Informatics Institute of Technology
Department of Computing

BEng: Software Engineering

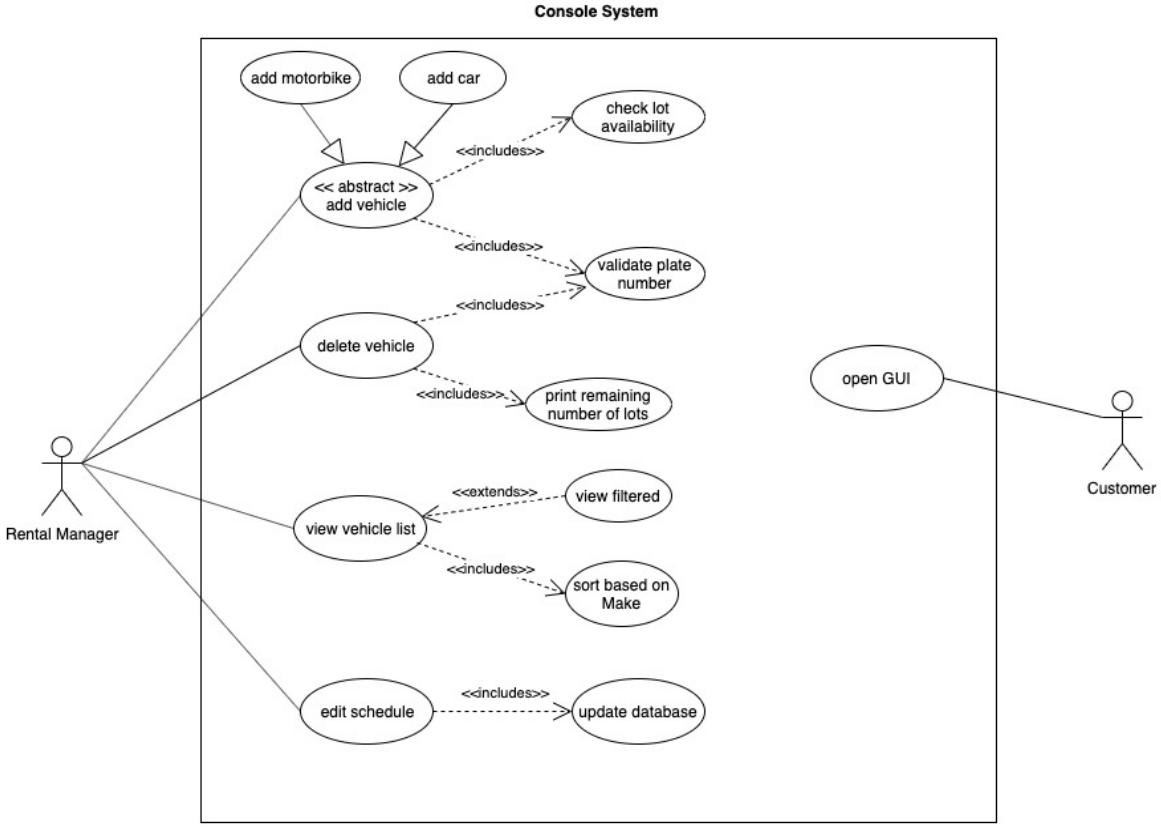
MODULE: 5COSC001W: Object Oriented Programming

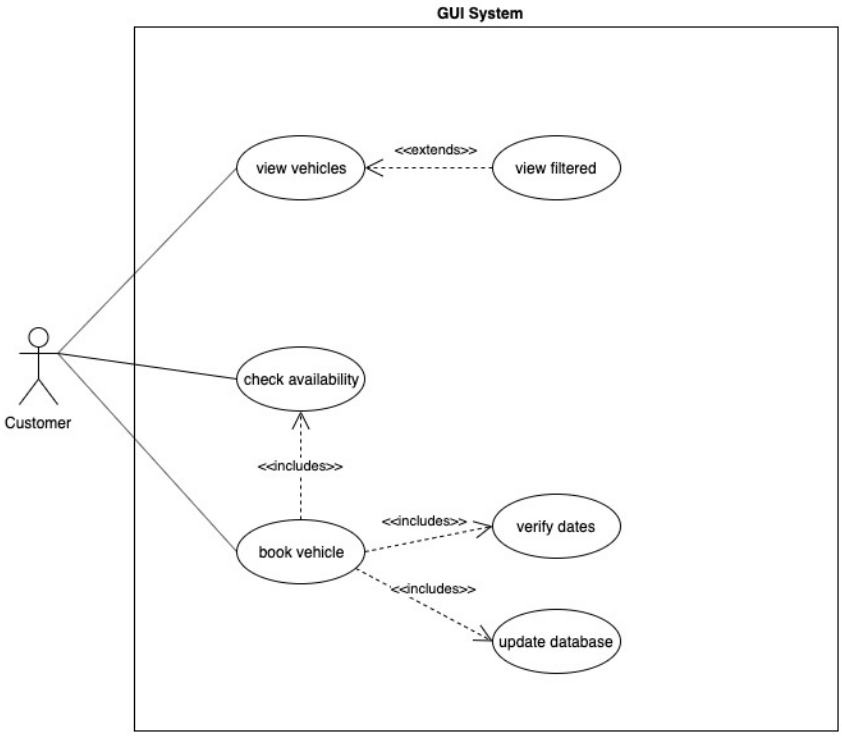
Coursework Report

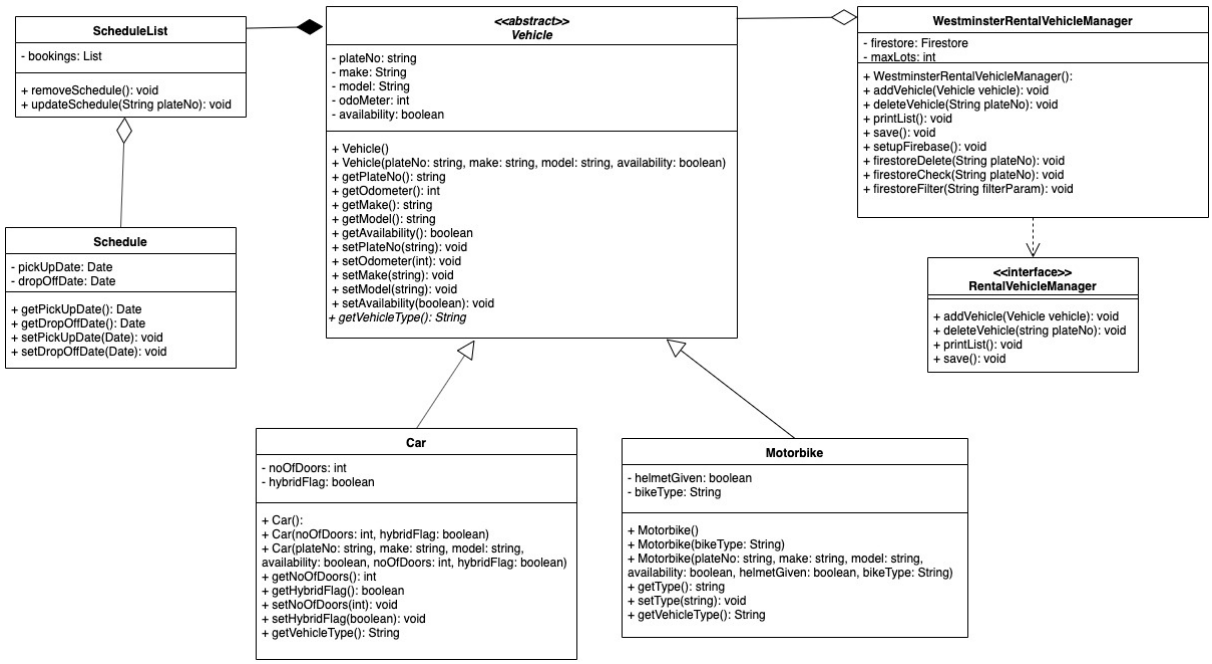
Tutorial Group	: Group G
IIT student ID	: 2018516
UoW student ID	: W1742286
Student Name	: Nujitha Wickramasurendra

