DOCUMENT OF PARKING SYSTEM

Submitted by : Batool Fatima

Submitted to : Nabeel Ahmad HotelKey

It is a system in which we provide a garage in which users can park their cars.To park the vehicle, user first needs to register his vehicle to the system the he can park in or out otherwise he would be asked to first register . Whenever user get his car registered or parked in/out, he will be notified by the system via email . Also if a registered car is not parked till 1am , he will be notified to park the car. The workflow of this system is shown below.

INPUT

User enters data through postman requests i.e. Post and Get requests
Total 3 types of requests can be sent, 2 POST (registration, parking)in which data is sent in payload and the other one is Get (park_history) which sends the name of vehicle in query params and get history as response.

PROCESSING

Total 9 classes are used in this project named:

- Resources
- Controller
- Database
- Notification
- Registration (obj)
- Parking (obj)
- Validation
- Scheduler
- Job

REGISTRATION: This class has 6 data members named first_name, last_name, email, contact no, vehicle no, garage no . It has all the getter, setter, constructors etc.

PARKING: This class has 4 data members named vehicle_number, park_status, date, time. It also has all the relevant methods in it.

VALIDATION: This class has 2 methods one for validating registration object and other for parking object.which validates either the data members are correct i.e. email is correct, contact number is validate, vehicle numbers is authentic etc

RESOURCES: It has 3 methods: Registration, Parking, History.

- **Registration:** first it converts the payload in the registration object using the Gson object. Then the object will be passed to doRegistartion method of Controller class which returns an integer and on the basis of that int value , Registration method returns response along with status to the postman i.e 200 for successfully registered, 200 for already registered and 400 for invalid input (payload).

- **Parking:** first convert payload into parking object using Gson . Then the object is passed to doParrking method of controller class which in return gives an integer based on what happened on the backend either car got parked or not or already parked.Then it returns 6 types of responses i.e. 400 for invalid payload, 400 for u need to register the car before parking, 200 for successfully parked in, 200 for successfully parked out, similarly 200 for already parked inor out.

- **History:** It gets the vehicle numbers as query param, first it checks the database whether this car is registered or not .If not then return 400 status with message i.e can't have any history as this car is not registered in the system.On the other hand if found registered in the system , then call carHistory method of controller class which returns a string containing the history which got return to postman

CONTROLLER: This class also contains 3 methods i.e. static doRegistration, doParking, carHistory.

doRegistration: gets registration object from resources method.Do validation by passing to checkRegistration of validation class. If validates then pass the object to register method of database class which returns int i.e 1 for already registered and 2 for successfully registered.On the basis of that this method returns either 2 or 3. Else if validation fails then it will return 1 i.e. invalid data.

- **doParking:** gets parking object from resource method parking. Do validation If validate then pass to parking method of database class which returns integer on basis of what happened on the backend i.e either stored in db or not or already in or out etc.Else if validation fails , it will returns 1.

- **carHistory:** gets vehicle number as parameter and call check_history method of database class which returns string which could be null or not on basis of what db returns.

DATABASE: This class contains 6 methods i.e. RegisterInfo, ParkingInfo, check_parking, check_registration, check1amAlarm, check_history.

- **RegisterInfo:** first call check_registration method to check whether vehicle is already registered or allowed to register. If already registered do nothing else if it update the database by entering the object details to the database.

- **ParkingInfo**: this method gets parking object it check whether vehicle is already in/out or allowed to park in/out.

- **Check_parking**: this method checks if the vehicle is already in/out.

- **Check_registartion:** checks whether the vehicle number entered in payload of registration is already registered or not

- **Check1amAlarm:** This methods runs by scheduler on the given time and it retrieves the email address of those vehicle users which are not parked till 1am.

- **checkHistory**: retrieves all the records of a certain vehicle from db and store them in an arraylist and then return that list.

NOTIFICATION: sendEmail method uses Java mail APi to send email to the address sent n parameters.

SCHEDULER: This class uses quartz scheduler in which first we create a job which we wants to be scheduled on a certain time . Then a trigger object is made which sets the time on which job has to be scheduled.Then scheduler is start , then we schedule a job.

JOB: This class implements Job interface in which execute methods need to be override i.e. we call the check1amAlarm method of database class which returns the emails which needs to be notified . Then those emails got notified by send email method of notification class.