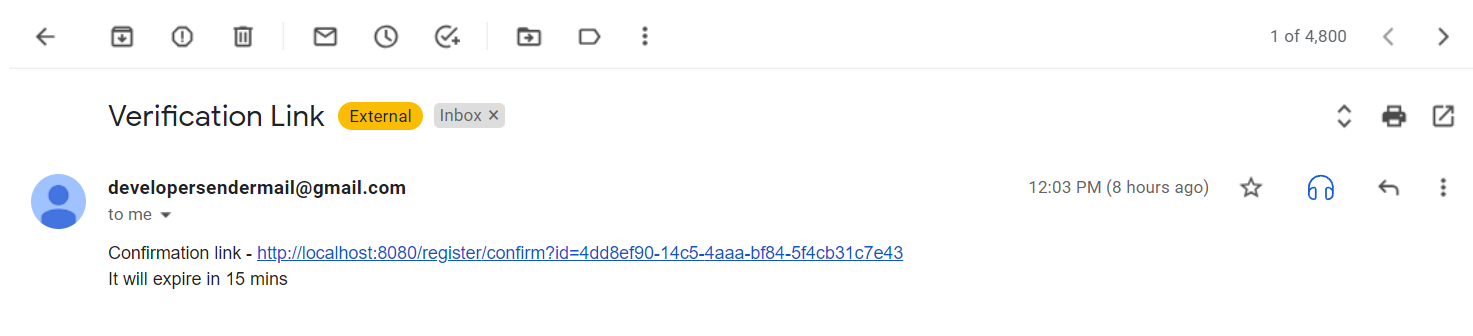


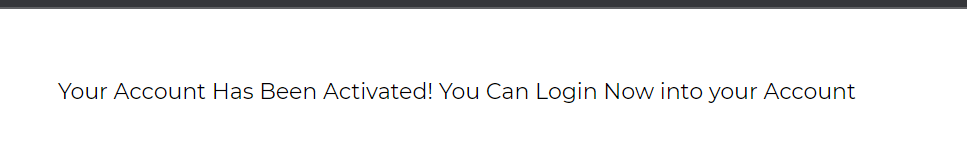
1. Verification/MailSender –

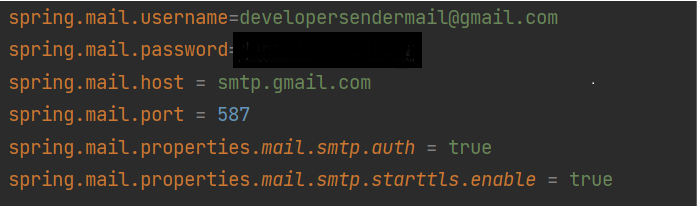
For sending mails for verification purpose we have used JavaMailSender API, for verification purposes there are 2 classes and an interface linked to the database namely Auth\_ID, ID\_save and ID\_storage. We are generating a unique UUID for every user that’s registering into our service, ID\_storage interface provides communication to the database regarding token validation time, expiry time and unique token ID. Once the user clicks on the verification link, the RegisterationController calls up required methods to check if AuthToken is correct and not expired to finally enable User Account.

JavaMailSender is hosted on smtp google servers (port 587).

The Generated App Password is hidden for confidential purposes.







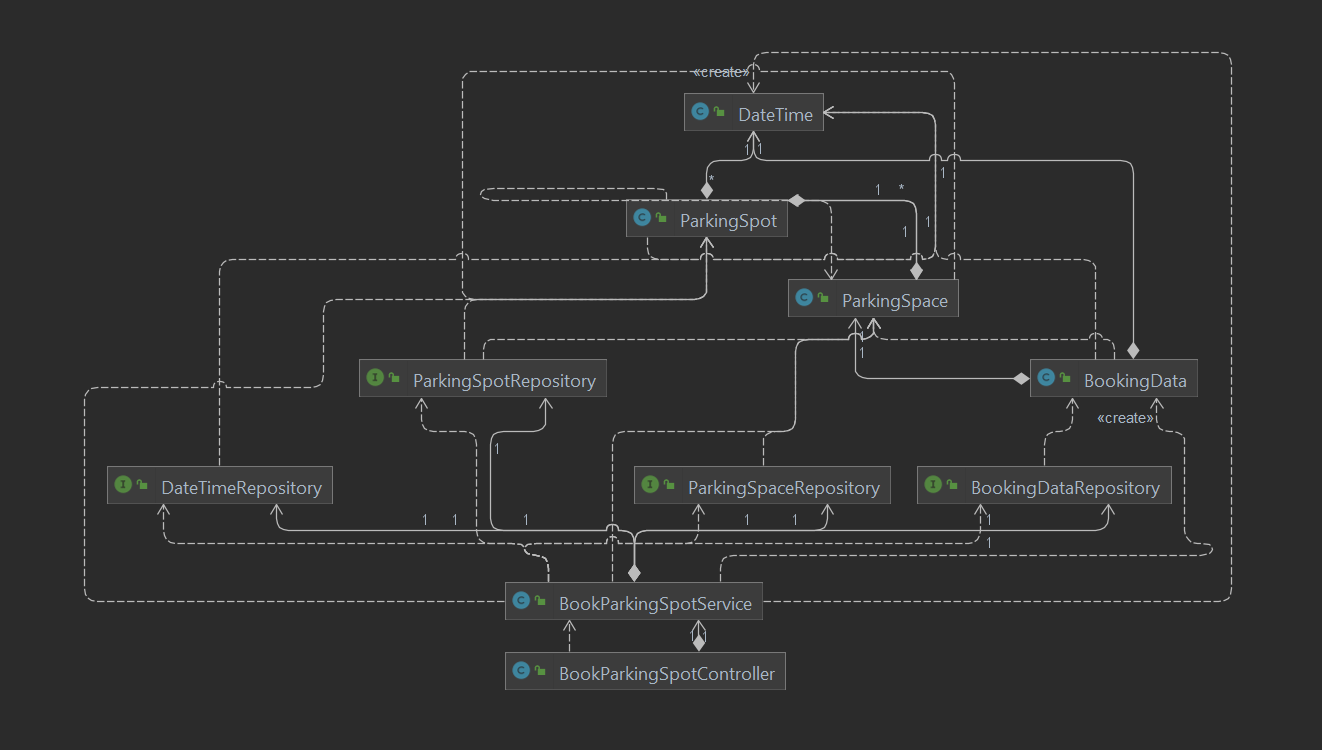
1. Login/GoogleAuth

Similar logic has been used for Login, the Controller gets the User Credentials from the frontend and calls on various methods from Service classes to check if –

