**­­** Car Parking Management System

By

Group 2

Japneet Singh - 300323547

Jagjit Singh – 300316857

Kosha Raval - 300324116

Vinay Kolli - 300316562

**INTRODUCTION**

We have developed a car parking system to override the problems that persists in practicing manual system. Our application is developed to eliminate the problems that are faced by the existing car parking systems. Moreover, this system is designed to make the life of the residents easy and hassle-free, thereby giving them a tool that makes their parking experience smooth and better than ever.

The application is made as much as minimalistic as possible to avoid complexities and errors while entering the data. No prior knowledge needed for the user to use this system as it is user-friendly, easy to navigate and use by any person of any age group.

Thus, by this all it provides a sense of security and reliability that leads to faster management of the system. It can therefore assist the user to focus on their activities and not worry about their vehicles.

Every organization, whether big or small, has challenges to overcome and managing the information of parking car, parking slots, car number, parking fees, etc. Every online car parking system has different car needs; therefore, we designed this system that are adapted to the user's needs. This is designed to assist in strategic planning and will help you ensure that you are equipped with the right level of information.

Finally, our system will ultimately allow to better manage your time by automatically allocating the parking slots based on the size of the car that you choose, thereby creating an organized and well implemented system that not only makes your life easy but help to make it provide a better user experience altogether.

**Major Technologies Used**

Spring Boot, React, and MongoDB.

**How to run the application**

* Run the project in IntelliJ.
* Make sure Node.js is installed on the pc.
* Open src\main\frontend in the Command Prompt.
* Inside this directory, run npm install to install the React dependencies.
* Run npm start.
* Application will start at localhost:3000.

**Login info for the application**

* Username: admin | Password: admin123
* Username: admin2 | Password: admin456

GIT HUB Project Link : [Group2\_CarParkingSystem](https://github.com/Jagjit-Douglas/Car-Parking-System.git)

**SYSTEM DESIGN AND ANALYSIS**

**Defining components of the system**

* We have defined 4 components in this system:
  + Admin
  + Car
  + Owner
  + Parking Lot
* This car parking system has only one user called the Admin. There could be multiple Admins with their unique pair of user-id and password. Admin will have all the privileges in the system. Admin will login into the system using the login page.
* When a new car arrives to be parked the Admin will add the car into system. He will use Park a Car form page to input information about the car and owner. Car inputs are Car number, Car size i.e., small, medium and large and Owner inputs are First Name, Last Name and Phone number.
* In addition to the information added by the Admin, the Car component has 4 dynamic fields: cars entry time, cars exit time, parking space number and a parking fee.
* The entry time of the car and parking lot are two which are inserted into the database when the car is parked. The entry time is the current time stamp of the device on which it will be deployed to. The parking lot for a car is allocated based on the nearest available parking lot based on the size of the car.
* However, the exit time of the car and the parking fee fields are added when the car exits the parking lot. The exit time current time stamp of the device from which the car is requested to exit. The parking fee is calculated based on the time it was parked for and size of the car.
* Parking lot component is of 3 sizes similar to the car. There are 70 parking lots available.
* The Owner Component has owner first name, last name and phone will be noted by the admin when he registers a new car into the system.