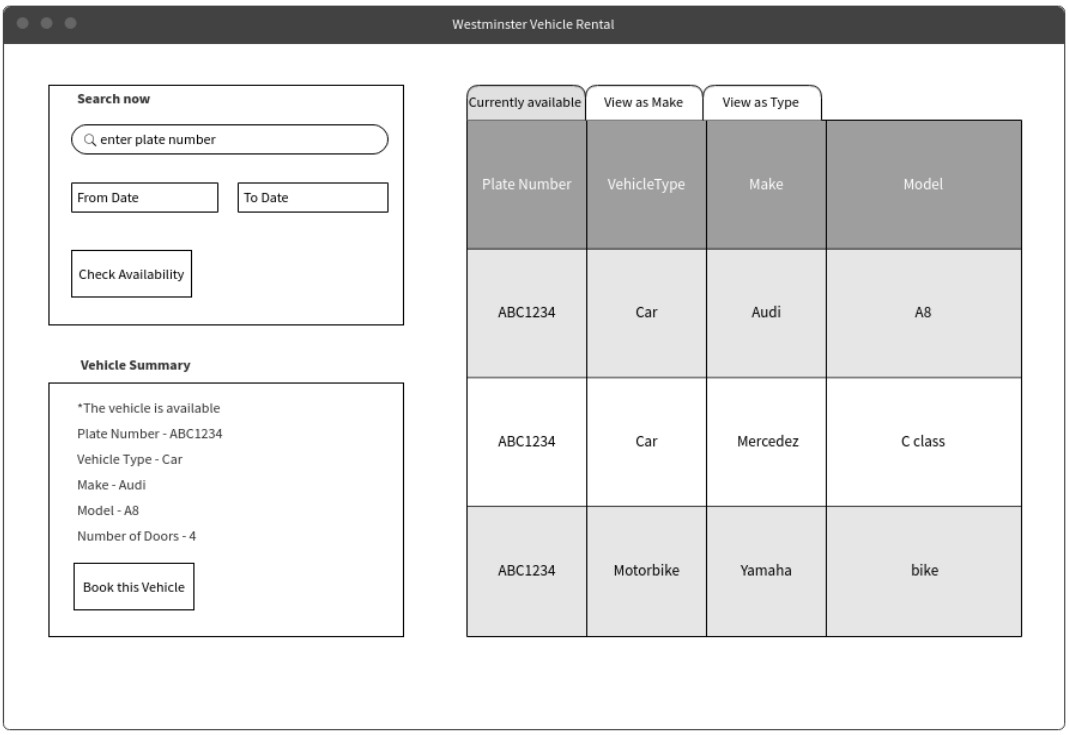
Table of Contents

1. Design solutions









Wireframe for GUI

2. Console System

2.1 Vehicle Class

```
package com.nujitha.project.classes;

import java.io.Serializable;
import java.util.Objects;

public abstract class Vehicle implements Comparable<Vehicle>, Serializable
{
    //class attributes
    private String plateNo;
    private String make;
    private String model;
    private int odoMeter;
    private boolean availability;

    //Constructor
    public Vehicle() {

    }

    public Vehicle(String plateNo, String make, String model, int
odoMeter, boolean availability) {
        super();
        this.plateNo = plateNo;
        this.make = make;
        this.model = model;
        this.odoMeter = odoMeter;
        this.availability = availability;
    }

    //getters and setters
    public abstract String getVehicleType();

    public String getPlateNo() {
        return plateNo;
    }

    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        this.make = make;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }
```

```
    public int getOdoMeter() {
        return odoMeter;
    }

    public void setOdoMeter(int odoMeter) {
        this.odoMeter = odoMeter;
    }

    public boolean isAvailability() {
        return availability;
    }

    public void setAvailability(boolean availability) {
        this.availability = availability;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Vehicle vehicle = (Vehicle) o;
        return odoMeter == vehicle.odoMeter &&
                availability == vehicle.availability &&
                Objects.equals(plateNo, vehicle.plateNo) &&
                Objects.equals(make, vehicle.make) &&
                Objects.equals(model, vehicle.model);
    }

    @Override
    public int hashCode() {
        return Objects.hash(plateNo, make, model, odoMeter, availability);
    }

    @Override
    public String toString() {
        return "plateNo='" + plateNo + '\'' +
                ", make='" + make + '\'' +
                ", model='" + model + '\'' +
                ", odoMeter=" + odoMeter +
                ", availability=" + availability ;
    }

    @Override
    public int compareTo(Vehicle other) {
        return make.compareTo(other.make);
    }

}
```

2.2 Car Class

```java
package com.nujitha.project.classes;

import java.util.Objects;

public class Car extends Vehicle {
    private int noOfDoors;
    private boolean hybridFlag;
    private String vehicleType;

    //constructor
    public Car(String plateNo, String make, String model, int odoMeter,
boolean availability, int noOfDoors, boolean hybridFlag) {
        super(plateNo, make, model, odoMeter, availability);
        this.noOfDoors = noOfDoors;
        this.hybridFlag = hybridFlag;
    }

    @Override
    public String getVehicleType() {
        return vehicleType = "Car";
    }

    //getters and setters
    public void setVehicleType(String vehicleType) {
        this.vehicleType = vehicleType;
    }

    public int getNoOfDoors() {
        return noOfDoors;
    }

    public void setNoOfDoors(int noOfDoors) {
        this.noOfDoors = noOfDoors;
    }

    public boolean isHybridFlag() {
        return hybridFlag;
    }

    public void setHybridFlag(boolean hybridFlag) {
        this.hybridFlag = hybridFlag;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;
        Car car = (Car) o;
        return noOfDoors == car.noOfDoors &&
                hybridFlag == car.hybridFlag;
    }

    @Override
    public int hashCode() {
        return Objects.hash(super.hashCode(), noOfDoors, hybridFlag);
    }

    //toString method
```

```
    @Override
    public String toString() {
        return super.toString() + " noOfDoors=" + noOfDoors +
                ", hybridFlag=" + hybridFlag +
                ", vehicleType='" + vehicleType + '\'';
    }
}
```

2.3 Motorbike Class

```
package com.nujitha.project.classes;

import java.util.Objects;

public class Motorbike extends Vehicle {
    private boolean helmetGiven;
    private String bikeType;
    private String vehicleType;

    //constructor
    public Motorbike(String plateNo, String make, String model, int
odoMeter, boolean availability, boolean helmetGiven, String bikeType) {
        super(plateNo, make, model, odoMeter, availability);
        this.helmetGiven = helmetGiven;
        this.bikeType = bikeType;
    }

    //getters and setters
    @Override
    public String getVehicleType() {
        return vehicleType = "Motorbike";
    }

    public void setVehicleType(String vehicleType) {
        this.vehicleType = vehicleType;
    }

    public boolean isHelmetGiven() {
        return helmetGiven;
    }

    public void setHelmetGiven(boolean helmetGiven) {
        this.helmetGiven = helmetGiven;
    }

    public String getBikeType() {
        return bikeType;
    }

    public void setBikeType(String bikeType) {
        this.bikeType = bikeType;
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;
        Motorbike motorbike = (Motorbike) o;
        return helmetGiven == motorbike.helmetGiven &&
                Objects.equals(bikeType, motorbike.bikeType);
    }

    @Override
    public int hashCode() {
        return Objects.hash(super.hashCode(), helmetGiven, bikeType);
    }

    @Override
    public String toString() {
        return super.toString() + " helmetGiven=" + helmetGiven +
                ", bikeType='" + bikeType + '\'' +
                ", vehicleType='" + vehicleType + '\'';
    }
}
```

2.4 Schedule Class

```
package com.nujitha.project.classes;

import java.util.Date;

public class Schedule {
    //class attributes
    private String plateNo;
    private String pickUpDate;
    private String dropOffDate;
    private String customerName;
    private int contactNo;

    //constructor
    public Schedule() {}

    //getters and setters
    public String getCustomerName() {
        return customerName;
    }

    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }

    public int getContactNo() {
        return contactNo;
    }
```

```
    public void setContactNo(int contactNo) {
        this.contactNo = contactNo;
    }

    public String getPlateNo() {
        return plateNo;
    }

    public void setPlateNo(String plateNo) {
        this.plateNo = plateNo;
    }

    public String getPickUpDate() {
        return pickUpDate;
    }

    public void setPickUpDate(String pickUpDate) {
        this.pickUpDate = pickUpDate;
    }

    public String getDropOffDate() {
        return dropOffDate;
    }

    public void setDropOffDate(String dropOffDate) {
        this.dropOffDate = dropOffDate;
    }

    @Override
    public String toString() {
        return plateNo + "    " + pickUpDate + "    " + dropOffDate + "
" + customerName;
    }
}
```

2.5 RentalVehicleManager Interface

```
package com.nujitha.project.classes;

import java.io.IOException;
import java.util.concurrent.ExecutionException;

public interface RentalVehicleManager {
    void addVehicle() throws ExecutionException, InterruptedException,
IOException;
    void removeVehicle() throws ExecutionException, InterruptedException;
    void printList() throws ExecutionException, InterruptedException;
    void runMenu() throws ExecutionException, InterruptedException,
IOException;
    void save();
}
```