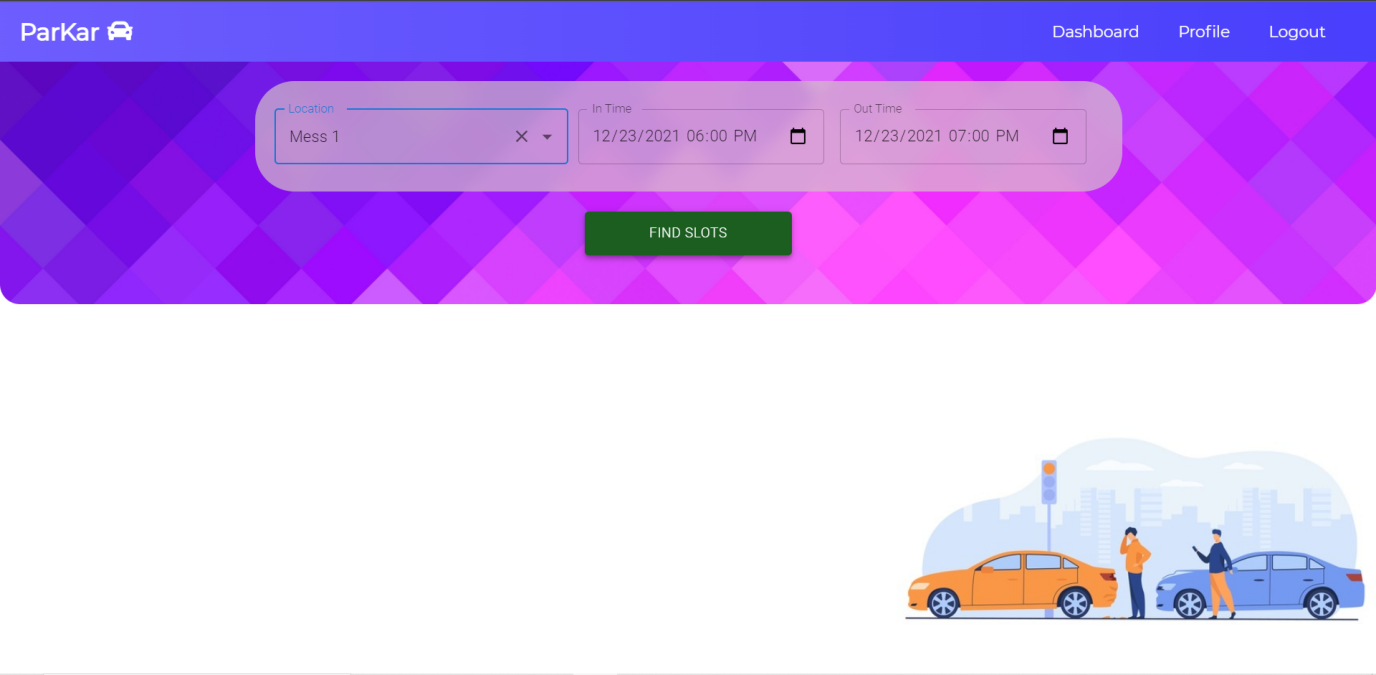
1. If User is in the database
2. If User exist then check for credentials
3. If credentials are correct , check if account it activated.

All of the following checks are easily done using JpaRepository functions and Sql queries. Only after the Backend returns the entire User object along with its role so that frontend can redirect to designated dashboards.

For Google Auth, we have used google developer console that on Authentication provides us a unique token for every Gmail user that signs in along with name and id. If the user is a first-time user, then it’s he/she is redirected for a shortened Registration and their token is updated, else the token is checked by backend and user object is sent to frontend for sign in.

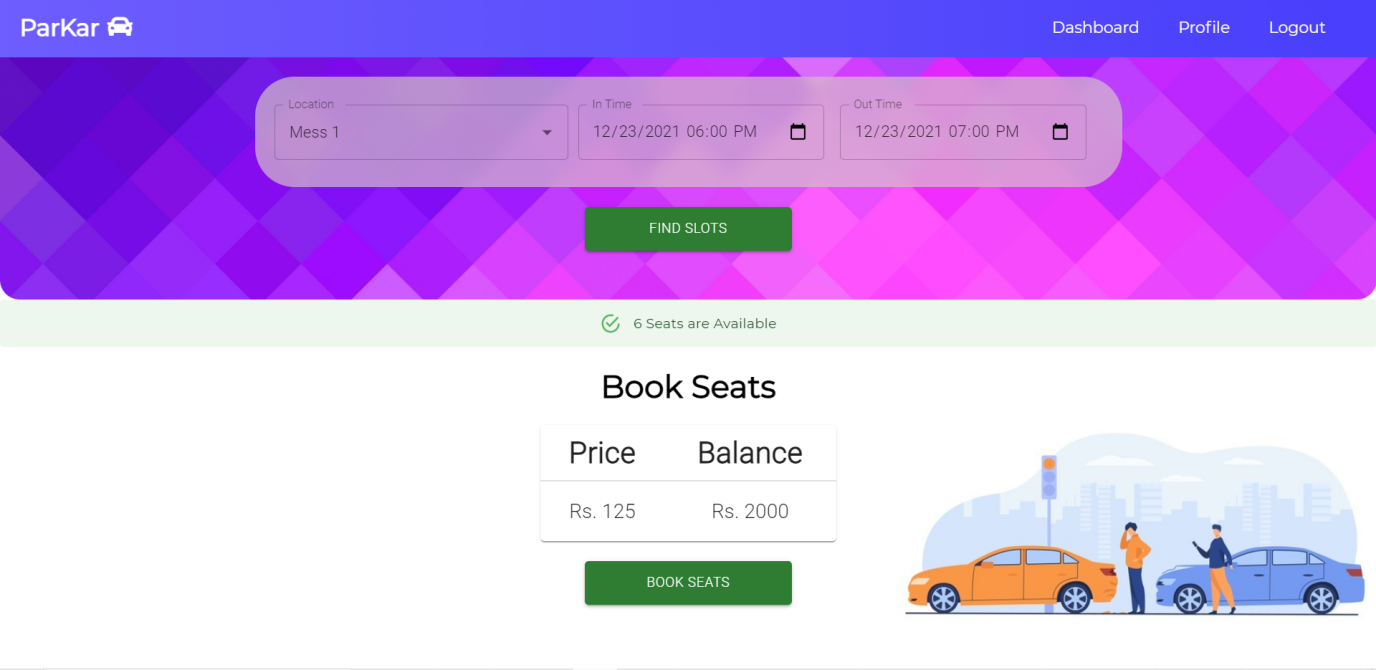
1. **Checking Availability of Slots and Booking**`

User DashBoard Booking Spot -

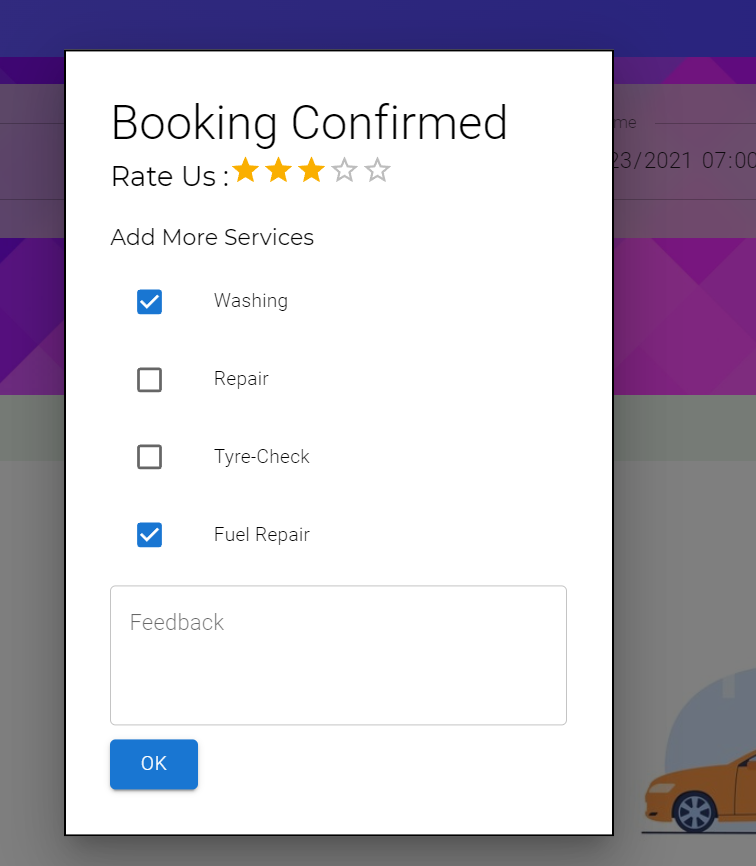


* 1. We have taken the required details from the user from the frontend which is sent to backend via POST requests. This information contains data about Parking Space , in Time and out Time. BookParkingSpotController gets the booking details from BookInfo model through the endpoint “/book/avialablespot”. This calls the BookSpace method in BookParkingSpotController which calls the BookSpot method in BookParkingSpotService. BookSpot method checks if there are any available spots in a particular Parking Space at a particular time.