**Description of the system architecture**

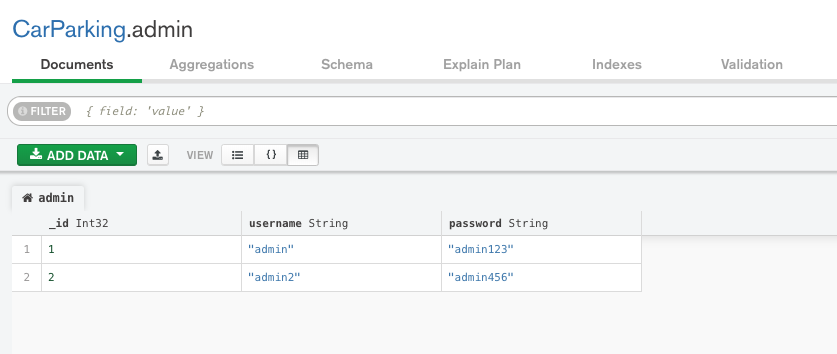
* We have used Multi-layered architecture for developing car-parking system which mainly consist of 3 layers.
* **The web layer:**
  + It is the uppermost layer of a web application. It is responsible of processing user’s input and returning the correct response back to the user. The web layer must also handle the exceptions thrown by the other layers. Because the web layer is the entry point of our application, it must take care of authentication and act as a first line of defense against unauthorized users.
  + In this project we have used React for creating the UI screens of the system and Spring Boot with java as the language to create the controllers of the system.
* **The service layer** resides below the web layer. It acts as a transaction boundary and contains both application and infrastructure services. The **application services** provide the public API of the service layer. They also act as a transaction boundary and are responsible of authorization. The **infrastructure services** contain code that communicates with external resources such as database in our case is MongoDB.
* **The repository layer** is the lowest layer of a web application. It is responsible of communicating with the used data storage.

**Development of Use Case diagram**

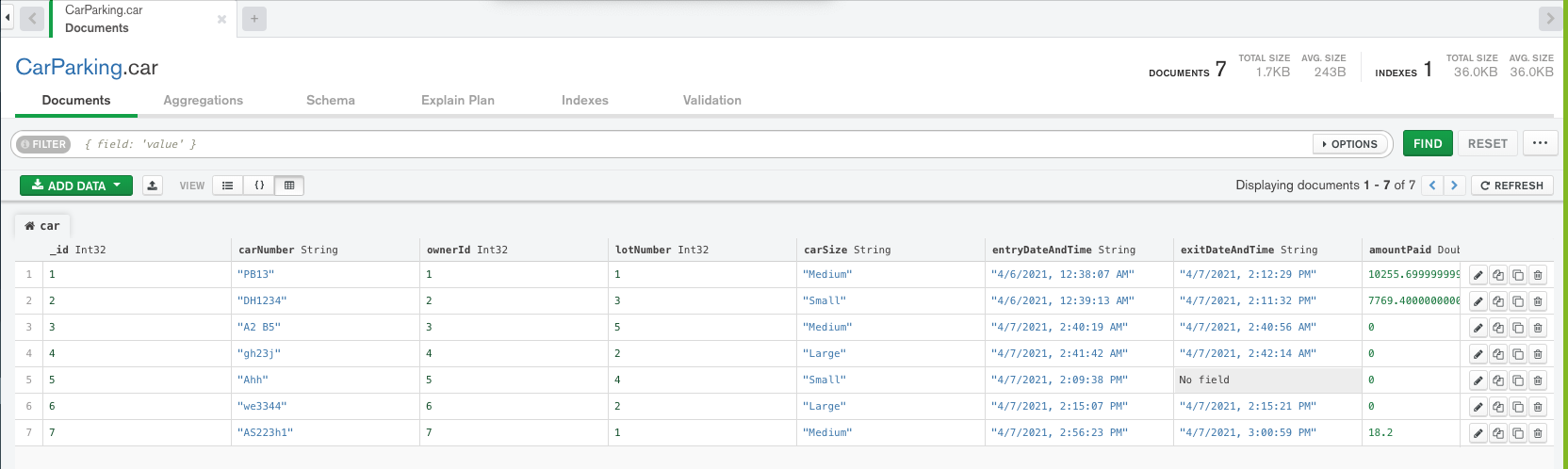
|  |  |
| --- | --- |
| Name | Add a Car |
| Participating Actors | Admin |
| Goals | Insert Car information into the system |
| Triggers | New car added to the database |
| Pre-Conditions | Open the car form and entering the details |
| Post-Conditions | Viewing the car information that is stored inside database |
| Basic-Flow | Login into the system, Open Car form |
| Alternate Flow | NA |
| Exceptions | All input fields are mandatory |
| Qualities | * Allocation of Parking space * Populating parking entry time of the car |

|  |  |
| --- | --- |
| Name | Exiting a Car |
| Participating Actors | Admin, Owner |
| Goals | Calculate the parking fees based on time constraint. |
| Triggers | Owner request to exit the car |
| Pre-Conditions | Select the required Car |
| Post-Conditions | Car Summary |
| Basic-Flow | Select car based on the car number from Show Car page |
| Alternate Flow | NA |
| Exceptions | NA |
| Qualities | * Populating parking exit time of the car * Total parking fee to be paid |