2.6 WestminsterRentalVehicleManager Class

```
package com.nujitha.project.classes;

import com.nujitha.project.firebaseUtils.FirebaseUtil;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.ExecutionException;


public class WestminsterRentalVehicleManager implements
RentalVehicleManager {
    private static final int maxLots = 50; //specifies the maximum
allowance in the vehicleCollection arrayList
    private static Scanner sc = new Scanner(System.in);
    private boolean availability = true; //default status of a vehicle if
not mentioned
    private List<Vehicle> vehicleCollection = new ArrayList<>(); //stores
all the vehicles store in the database
    private List<Schedule> bookedVehicles = new ArrayList<>();  //stores
all the details of bookings made
    private List<Object> bookingReferenceList = new ArrayList<>();
//reference numbers of the bookings currently made

    public WestminsterRentalVehicleManager() {
        try {
            FirebaseUtil.setupFirebase();   //setting-up database at the
beginning
            setVehicleCollection(FirebaseUtil.getToArray());    //retrieve
all vehicle information
            setBookedVehicles(FirebaseUtil.bookingObjects());   //retrieve
all bookings
            setBookingReferenceList(FirebaseUtil.bookingReferences());
//retrieve all booking reference numbers
            //booting-up the angular server
            File file = new
File("/personal/IIT/year02/OOP/coursework01/CustomerInterface/customer-
gui");
            Process pb = new ProcessBuilder("ng",
"serve").directory(file).start();
        } catch (ExecutionException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public List<Object> getBookingReferenceList() {
        return bookingReferenceList;
    }

    public void setBookingReferenceList(List bookingReferenceList) {
        this.bookingReferenceList = bookingReferenceList;
    }

    public List<Schedule> getBookedVehicles() {
        return bookedVehicles;
    }

    public void setBookedVehicles(List bookedVehicles) {
        this.bookedVehicles = bookedVehicles;
    }

    public List<Vehicle> getVehicleCollection() {
        return vehicleCollection;
    }

    public void setVehicleCollection(List<Vehicle> vehicleCollection) {
        this.vehicleCollection = vehicleCollection;
    }



    //method to add vehicle objects to the store



    @Override
    public void addVehicle() throws ExecutionException,
InterruptedException, IOException {
        int option; //preference on cases
        if (isCapacityAvailable()) {    //validating that the maximum lot
capacity allowance will not be exceeded
            do {
                System.out.println("Select your preference : " +
                        "\n 1) Add a Car" +
                        "\n 2) Add a Motorbike");
                isIntegerInput();
                option = sc.nextInt();
            } while (option > 2 || option < 1);

            //prompting for user inputs

            System.out.println("Please enter following information : ");
            System.out.println("Plate Number => ");
            String plateNo = sc.next();

            System.out.println("Make => ");
            String make = sc.next();

            System.out.println("Model => ");
            String model = sc.next();

            System.out.println("ODO meter(km) => ");
            isIntegerInput();
            int odo = sc.nextInt();
```

```
            switch (option) {
                case 1: //case of cars
                    System.out.println("Number of doors => ");
                    isIntegerInput();
                    int doors = sc.nextInt();

                    System.out.println("Is it hybrid (true/false) => ");
                    isBooleanInput();
                    boolean isHybrid = sc.nextBoolean();

                    //creating the object
                    Car car = new Car(plateNo, make, model, odo,
availability, doors, isHybrid);
                    //check on duplicate entries
                    checkDuplicates(car);

                    //adding details to the stores
                    vehicleCollection.add(car);
                    FirebaseUtil.addVehicle(car);

                    System.out.println("\nVehicle added
successfully..!!\n");

                    break;

                case 2: //case of motorbikes
                    System.out.println("Is the helmet given? (true/false)
=> ");
                    isBooleanInput();
                    boolean isHelmetGiven = sc.nextBoolean();

                    System.out.println("Type of bike (scooter/regular) =>
");
                    String bikeType = sc.next();

                    //creating the object
                    Motorbike motorbike = new Motorbike(plateNo, make,
model, odo, availability, isHelmetGiven, bikeType);
                    //check on duplicate entries
                    checkDuplicates(motorbike);

                    //adding details to the stores
                    vehicleCollection.add(motorbike);
                    FirebaseUtil.addVehicle(motorbike);

                    System.out.println("\nVehicle added
successfully..!!\n");

                    break;
            }
        } else {
            runMenu();
        }

    }
```

//method to remove data from the store

```
    @Override
    public void removeVehicle() throws ExecutionException,
InterruptedException {
        boolean existFlag = false;  //to confirm that the vehicle exitst
after iterating through the list
        Vehicle obj = null;
        System.out.println("Enter the Plate Number of the vehicle you want
to delete : ");  //prompting for plate number
        String removePlateNo = sc.next();
        //iterating through the list and finding the matching vehicle with
the plate number
        for (Vehicle currVehicle : vehicleCollection) {
            if (currVehicle.getPlateNo().equals(removePlateNo)) {
                obj = currVehicle;
                existFlag = true;
                break;
            }
        }
        //if vehicle exists
        if (existFlag) {
            //display details of the vehicle before deleting
            searchVehicle(removePlateNo);
            //removing vehicle from the store
            vehicleCollection.remove(obj);
            FirebaseUtil.deleteFirestore(removePlateNo);
            System.out.println("Vehicle removed successfully..!!");
        } else {
            System.out.println("Sorry, couldn't find a vehicle with that
plate number..!\n");
        }

        //notifies the remaining number of lots that can be added
        isCapacityAvailable();

    }
```

//method to print the list of all the vehicles in the store

```
    @Override
    public void printList() throws ExecutionException,
InterruptedException {
        int prefer; //viewing preference case by the user as in filtered
        do {
            System.out.println(" 1) View cars only" +
                    "\n 2) View Motorbikes only" +
                    "\n 3) View all");

            isIntegerInput();
            prefer = sc.nextInt();
        } while (prefer > 3 || prefer < 1); //validating input range
        Collections.sort(vehicleCollection);    //sorting accordingly to
```

```
the 'make' of the vehicle
        //do case of preference
        switch (prefer) {
            case 1: //case of viewing only cars
                System.out.println("PlateNo   Make\n_____________");
                for (Vehicle currVehicle : vehicleCollection) {
                    if (currVehicle.getVehicleType().equals("Car")) {
                        System.out.println(currVehicle.getPlateNo() + "
" + currVehicle.getMake());
                    }
                }
                System.out.println();
                break;

            case 2: //case of viewing only motorbikes
                System.out.println("PlateNo   Make\n_____________");
                for (Vehicle currVehicle : vehicleCollection) {
                    if (currVehicle.getVehicleType().equals("Motorbike"))
{
                        System.out.println(currVehicle.getPlateNo() + "
" + currVehicle.getMake());
                    }
                }
                System.out.println();
                break;

            case 3: //case of viewing all the vehicles regardless of any
filtering
                System.out.println("PlateNo   Make
Type\n_________________________");
                for (Vehicle currVehicle : vehicleCollection) {
                    System.out.println(currVehicle.getPlateNo() + "   " +
currVehicle.getMake() + "   " + currVehicle.getVehicleType());
                }
                System.out.println();
        }
    }



    //initial run menu



    @Override
    public void runMenu() throws ExecutionException, InterruptedException,
IOException {
        int choice; //choice selection from the menu
        do {
            System.out.println("\n________Welcome to Westminster Vehicle
Rental________\n" +
                    "\n 1) Add a vehicle" +
                    "\n 2) Delete a vehicle" +
                    "\n 3) View list of vehicles" +
                    "\n 4) View booked vehicles" +
                    "\n 5) Search a vehicle" +
                    "\n 6) Save" +
                    "\n 7) Delete booking" +
```

```
                    "\n 8) Open GUI" +
                    "\n 9) Exit the program" +
                    "\nEnter here : ");

            isIntegerInput();
            choice = sc.nextInt();
        } while (choice > 9 || choice < 1); //validating input ranges

        switch (choice) {
            case 1: //case of adding a vehicle
                addVehicle();
                runMenu();
            case 2: //case of deleting a vehicle
                removeVehicle();
                runMenu();
            case 3: //case of printing the vehicle
                printList();
                runMenu();
            case 4: //case of viewing all the booked details
                bookedVehicles();
                runMenu();
            case 5: //case of searching and viewing the particular vehicle
details
                System.out.println("Please enter plate number : ");
                String findThisPlateNo = sc.next();
                searchVehicle(findThisPlateNo);
                runMenu();
            case 6: //case of writing of the vehicle datils into a text
file
                save();
                runMenu();
            case 7: //case of deleting a booking
                removeBooking();
                runMenu();
            case 8: //case of opening the angular GUI
                openGUI();
                System.out.println("Web page opened in browser");
                runMenu();
            case 9: //case of ending the project application
                System.out.println("[[ Shutting down the servers ]]");
                System.out.println("\n ________Thank You________\n");
                System.exit(0);
                break;
            default:
                System.out.println("Please check again..!!");
        }
    }
```

//method to open the GUI in the browser

```
    private void openGUI() throws IOException {
        Runtime.getRuntime().exec(new String[]{"/usr/bin/open", "-a",
"/Applications/Google Chrome.app", "http://localhost:4200/"});
```

```
    }
```

//method to write the vehicle information into a txt file

```
    @Override
    public void save() {
        try {
            FileWriter fileWriter = new FileWriter("Vehicles.txt");
            //iterating through the vehicleCollection list and printing
each object in the file line by line
            for (Vehicle vehicle :
                    vehicleCollection) {
                fileWriter.write(vehicle.toString() + "\n");
            }
            System.out.println("\nSaved Successfully..!");
            //closing the file
            fileWriter.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

//method to view all the booking details

```
    private void bookedVehicles() throws ExecutionException,
InterruptedException {
        System.out.println("PlateNo   Pick-up       Drop-off
Customer_Name\n_____________________________________________");
        for (Schedule currVehicle : bookedVehicles) {
            System.out.println(currVehicle.toString());
        }
        System.out.println();

    }

    //method to remove a booking with the matching reference number
    public void removeBooking() throws ExecutionException,
InterruptedException {
        boolean hasBooking = false;     //validate if a matching booking
exists with the reference
        Vehicle vehicleObj = null;
        Schedule bookingObj = null;
        System.out.println("Enter the reference number of the vehicle you
want to delete : ");
        String removeRefNo = sc.next();
        System.out.println("Enter the Plate Number of the vehicle : ");
        String updatePlateNo = sc.next();
        //searching for the reference number
        for (Object currRef : bookingReferenceList) {
            if (currRef.equals(removeRefNo)) {
```

```
                hasBooking = true;
                break;
            }
        }
        //searching for the vehicle
        for (Vehicle currVehicle : vehicleCollection) {
            if (currVehicle.getPlateNo().equals(updatePlateNo)) {
                vehicleObj = currVehicle;
                break;
            }
        }
        for (Schedule currBooking : bookedVehicles) {
            if (currBooking.getPlateNo().equals(updatePlateNo)) {
                bookingObj = currBooking;
                break;
            }
        }
        if (hasBooking) {
            //update vehicle status
            int vehicleIndex = vehicleCollection.indexOf(vehicleObj);
            vehicleCollection.get(vehicleIndex).setAvailability(true);
            //remove booking from the stores
            bookedVehicles.remove(bookingObj);
            bookingReferenceList.remove(removeRefNo);
            FirebaseUtil.deleteBooking(removeRefNo, updatePlateNo);
            System.out.println("Booking removed successfully..!");
        } else {
            System.out.println("Sorry, couldn't find a reference
matching..!\n");
        }
    }
```

//method to search a particular vehicle