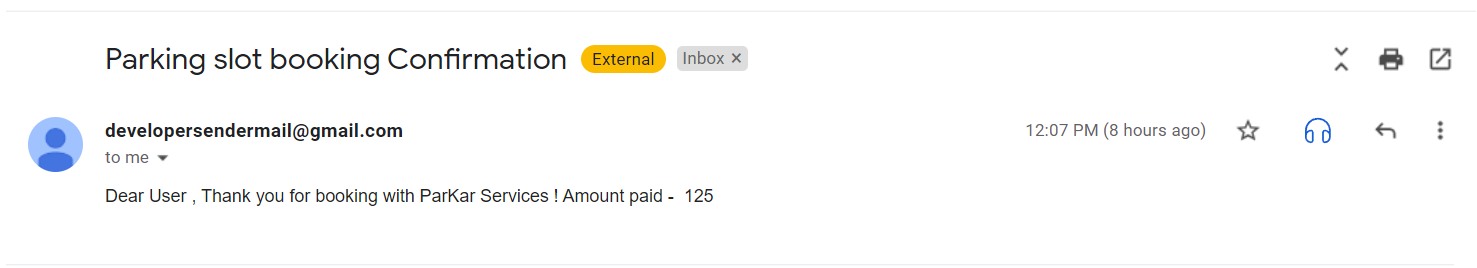
Booking Confirmation -



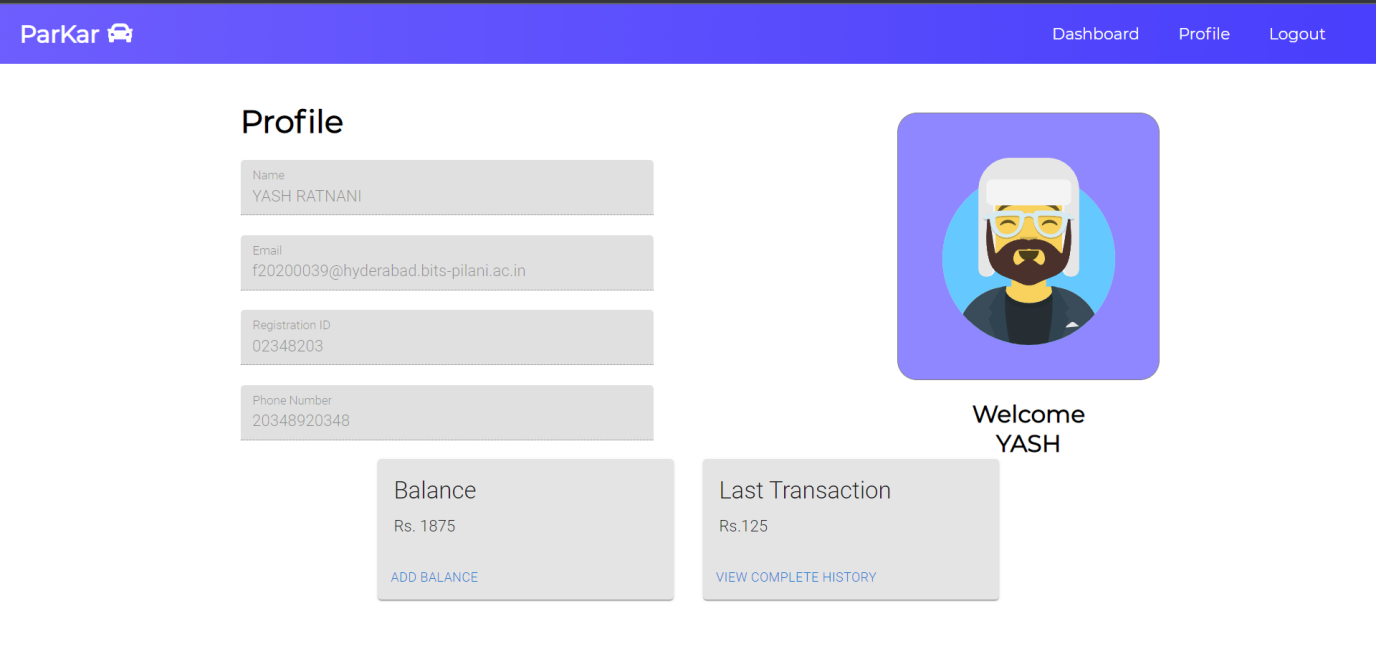
* 1. If the spot is available user gets an option to confirm their booking which calls the endpoint “/book/confirm”. This calls the method confirmBooking in BookParkingSpotController which in turn calls the method bookSpot in BookParkingSporService. This again checks if any space is available or not else throws an error. If the space is booked, amount is deducted from the user’s balance and the user gets a mail that their booking is confirmed and also the amount deducted through JavaMailSender API.