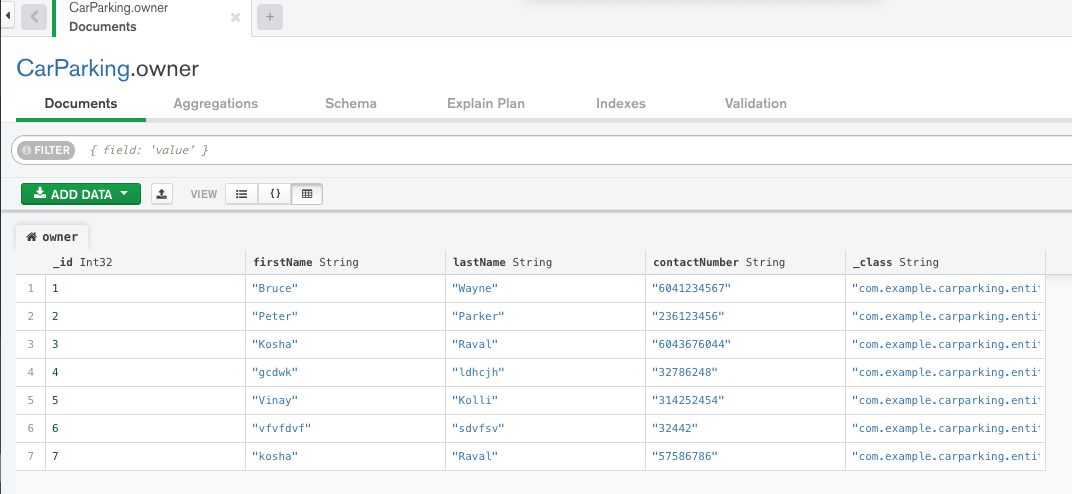
**Defining the database and schemas**

Admin Table:

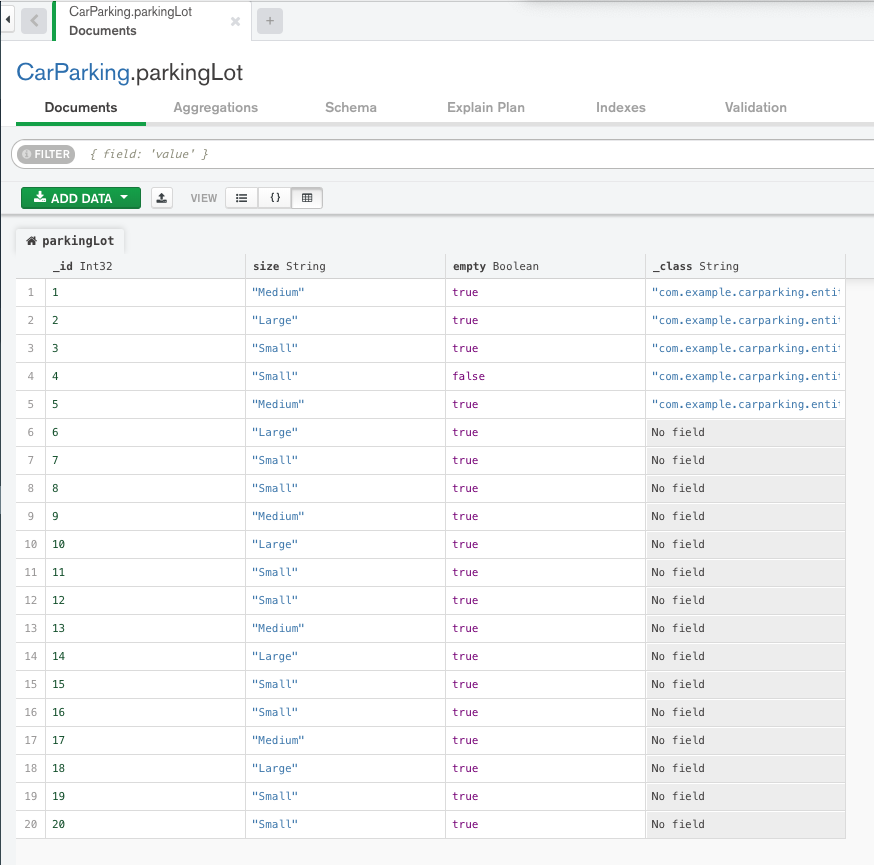
Car Table:



Owner Table:



Parking lot Table:



**Project plan**

* Scrum methodology was used to manage our product development that helped the team to work together.
* Through scrum, we managed a set of meetings (Scrum meetings) that enabled us to work coherently and efficiently as a team.
* Scrum was adopted and well suited for our application as we split the tasks into iterations that provided a sense of consistency throughout the development cycle.
* Following principles were kept in mind:
  + Transparency: Our team worked in an environment where every member was aware of what issues other team members are/might pose during the development process. Having a sense of transparency was the key in successful iteration.
  + Inspection: Recurrent inspection points allowed the team to have insights on how well the process is working.
  + Adaptation: Constant revision done by the team on how well the system was working and revise those items that were inconsistent.
* Practices:
* Sprint: Typically, sprint period is a timebox of one month or less, during which our team worked on the product with the following factors in mind:
  + Maintaining consistent duration
  + New sprint following the conclusion of previous sprint.
  + Fixing the start date and end date of each sprint.
* Daily Scrum: Daily scrum is a short discussion (usually 10-5 minutes) where our team coordinated the tasks for the day. Based on that, our team worked on those assigned tasks, for example creating the login form for the cars, changing the colors schemes, tweaking the UI etc.
* Sprint Review: Sprint review took place after the end of each sprint, which usually involved participation of whole team which then reviewed the results of sprint. The purpose of this discussion was to discuss, demonstrate, and potentially have a chance to make the desired changes before the new sprint.
* Sprint Retrospective: At the end of each sprint following the sprint review, our every team member reflected upon how things went during the previous sprints and identified various adjustments could be implemented as we go forward.

**FEATURES DEVELOPMENT AND PROGRAMMING**

**User Stories**

|  |  |  |
| --- | --- | --- |
| As a | I want to | So that I can |
| Stakeholder with parking space | Keep track of parking lots | Manage the space in a better way |
| Stakeholder with parking space | Keep track of parking fee | All records are in one place |
| Stakeholder as City/Country Transportation | Improve coordination with local transportation | Have well planned transit development |
| Stakeholder as City Police Department | Reduce the number of illegally parked vehicles on the streets | Better managed traffic in close proximity to major attractions |
| User | Find a parking space | Park my car in legal and safe place |
| User | Parking space which has fair parking fees and easy payment methods | Avoid caring quarters in my pocket |