```
    public boolean searchVehicle(String findPlateNo) throws
ExecutionException, InterruptedException {
        boolean foundFlag = false;  //flag if the vehicle do exists
        for (Vehicle currVehicle : vehicleCollection) {
            if (currVehicle.getPlateNo().equals(findPlateNo)) {
                //printing vehicle object's details
                System.out.println("Plate number    -> " +
currVehicle.getPlateNo());
                System.out.println("Make            -> " +
currVehicle.getMake());
                System.out.println("Model           -> " +
currVehicle.getModel());
                System.out.println("ODO meter(km)   -> " +
currVehicle.getOdoMeter());
                System.out.println("Type            -> " +
currVehicle.getVehicleType());
                System.out.println();
                foundFlag = true;
                break;
            }
        }
        //in case if couldn't find the vehicle
```

```
        if (!foundFlag) {
            System.out.println("Sorry, the vehicle doesn't exist..!");
        }
        return foundFlag;
    }

    //to ensure that the store capacity will not be exceeded to the given
limit
    public boolean isCapacityAvailable() {
        int dbCount = vehicleCollection.size();
        if (dbCount >= maxLots) {
            System.out.println("Sorry, the lots are almost full..!!");
            return false;
        } else {
            System.out.println("\nYou can add " + (maxLots - dbCount) + "
more vehicles.\n");
            return true;
        }
    }

    //to check the arrayList for duplicate entries before adding it
    public boolean checkDuplicates(Vehicle obj) throws ExecutionException,
InterruptedException, IOException {
        boolean duplicateFlag = false;
        if (this.vehicleCollection.contains(obj)) {
            System.out.println("The vehicle already exists that you have
added earlier. Please add another.");
            duplicateFlag = true;
        }
        return duplicateFlag;
    }

    //to ensure the user input is an integer input, else re-prompt
    public void isIntegerInput() {
        while (!sc.hasNextInt()) {
            System.out.println("Wrong input, please enter a valid integer
: ");
            sc.next();
        }
    }

    //to ensure the user input is a boolean true/ false input, else re-
prompt
    public void isBooleanInput() {
        while (!sc.hasNextBoolean()) {
            System.out.println("Wrong input, please enter true / false :
");
            sc.next();
        }
    }
}
```

2.7 FirebaseUtil Class

```
package com.nujitha.project.firebaseUtils;

import com.google.api.core.ApiFuture;
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.firestore.*;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.google.firebase.cloud.FirestoreClient;
import com.nujitha.project.classes.Car;
import com.nujitha.project.classes.Motorbike;
import com.nujitha.project.classes.Schedule;
import com.nujitha.project.classes.Vehicle;
import org.springframework.stereotype.Service;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.ExecutionException;

@Service //serving to the API requests
public class FirebaseUtil {
    private static Firestore firestore; //instance of firestore

    //initializing firebase SDK
    public static void setupFirebase() throws IOException {
        InputStream serviceAccount = new FileInputStream("./year02oopcw01-
firebase-adminsdk-6nsyw-39361258d8.json");
        GoogleCredentials credentials =
GoogleCredentials.fromStream(serviceAccount);
        FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(credentials)
                .build();
        FirebaseApp.initializeApp(options);
        firestore = FirestoreClient.getFirestore();

    }



//Database CRUD operations


    //retrieving all vehicle objects
    public static List getToArray() throws ExecutionException,
InterruptedException {
        List<Vehicle> collection = new ArrayList<>();
        Query query = firestore.collection("vehicles");
        ApiFuture<QuerySnapshot> querySnapshot = query.get();

        //iterating through the the collection and adding to a list to
return
        for (DocumentSnapshot document :
querySnapshot.get().getDocuments()) {
            if
(Objects.requireNonNull(document.getString("vehicleType")).equalsIgnoreCas
e("Car")) {
```

```
                collection.add(document.toObject(Car.class)); //casting
and adding a car object
            } else {
                collection.add(document.toObject(Motorbike.class));
//casting and adding a motorbike object
            }
        }
        return collection;
    }

    //deleting a vehicle from the database
    public static void deleteFirestore(String pNum) throws
ExecutionException, InterruptedException {
        DocumentReference docRef =
firestore.collection("vehicles").document(String.valueOf(pNum));
        ApiFuture<DocumentSnapshot> future = docRef.get();
        DocumentSnapshot document = future.get();
        if (document.exists()) {

firestore.collection("vehicles").document(String.valueOf(pNum)).delete();
        } else {
            System.out.println();
        }
    }

    //adding a vehicle object
    public static void addVehicle(Vehicle uObject) {

firestore.collection("vehicles").document(String.valueOf(uObject.getPlateN
o())).set(uObject);
    }

    //retrieving only the reference numbers of bookings
    public static List bookingReferences() throws ExecutionException,
InterruptedException {
        List<Object> bookedVeh = new ArrayList<>();
        Query query = firestore.collection("bookings");
        ApiFuture<QuerySnapshot> querySnapshot = query.get();

        for (DocumentSnapshot document :
querySnapshot.get().getDocuments()) {
            bookedVeh.add(document.getId());
        }
        return bookedVeh;
    }

    //retrieving schedules
    public static List bookingObjects() throws ExecutionException,
InterruptedException {
        List<Object> objectList = new ArrayList<>();
        Query query = firestore.collection("bookings");
        ApiFuture<QuerySnapshot> querySnapshot = query.get();
        for (DocumentSnapshot document :
querySnapshot.get().getDocuments()) {
            objectList.add(document.toObject(Schedule.class));
        }
        return objectList;
    }
```

```
    //adding a schedule and updating vehicle's availability
    public static void addBooking(Schedule schedule) {
        firestore.collection("bookings").document().set(schedule);

firestore.collection("vehicles").document(schedule.getPlateNo()).update("a
vailability", false);
    }

    //deleting a schedule from the database and updating the vehicle
availability
    public static void deleteBooking(String refNo, String updatePlateNo)
throws ExecutionException, InterruptedException {
        DocumentReference docRef =
firestore.collection("bookings").document(String.valueOf(refNo));
        ApiFuture<DocumentSnapshot> future = docRef.get();
        DocumentSnapshot document = future.get();
        if (document.exists()) {

firestore.collection("bookings").document(String.valueOf(refNo)).delete();

firestore.collection("vehicles").document(updatePlateNo).update("availabil
ity", true);
        } else {
            System.out.println();
        }
    }

}
```

2.8 ApiController

```
package com.nujitha.project.controllers;

import com.nujitha.project.classes.Schedule;
import com.nujitha.project.firebaseUtils.FirebaseUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.concurrent.ExecutionException;

@CrossOrigin(origins = "*", allowedHeaders = "*") //cross origin resource
sharing
@RestController
@RequestMapping("/api")
public class ApiController {

    private FirebaseUtil firebaseUtil;

    @Autowired // injecting object dependencies
    public ApiController(FirebaseUtil firebaseUtil) {
        super();
        this.firebaseUtil = firebaseUtil;
    }
```

```
    @GetMapping("/vehicles")
    public List getToArray() throws ExecutionException,
InterruptedException {
        return FirebaseUtil.getToArray();
    }

    @PostMapping("/booking")
    public void createBooking(@RequestBody Schedule schedule) {
        FirebaseUtil.addBooking(schedule);
    }

    @GetMapping("/bookinglist")
    public List bookedCollection() throws ExecutionException,
InterruptedException {
        return FirebaseUtil.bookingObjects();
    }
}
```

3 Graphical User Interface

3.1 Visualizing the vehicles and filtering view

vehicle-collection.component.html

```
<!--navigation pane-->
<ul class="nav nav-pills" id="myTab" role="tablist">
  <!--  all-->
  <li class="nav-item">
    <a aria-controls="all" aria-selected="true" class="nav-link active"
data-toggle="tab" href="#all" id="all-tab"
       role="tab">All</a>
  </li>
  <!--  available-->
  <li class="nav-item">
    <a aria-controls="available" aria-selected="false" class="nav-link"
data-toggle="tab" href="#available"
       id="available-tab"
       role="tab">Available</a>
  </li>
  <!--  cars-->
  <li class="nav-item">
    <a aria-controls="car" aria-selected="false" class="nav-link" data-
toggle="tab" href="#car" id="car-tab" role="tab">Cars</a>
  </li>
  <!--  motorbikes-->
  <li class="nav-item">
    <a aria-controls="motorbike" aria-selected="false" class="nav-link"
data-toggle="tab" href="#motorbike"
       id="motorbike-tab"
       role="tab">Motorbikes</a>
  </li>
</ul>
<!--end of navigation pane-->
<!--table view-->
<div class="tab-content" id="myTabContent">
  <!--  all vehicles tab-->
  <div aria-labelledby="car-tab" class="tab-pane fade show active"
id="all" role="tabpanel">
    <table class="table table-hover">
      <thead style="background-color: gray">
      <tr>
        <th>Plate Number</th>
        <th>Vehicle Type</th>
        <th>Make</th>
        <th>Model</th>
      </tr>
      </thead>
      <tbody>
      <tr *ngFor="let veh of vehicles">
        <td (click)="onSelect(veh)">{{veh.plateNo}}</td>
        <td>{{veh.vehicleType}}</td>
        <td>{{veh.make}}</td>
        <td>{{veh.model}}</td>
      </tr>
```