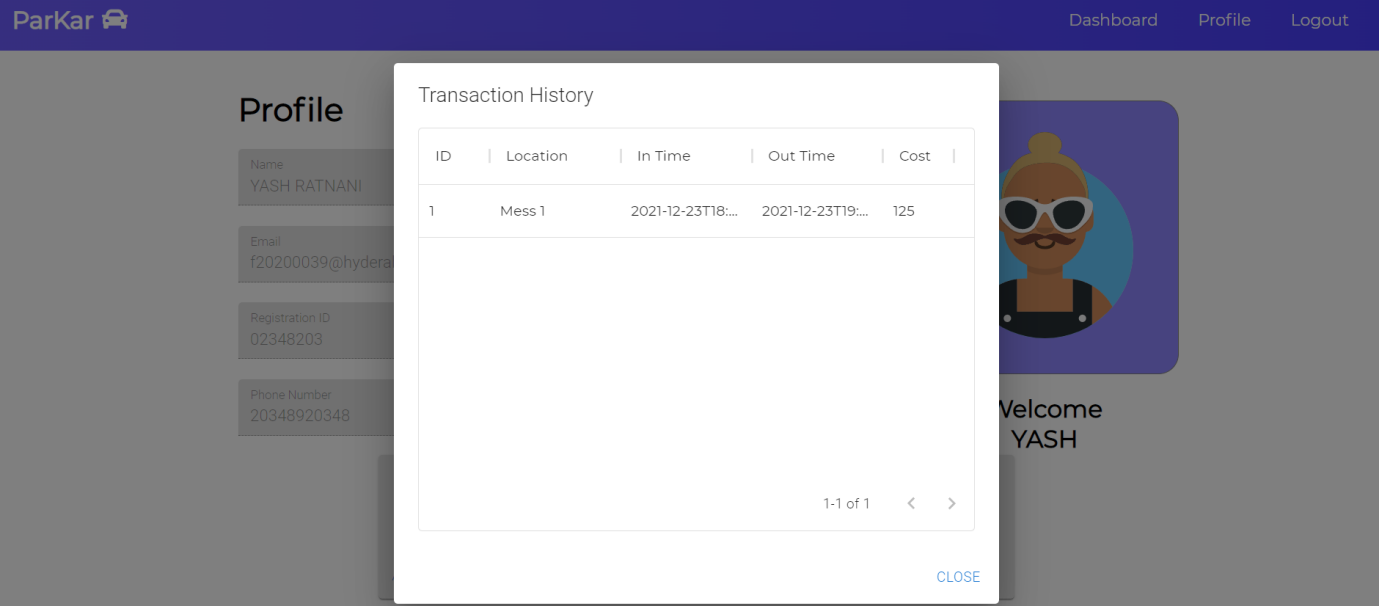


Booking Confirmation Mail -

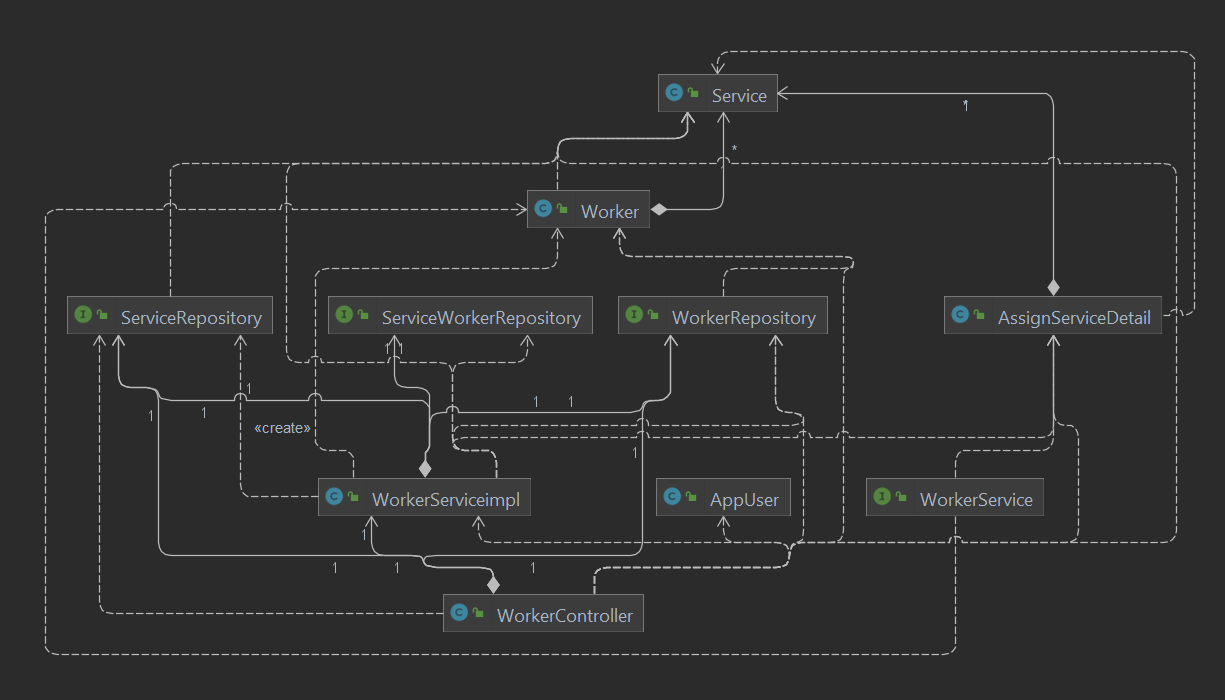


User Profile with Transaction History -





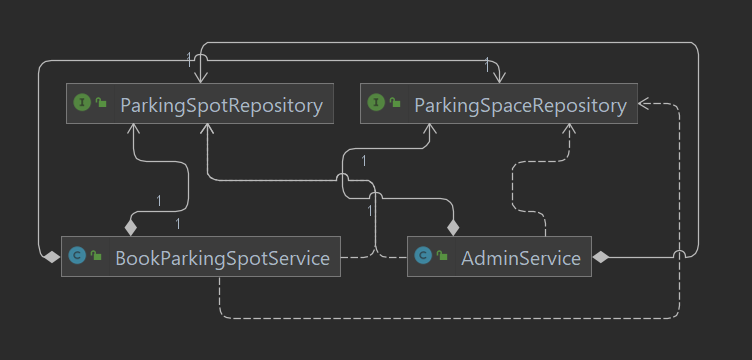
1. **Worker-Services and Admin**

****

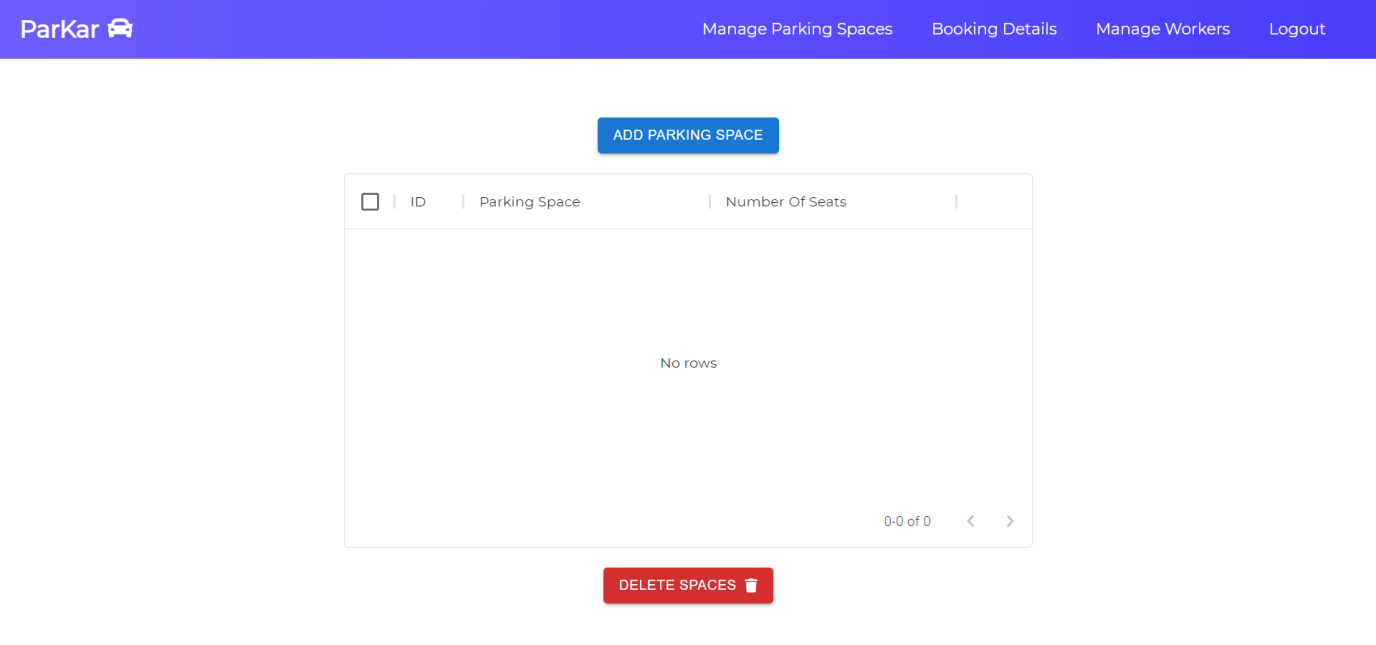
* 1. Worker Controller has all the endpoints related to worker and services provided. Worker controller calls worker service implementation which Autowires ServiceRepository, ServiceWorkerRepository, WorkerRepository.

Admin has the option to add worker. Add worker calls the endpoint “/worker/add” which calls the method add in WorkerController. This method saves worker information in the database by calling various repository interfaces. Admin also has the option to delete worker, delete worker calls the end point “/worker/delete/{Id}”. Id is sent by the frontend which is used to delete the worker having that Id by calling WorkerRepository.

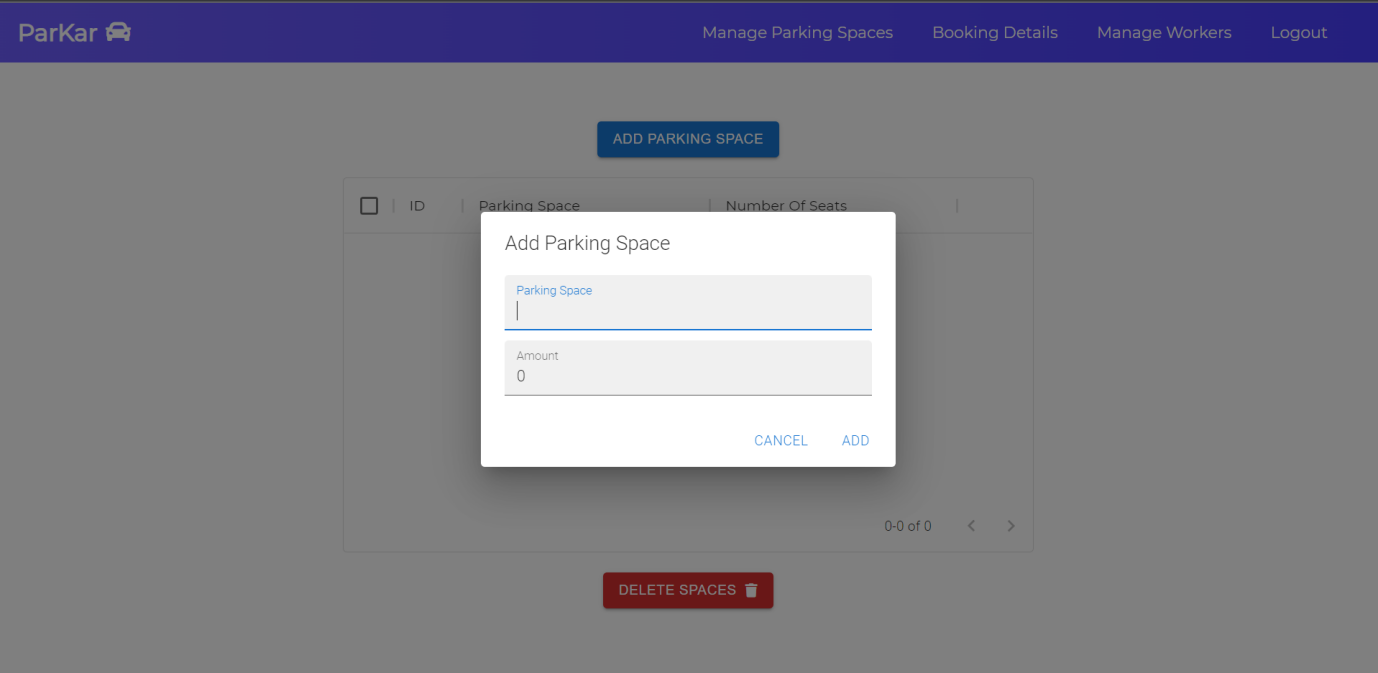
* 1. User gets an option to assign services after booking is confirmed. This is done through the endpoint “/worker/assignWork”. This call the method assignWork in WorkerController which in turn calls the method work in WorkerServiceImpl. This method contain logic to assign the work to a worker which has minimum work at that time in that parking space.
  2. To check the assigned work in worker dashboard, getWorker method is called. This returns the service assigned to the specific worker at different times.
  3. To check the services of all the worker admin can call the method allWork through the endpoint”/worker/alWork”. This calls the method getAllWork present in WorkerServiceImpl.
  4. Admin has access to all details regarding current Booking using GET mapping.