```
      </tbody>
    </table>
  </div>
  <!--  available vehicles tab-->
  <div aria-labelledby="available-tab" class="tab-pane fade"
id="available" role="tabpanel">
    <table class="table table-hover">
      <thead style="background-color: gray">
      <tr>
        <th>Plate Number</th>
        <th>Vehicle Type</th>
        <th>Make</th>
        <th>Model</th>
      </tr>
      </thead>
      <tbody>
      <tr *ngFor="let veh of vehicles">
        <td (click)="onSelect(veh)"
*ngIf="veh.availability==true">{{veh.plateNo}}</td>
        <td *ngIf="veh.availability==true">{{veh.vehicleType}}</td>
        <td *ngIf="veh.availability==true">{{veh.make}}</td>
        <td *ngIf="veh.availability==true">{{veh.model}}</td>
      </tr>
      </tbody>
    </table>
  </div>
  <!--  cars only tab-->
  <div aria-labelledby="profile-tab" class="tab-pane fade" id="car"
role="tabpanel">
    <table class="table table-hover">
      <thead style="background-color: gray">
      <tr>
        <th>Plate Number</th>
        <th>Vehicle Type</th>
        <th>Make</th>
        <th>Model</th>
      </tr>
      </thead>
      <tbody>
      <tr *ngFor="let veh of vehicles">
        <td (click)="onSelect(veh)"
*ngIf="veh.vehicleType=='Car'">{{veh.plateNo}}</td>
        <td *ngIf="veh.vehicleType=='Car'">{{veh.vehicleType}}</td>
        <td *ngIf="veh.vehicleType=='Car'">{{veh.make}}</td>
        <td *ngIf="veh.vehicleType=='Car'">{{veh.model}}</td>
      </tr>
      </tbody>
    </table>
  </div>
  <!--  motorbikes only tab-->
  <div aria-labelledby="contact-tab" class="tab-pane fade" id="motorbike"
role="tabpanel">
    <table class="table table-hover">
      <thead style="background-color: gray">
      <tr>
        <th>Plate Number</th>
        <th>Vehicle Type</th>
        <th>Make</th>
        <th>Model</th>
```

```
      </tr>
      </thead>
      <tbody>
      <tr *ngFor="let veh of vehicles">
        <td (click)="onSelect(veh)"
*ngIf="veh.vehicleType=='Motorbike'">{{veh.plateNo}}</td>
        <td *ngIf="veh.vehicleType=='Motorbike'">{{veh.vehicleType}}</td>
        <td *ngIf="veh.vehicleType=='Motorbike'">{{veh.make}}</td>
        <td *ngIf="veh.vehicleType=='Motorbike'">{{veh.model}}</td>
      </tr>
      </tbody>
    </table>
  </div>
</div>
```

vehicle-collection.component.ts

```
import {Component, OnInit} from '@angular/core';
import {VehicleService} from '../../shared/vehicle.service';
import {Vehicle} from '../../shared/vehicle.model';
import {Booking} from '../../shared/booking.model';

@Component({
  selector: 'app-vehicle-collection',
  templateUrl: './vehicle-collection.component.html',
  styleUrls: ['./vehicle-collection.component.css']
})
export class VehicleCollectionComponent implements OnInit {
  vehicles: Vehicle[]; // list to store retrieved vehicle objects

  // constructor
  constructor(private  service: VehicleService) {
  }

  ngOnInit() {
    this.service.getVehicle().subscribe(
      response => {
        this.vehicles = response;
      });
  }

  // sets the plate number in the input field when clicked on the table's
plate number field
  onSelect(booking: Booking) {
    this.service.formData = Object.assign({}, booking);
  }

}
```

3.2 Check availability of a vehicle and make a booking

vehicle.component.html

```
<h4>Search Vehicle</h4>
<!--form to search a vehicle's availability-->
<form #form1="ngForm" autocomplete="off">
  <div>
    <!--    plate number input field -->
    <div class="form-group">
      <input #plateNo="ngModel" [(ngModel)]="service.formData.plateNo"
class="form-control" name="plateNo"
             placeholder="Plate Number" required="required">
    </div>
    <div *ngIf="plateNo.touched && plateNo.invalid" class="alert alert-
danger">
      required
    </div>
    <!--    date input fields-->
    <div class="form-row">
      <div class="form-group col-md-6">
        <input #sDate="ngModel" [(ngModel)]="service.formData.pickUpDate"
class="form-control" name="sDate"
               onfocus="(this.type='date')"
               placeholder="Pick-up Date" required="required" type="text">
      </div>
      <div *ngIf="sDate.touched && sDate.invalid" class="alert alert-
danger">
        required
      </div>
      <div class="form-group col-md-6">
        <input #eDate="ngModel" [(ngModel)]="service.formData.dropOffDate"
class="form-control" name="eDate"
               onfocus="(this.type='date')"
               placeholder="Drop-off Date" required="required"
type="text">
      </div>
      <div *ngIf="eDate.touched && eDate.invalid" class="alert alert-
danger">
        required
      </div>
    </div>
    <!--    check availability button-->
    <div class="form-group">
      <button (click)="checkAvailability()" [disabled]="form1.invalid"
class="btn btn-info btn-block" type="submit">
        Check Availability
      </button>
    </div>
    <!--    display summary of the selected vehicle-->
    <table class="table borderless">
      <tbody>
      <tr>
        <td>{{vehicleType}}</td>
      </tr>
      <tr>
        <td>{{make}}</td>
```

```
      </tr>
      <tr>
        <td>{{model}}</td>
      </tr>
      <tr>
        <td>{{odo}}</td>
      </tr>
      <!--      toggle switch-->
      <tr>
        <div class="custom-control custom-switch">
          <input class="custom-control-input" id="customSwitches"
type="checkbox">
          <label class="custom-control-label" for="customSwitches">Toggle
this switch if you need the helmet</label>
        </div>
      </tr>
      </tbody>
    </table>
  </div>
</form>
<!--form to enter customer details-->
<form #form2="ngForm" autocomplete="off">
  <div class="form-group">
    <!--    enter customer name field-->
    <input #customerName="ngModel"
[(ngModel)]="service.formData.customerName" class="form-control"
name="customerName"
           placeholder="Your Name" required="required">
  </div>
  <div *ngIf="customerName.touched && customerName.invalid" class="alert
alert-danger">
    name is required
  </div>
  <!--  enter customer contact number field-->
  <div class="form-group">
    <input #contactNo="ngModel" [(ngModel)]="service.formData.contactNo"
class="form-control" maxlength="10"
           name="contactNo" placeholder="Contact Number"
required="required">
  </div>
  <div *ngIf="contactNo.touched && customerName.invalid" class="alert
alert-danger">
    required and should be valid
  </div>
  <!--  confirm booking button-->
  <button (click)="confirmBooking()" [disabled]="form1.invalid ||
form2.invalid" class="btn btn-info btn-block"
          type="submit">Confirm Booking
  </button>
</form>
```

vehicle.component.ts

```
import {Component, OnInit} from '@angular/core';
import {VehicleService} from '../../shared/vehicle.service';
```

```
import {NgForm} from '@angular/forms';
import {Vehicle} from '../../shared/vehicle.model';
import {Booking} from '../../shared/booking.model';
import {ToastrService} from 'ngx-toastr';

@Component({
  selector: 'app-vehicle',
  templateUrl: './vehicle.component.html',
  styleUrls: ['./vehicle.component.css']
})
export class VehicleComponent implements OnInit {

  booking: Booking;
  vehicleType: string;
  make: string;
  model: string;
  odo: string;
  vehicles: Vehicle[];
  bookingList: Booking[];
  private vehicleFoundFlag: boolean;
  private vehicleAvailableFlag: boolean;
  private maxDate: number;
  private minDate: number;
  private formPickDate: number;
  private formDropDate: number;

  // constructor
  constructor(private service: VehicleService,
              private toastr: ToastrService) {
    this.booking = new Booking();
  }

  ngOnInit() {
    this.resetForm();
    // get all the vehicles into vehicle array
    this.service.getVehicle().subscribe(
      response => {
        this.vehicles = response;
      });

    // get all the schedules into bookings array
    this.service.getBookings().subscribe(
      data => {
        this.bookingList = data;
      }
    );
  }

  // method to reset form on start
  resetForm(form?: NgForm) {
    if (form != null) {
      form.resetForm();
    }
    this.service.formData = {
      plateNo: null,
      pickUpDate: null,
      dropOffDate: null
    };
  }
```

```
  // method which will be invoked when 'confirm booking' button is pressed
  confirmBooking() {
    this.service.doBooking(this.service.formData).subscribe(result =>
this.booking = result);
    this.toastr.success('Booking recorded successfully', 'Vehicle
Booking');
    setTimeout(() => {
      window.location.reload();
    }, 2000);
  }