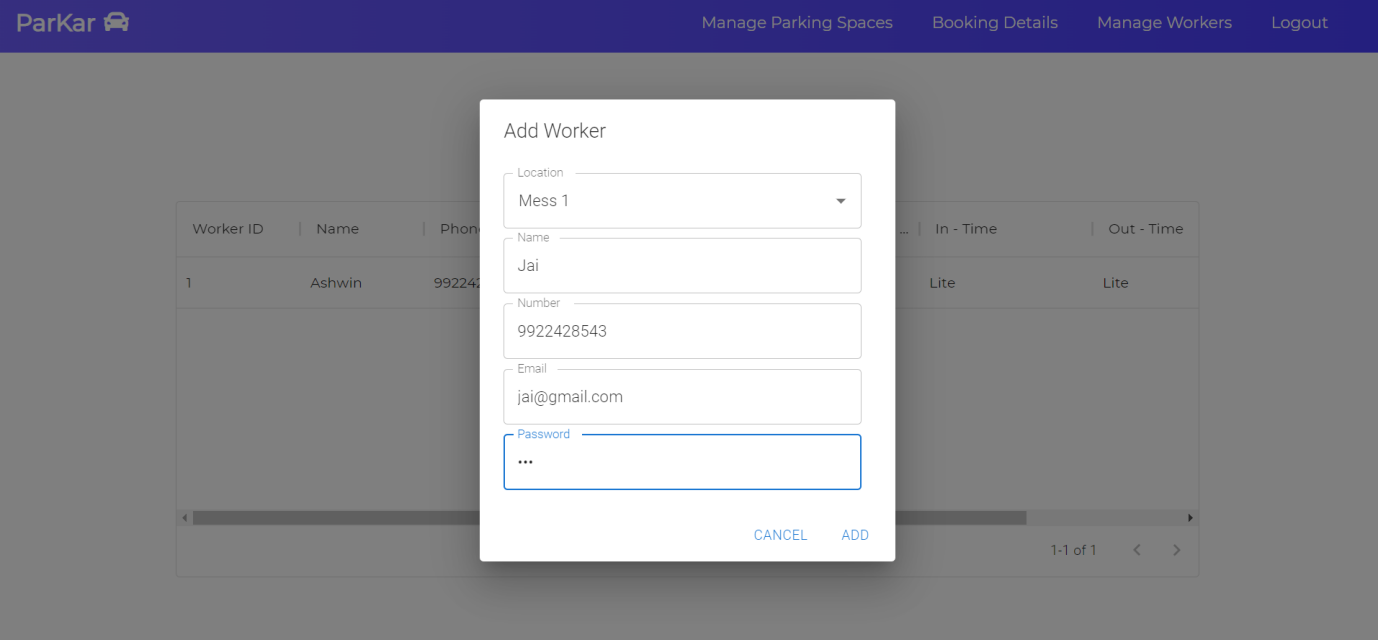
Admin DashBoard -



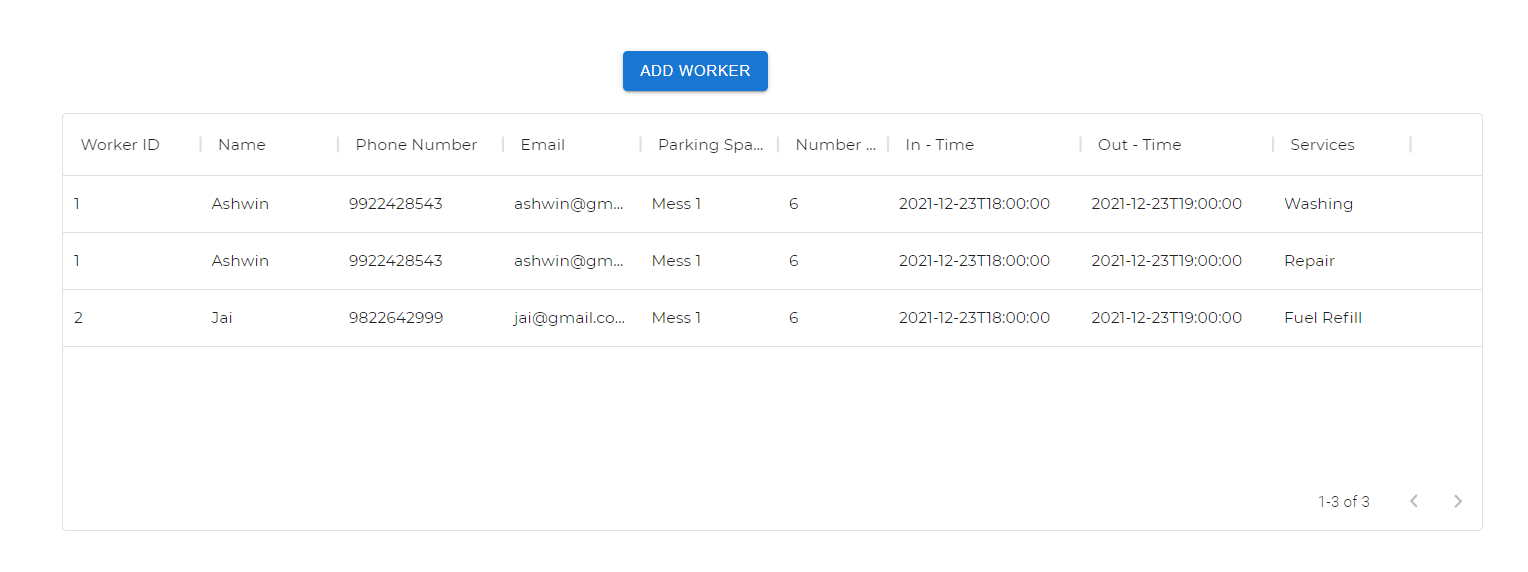
Adding and deleting parking spaces with desired amount of parking spots