  // method which will be invoked when 'check availability' button is
clicked
  checkAvailability() {
    this.vehicleAvailableFlag = false;
    this.vehicleFoundFlag = false;
    this.bookingList.forEach(dateObject => {
      if (this.service.formData.plateNo === dateObject.plateNo) {
        // casting into date formats
        this.minDate = Date.parse(dateObject.pickUpDate);
        this.maxDate = Date.parse(dateObject.dropOffDate);
        this.formPickDate = Date.parse(this.service.formData.pickUpDate);
        this.formDropDate = Date.parse(this.service.formData.dropOffDate);
      }
    });
    // checking if the vehicle exists and if available for the form given
dates
    this.vehicles.forEach((obj) => {
      if (this.service.formData.plateNo === obj.plateNo) {
        this.vehicleFoundFlag = true;
        if (!(this.formPickDate > this.minDate && this.formPickDate <
this.maxDate)) {
          if (!(this.formDropDate > this.minDate && this.formDropDate <
this.maxDate)) {
            this.toastr.info('Vehicle is available for booking',
'Available');  // pop-up toastr notification
            this.vehicleAvailableFlag = true;
            this.vehicleType = 'Vehicle Type         -> ' +
obj.vehicleType;
            this.make = 'Make         -> ' + obj.make;
            this.model = 'Model        -> ' + obj.model;
            this.odo = 'odo          -> ' + obj.odoMeter;
          }
        }
      }
    });
    // if there is a booking within form required dates
    if (this.vehicleFoundFlag && !this.vehicleAvailableFlag) {
      this.toastr.warning('This vehicle is booked on these dates',
'Unavailable'); // pop-up toastr notification
    }
    // if vehicle does not exist
    if (!this.vehicleFoundFlag) {
      this.toastr.error('Entered plate number is invalid', 'Invalid'); //
pop-up toastr notification
    }
  }
```

```
}
```

3.3 Service class (vehicle.service.ts)

```
import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {Vehicle} from './vehicle.model';
import {Booking} from './booking.model';

@Injectable({
  providedIn: 'root'
})
export class VehicleService {
  formData: Booking; // extracts the form input data
  // constructor
  constructor(private http: HttpClient) {}
  // method to get the vehicle list from the backend
  getVehicle() {
    return this.http.get<Vehicle[]>('http://localhost:8080/api/vehicles');
  }
  // method to send booking details to the backend
  doBooking(formData) {
    return this.http.post<Booking>('http://localhost:8080/api/booking',
this.formData);
  }
  // method to get bookings list from backend
  getBookings() {
    return
this.http.get<Booking[]>('http://localhost:8080/api/bookinglist');
  }
}
```

3.4 Screenshots

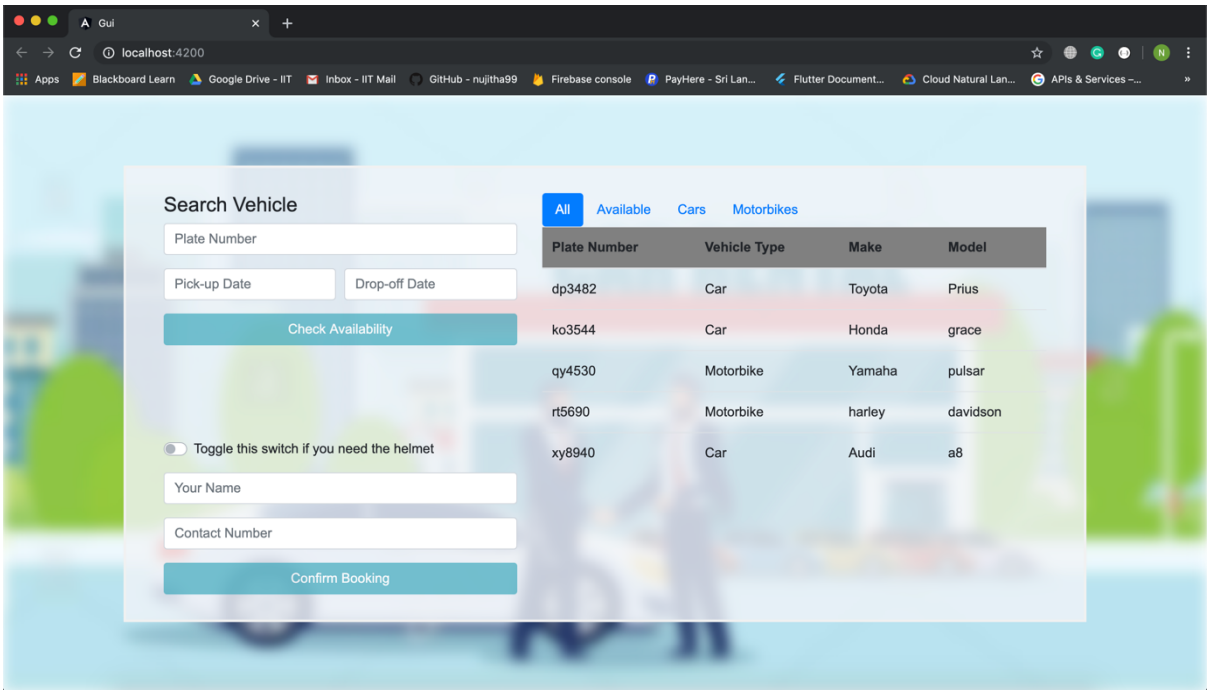


Figure 1 initial view of the GUI

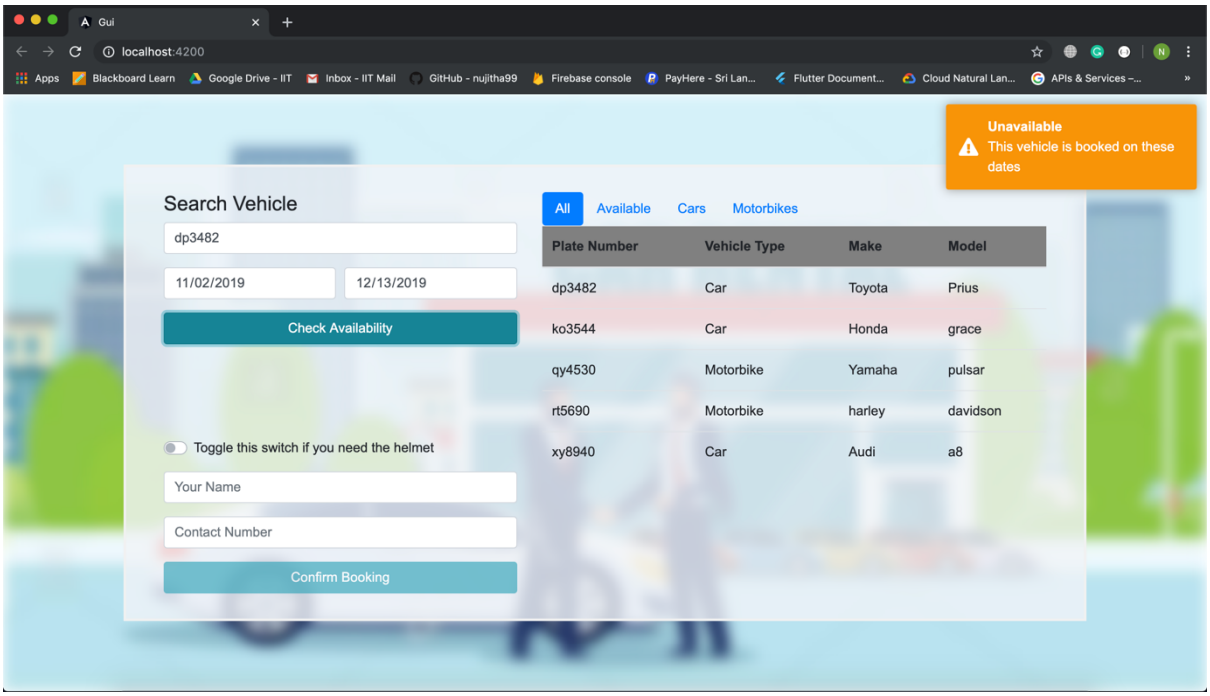


Figure 2 notification will be displayed if the vehicle is not available within the selected dates

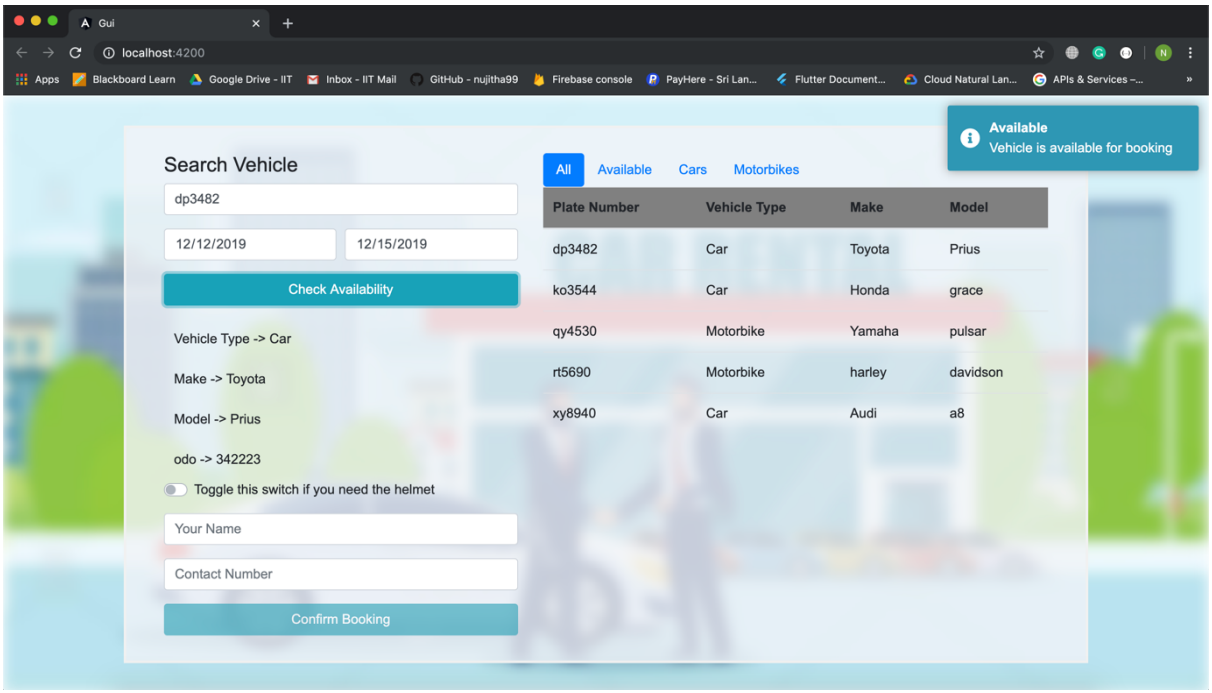


Figure 3 A notification will be prompted if the vehicle is available for booking the selected dates

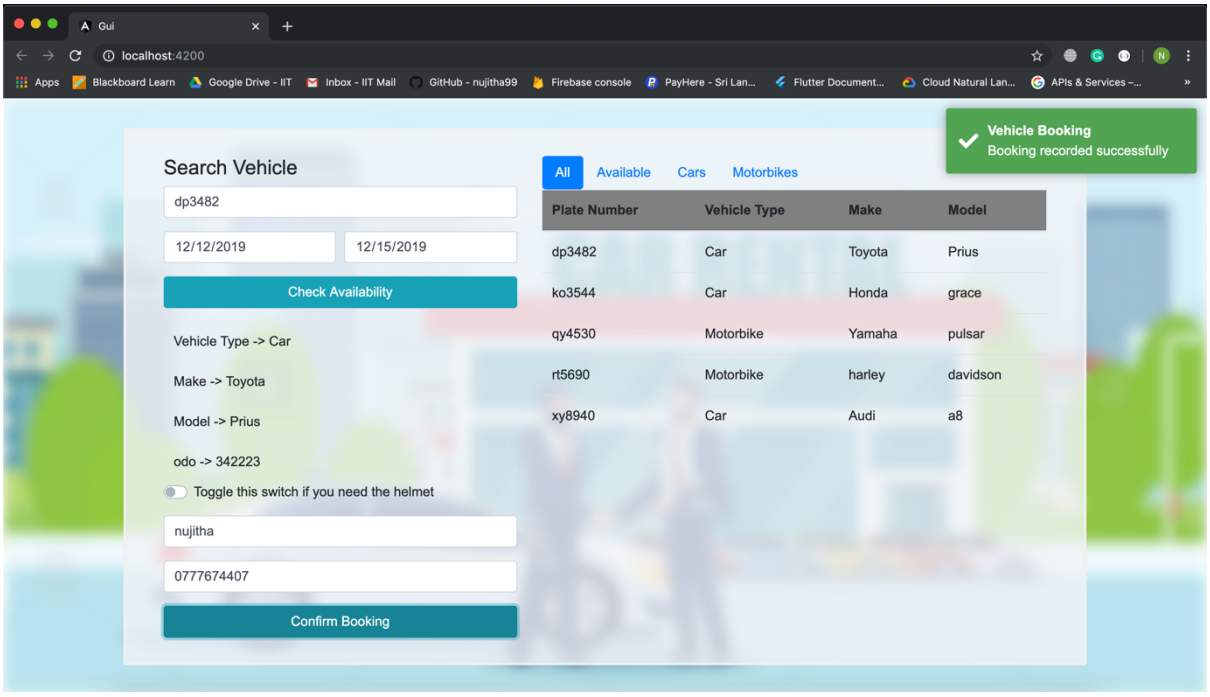


Figure 4 Booking confirmation notification prompted after confirming the booking

4. Testing

4.1Test Plan

#	Condition / Description	Input	Expected Output	Actual output	Result
1	Validating that two duplicate vehicle objects will not be allowed to entered by the user Invoked method -> checkDuplicates() Conditions -> return true if detected any	Inserted two duplicate Car objects (same plate number) and invoked checkDuplicates() method	True	True	Passed
2	Checking for invalid plate number inputs in search option Invoked method -> searchVehicle() Conditions -> return false if no match found	Inserted an invalid plate number - searchVehicle ("invalidPlateNumber")	False	False	Passed
3	Confirming that bookings will be removed from all the lists when a booking is removed by the manger Invoked method -> removeBooking() Condition1 -> return false if couldn't find the Schedule object after deleting Condition 2 -> return true if vehicle status updated as true (available back)	Valid plate number -> Dp3482 Valid reference number -> `Pezj0UnL3DuGYh3mHRCM`	False True	False True	Passed Passed

4	Condition3 -> return false if couldn't find the reference number after deleting		False	False	Passed
5	Verifying that the that the lot capacity will not be exceeded Invoked method -> addVehicle() Condition1 -> return false if store has reached the maximum capacity of 50	51 different Car objects in a loop	False	False	Passed
6	Condition2 -> return true if lots available	25 different Car objects in a loop	True	True	Passed

4.2 WestminsterRentalVehicleManagerTest

```
package com.nujitha.project;

import com.nujitha.project.classes.Car;
import com.nujitha.project.classes.Schedule;
import com.nujitha.project.classes.Vehicle;
import com.nujitha.project.classes.WestminsterRentalVehicleManager;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.List;
import java.util.concurrent.ExecutionException;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest
class ProjectApplicationTests {

    private static WestminsterRentalVehicleManager
westminsterRentalVehicleManager = new WestminsterRentalVehicleManager();

    @Test
    void contextLoads() {
    }
```

```
    @Test
        //test method to verify that duplicate entries will not be entered
    void checkDuplicateVehicles() throws InterruptedException,
ExecutionException, IOException {
        List<Vehicle> list =
westminsterRentalVehicleManager.getVehicleCollection();
        Car car1 = new Car("plateNo", "make", "model", 123, true, 4,
true);
        Car car2 = new Car("plateNo", "make", "model", 123, true, 4,
true);
        list.add(car1);
        list.add(car2);
        assertTrue(westminsterRentalVehicleManager.checkDuplicates(car2));
    }

    @Test
        //check for invalid plate numbers
    void searchInvalidVehicle() throws ExecutionException,
InterruptedException {

assertFalse(westminsterRentalVehicleManager.searchVehicle("invalidPlateNum
ber"));
    }

    @Test
        //check if data will be successfully removed from all the lists
    void removeBooking() {
        List<Vehicle> vehicles =
westminsterRentalVehicleManager.getVehicleCollection();
        List<Schedule> bookigns =
westminsterRentalVehicleManager.getBookedVehicles();
        List refs =
westminsterRentalVehicleManager.getBookingReferenceList();

        String thisRefNo = "Pezj0UnL3DuGYh3mHRCM";
        String thisPlateNo = "ko3544";

        Vehicle vehicleObj = null;
        for (Vehicle currVehicle : vehicles) {
            if (currVehicle.getPlateNo().equals(thisPlateNo)) {
                vehicleObj = currVehicle;
                break;
            }
        }
        Schedule bookingObj = null;
        for (Schedule currBooking : bookigns) {
            if (currBooking.getPlateNo().equals(thisPlateNo)) {
                bookingObj = currBooking;
                break;
            }
        }
        int vehicleIndex = vehicles.indexOf(vehicleObj);
        vehicles.get(vehicleIndex).setAvailability(true);
        bookigns.remove(bookingObj);
        refs.remove(thisRefNo);

        boolean finalStatusUpdated = false;
        boolean finalFalseBooking = true;
        boolean finalHasRef = false;
```

```
        for (Vehicle currVehicle : vehicles) {
            if (currVehicle.getPlateNo().equals(thisPlateNo)) {
                if (!currVehicle.isAvailability()) {
                    finalStatusUpdated = true;
                    break;
                }
            }
        }
        for (Schedule currBooking : bookigns) {
            if (currBooking.getPlateNo().equals(thisPlateNo)) {
                finalFalseBooking = false;
                break;
            }
        }
        for (Object currRef : refs) {
            if (currRef.equals(thisRefNo)) {
                finalHasRef = true;
                break;
            }
        }

        assertFalse(finalStatusUpdated);
        assertTrue(finalFalseBooking);
        assertFalse(finalHasRef);


    }

    @Nested
            //test method that verifies capacity limit
    class capacityCheck {

        @Test
        void capacityFull() {
            List<Vehicle> cap1 =
westminsterRentalVehicleManager.getVehicleCollection();
            for (int i = 0; i < 51; i++) {
                cap1.add(new Car("plateNo" + i, "make", "model", 123,
true, 4, true));
            }

assertFalse(westminsterRentalVehicleManager.isCapacityAvailable());
            cap1.clear();
        }