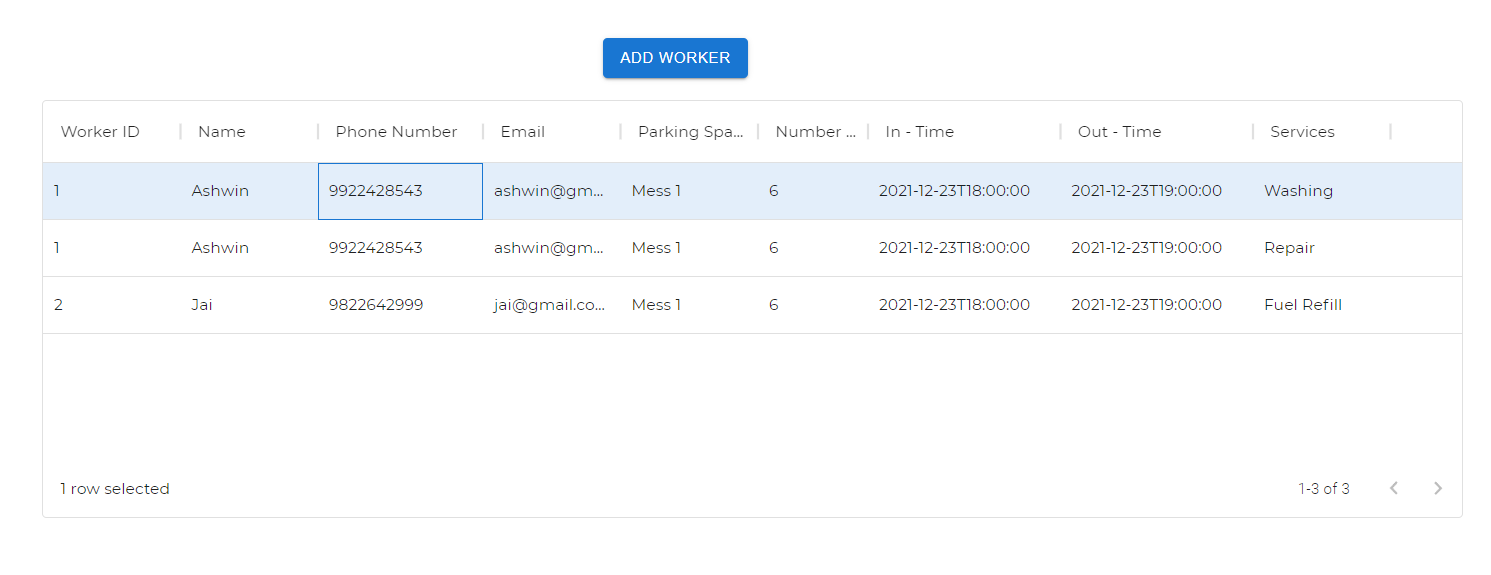


Admin Dashboard - Editing Workers and checking assigned work -



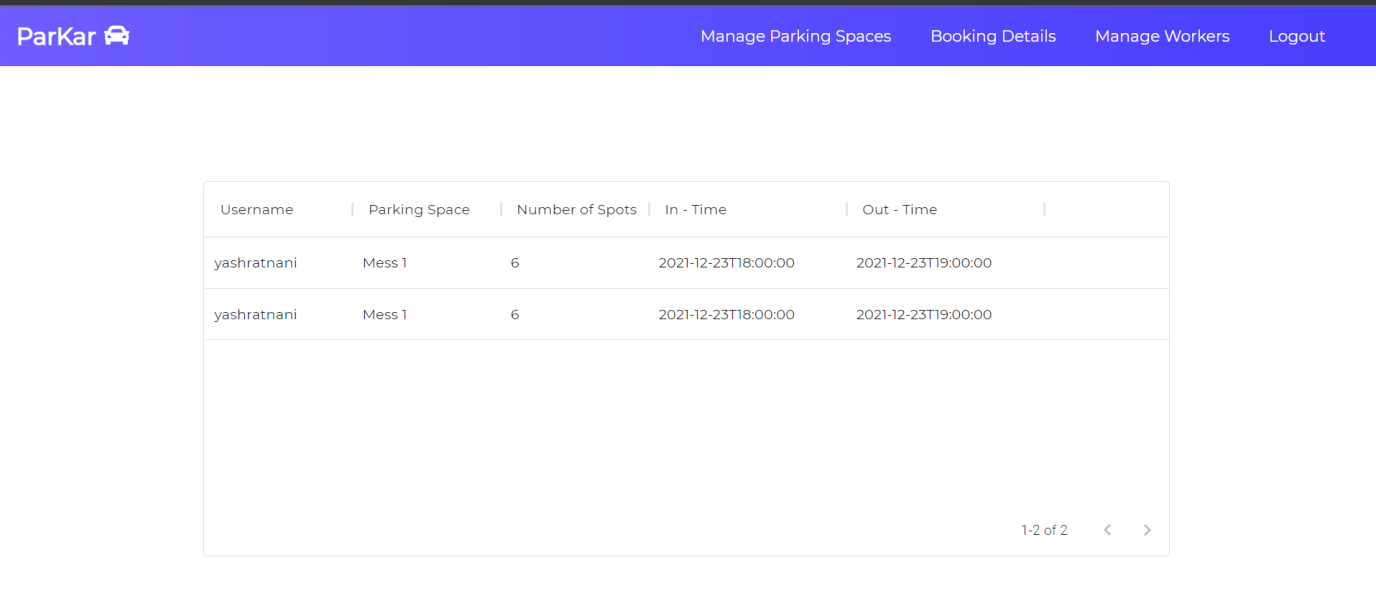
Worker Task Assignment after Spot Booking -



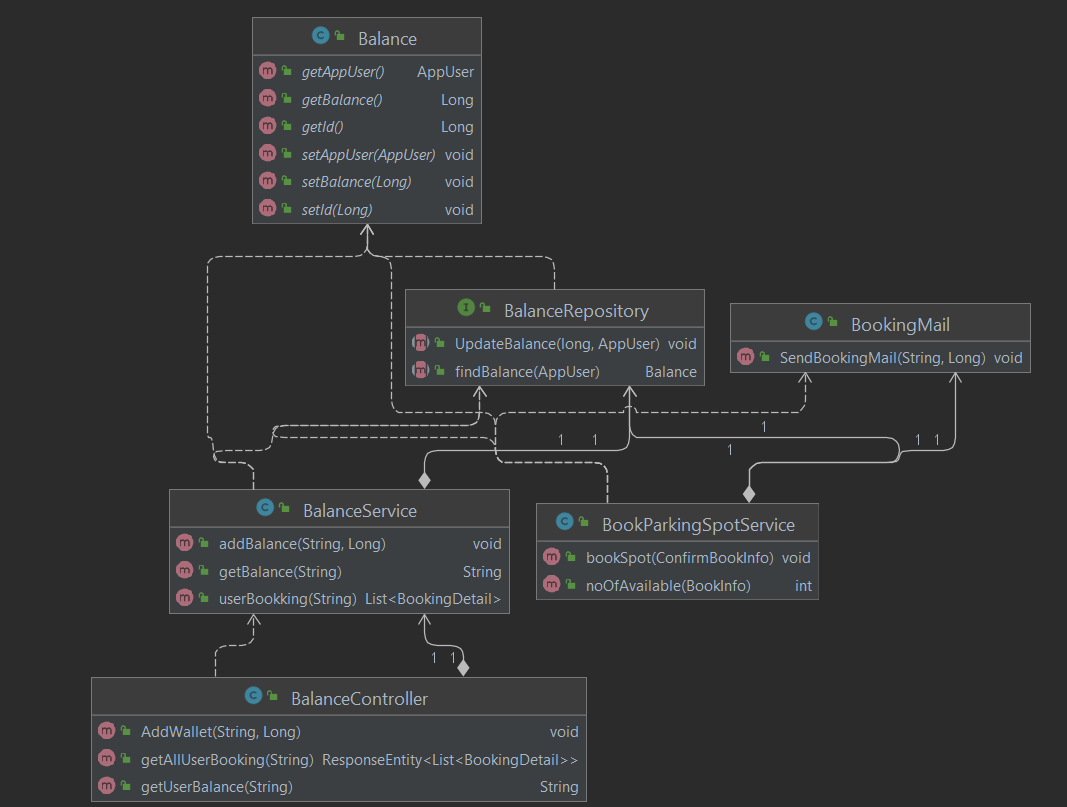


On selecting the worker and pressing delet , the frontend sends a DELET Request to the backend , namely WorkerController class which calls on JpaRepository functions to delet all instances of worker from every used table.

Admin has control over entire parking services , Admin can call up on respective ParkingSpaceService and ParkingSpotService for adding deleting and getting all the Details.

Entire booking detail of the System is shown to admin - 

1. **User Balance(Wallet) , Payment Feedback Mail and Real-Time-Payment portal (RazorPay)for adding to Balance**

****

BalanceRepository is the Interface which directly connects backend to the database. The Balance table in the database includes User along with UserBalance and UserID.

BalanceController is the controller class that directly links with frontend using POST and GET requests and call on BalanceService class to perform various actions namely addBalance , getBalance and userBookingDetails. The following end-points are used for the same –

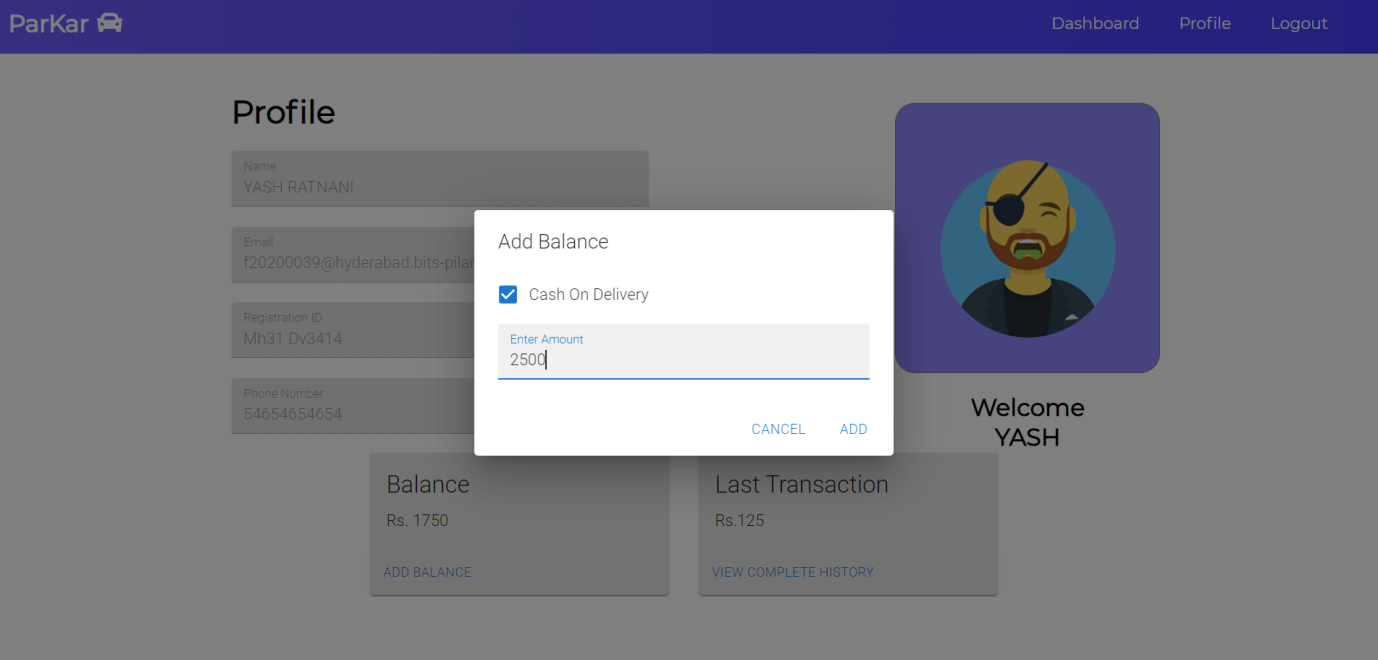
”/balance/{username}” for getting current user balance.