        @Test
        void capacityAvailable() {
            List<Vehicle> cap2 =
westminsterRentalVehicleManager.getVehicleCollection();
            for (int i = 0; i < 25; i++) {
                cap2.add(new Car("plateNo" + i, "make", "model", 123,
true, 4, true));
            }

assertTrue(westminsterRentalVehicleManager.isCapacityAvailable());
        }

    }
}
```

4.3 Screenshots

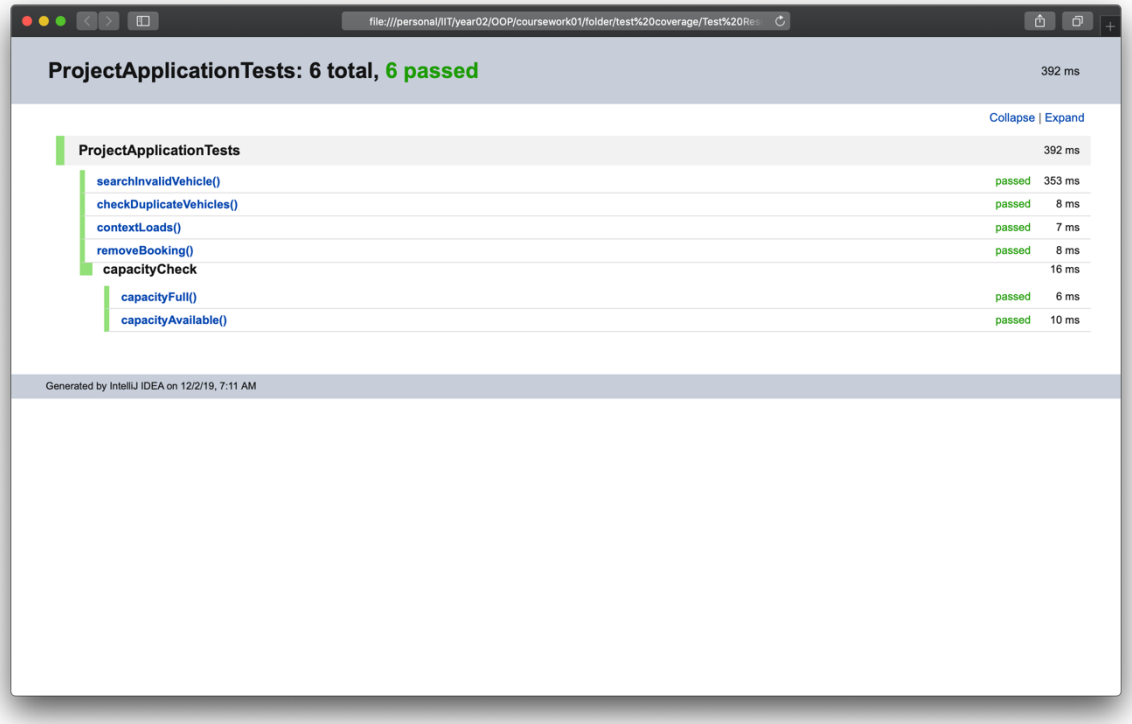


Figure 5 Report generated by IntelliJ IDEA for the test results

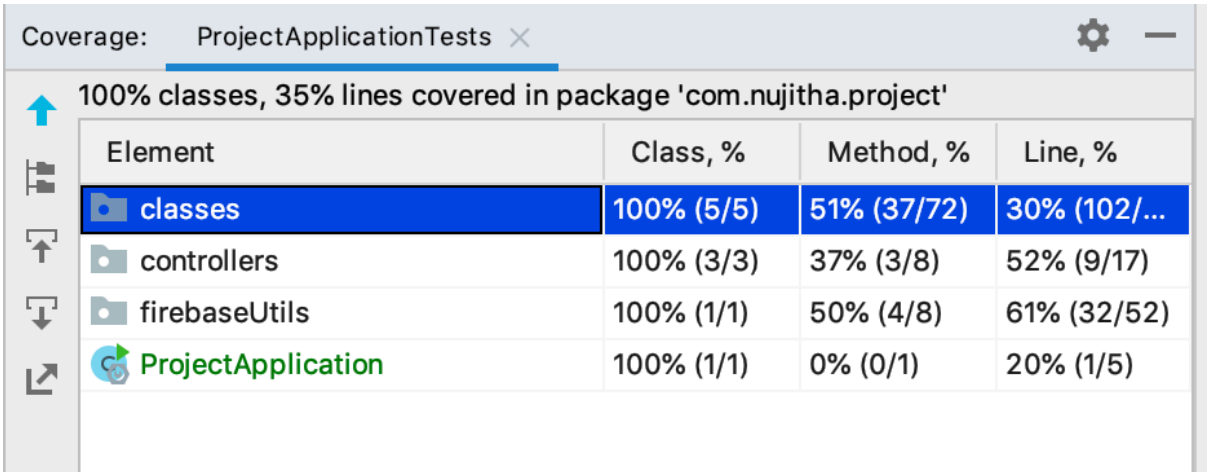


Figure 6 Coverage results generated by the IDE

5. Database view

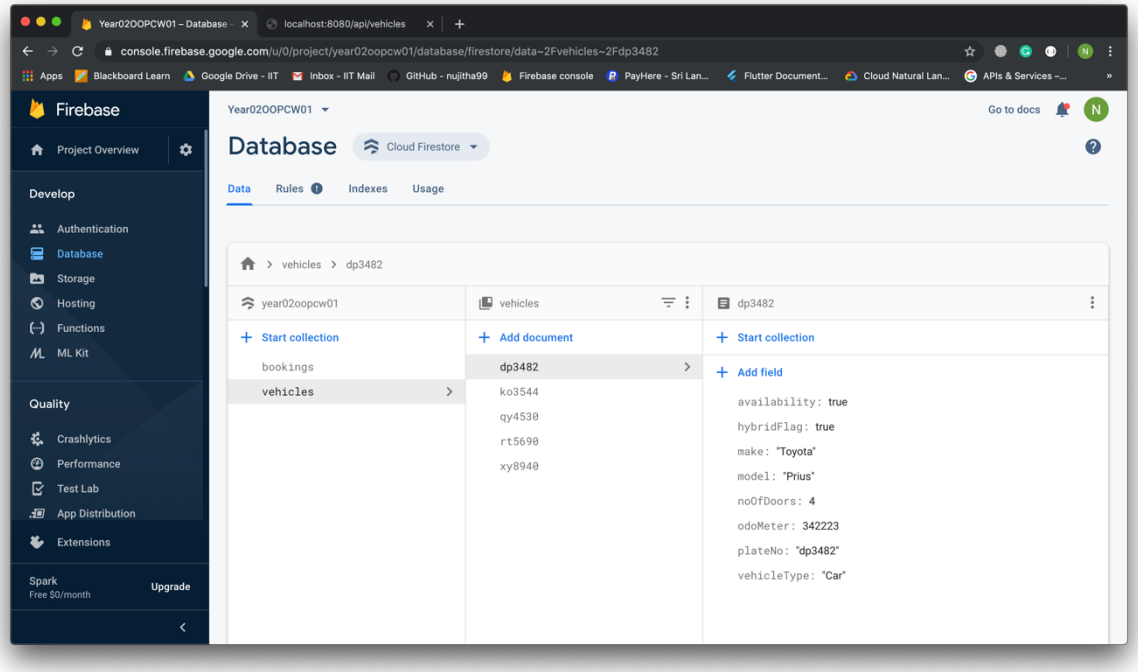


Figure 7 Vehicles stored in the database

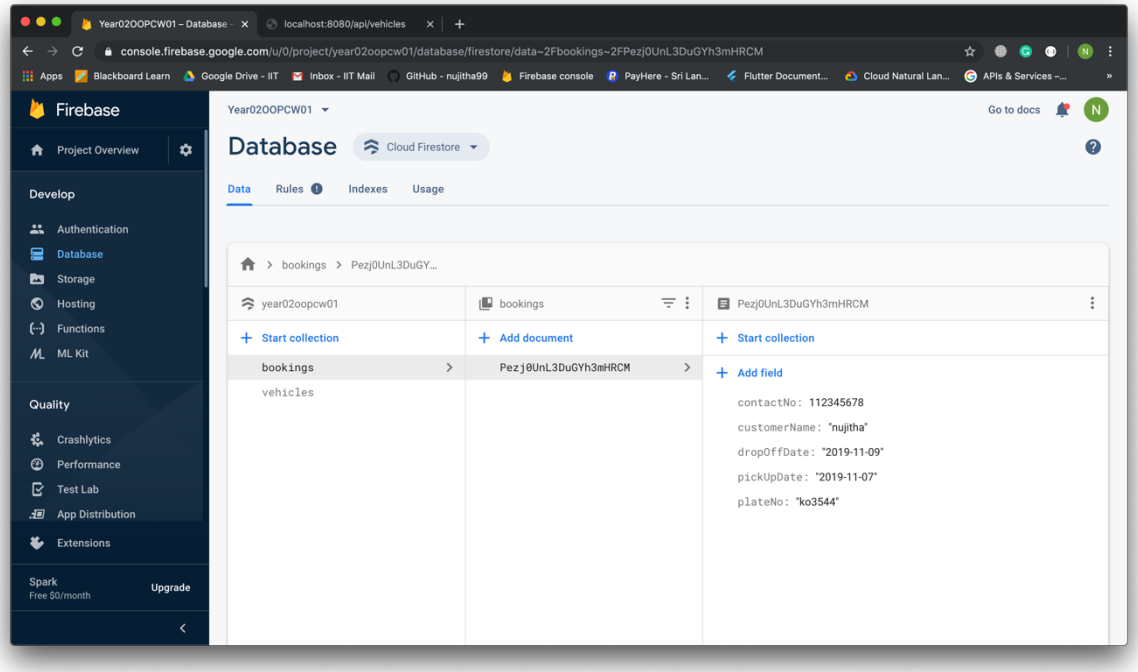


Figure 8 Bookings made with the reference numbers