“/userbooking/{username}” for getting user booking details to show previous User Bookings.

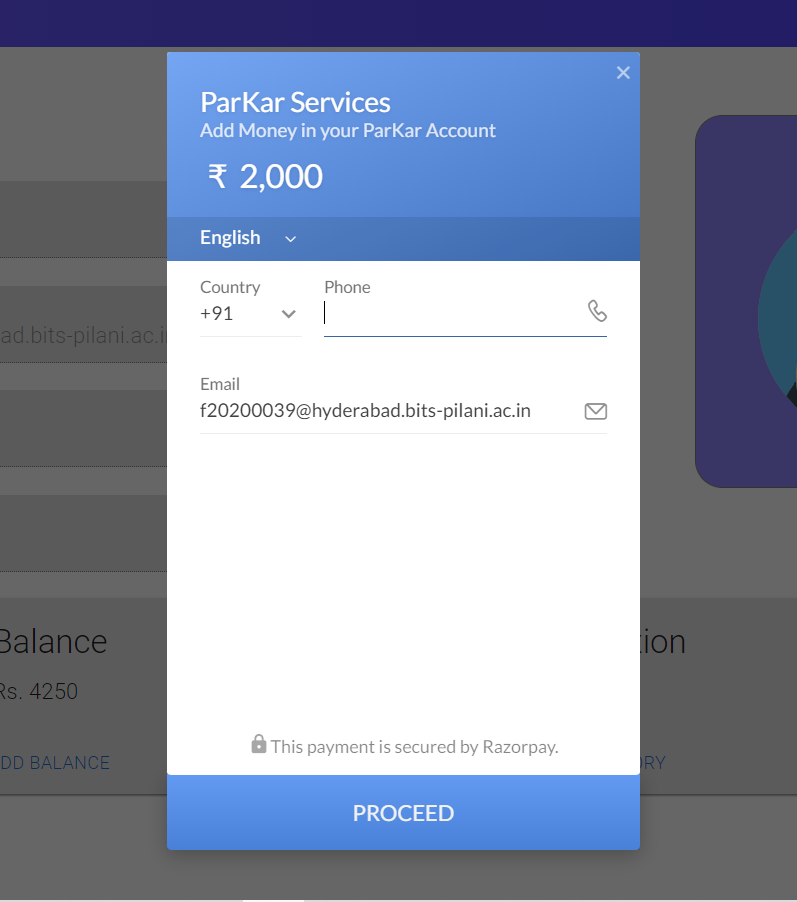
Once the BookParkingSpotService successfully books a spot for the User , BalanceRepository is called on to update the balance of the user after the calculated deductions for the booking which is done using sql modifying queries and its checked if User has enough Balance in the wallet to book the spot. After the deductions , it calls upon BookingMail class which uses JavaMailSender to send Confirmed BookingInfo with amount paid mail to the User as shown in Slot Booking Module.

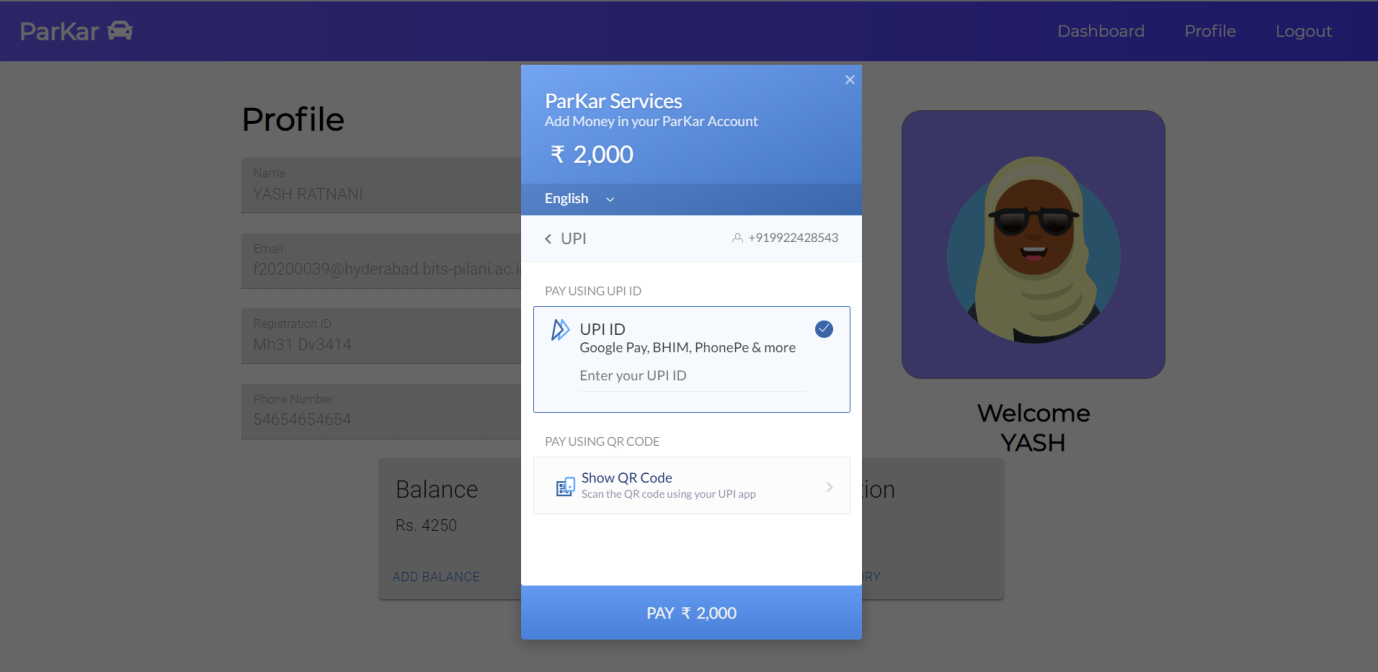
For Adding Balance we have incorporated real time payment portal Rayzorpay

Rayzorpay provides unique API key for developers which is incorporated for payment services in the project.

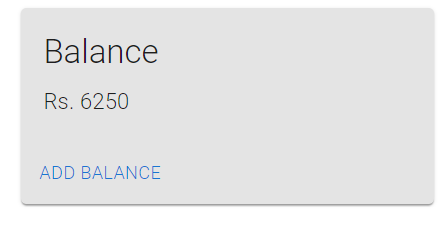


Using Real Time Payment Portal for adding balance to the wallet.





On confirming the UPI Request from your phone , the amount gets deducted and transaction is successful and the balance is finally updated.



Database MySQl Workbench -