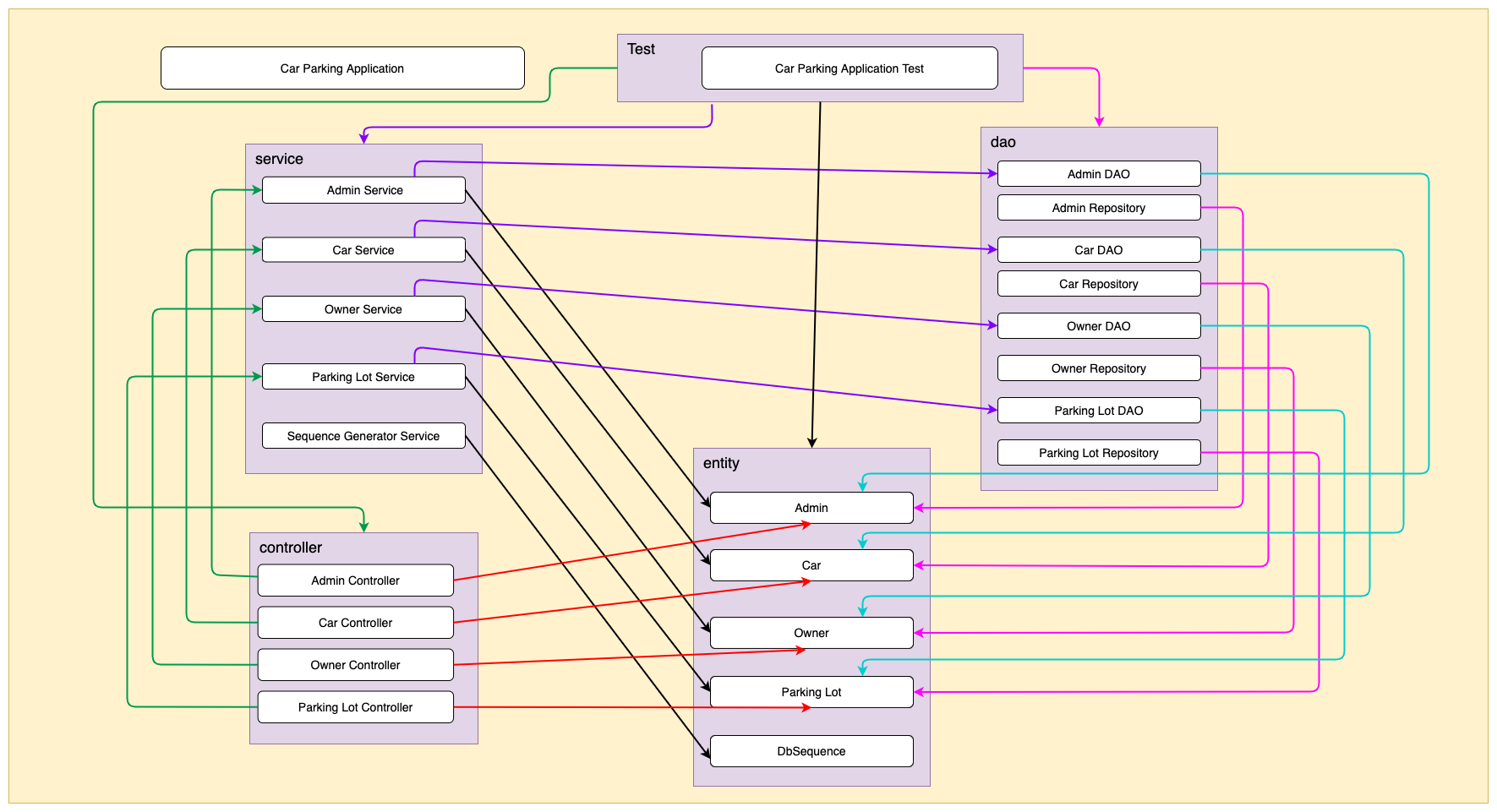
**Acceptance Testing**

Following acceptance testing methods have in practiced while developing this system:

1. Alpha Testing:
   1. The type of testing is done by the development team while developing the system.
   2. These test cases are simple as navigation showing in correct position, with correct colors and correct navigation tabs.
   3. We tested out individual entities while developing them.
2. Contract Acceptance Testing
   1. This testing is done when we need to tally the requirements and the developed system. The project team defines the relevant criteria and specifications for acceptance at the same time when the team agrees on the contract itself.
   2. We tested different scenarios such as validating the login page, validating the input values on Park car page.
   3. Tested displaying of the Show cars which shows all the car information in the system.
   4. Tested onClick event on Car Number to show the summary of the car.
   5. Test case for Exit car button should add the exit time and calculated the parking fee
3. Operational Readiness Testing
   1. It is also called Production Acceptance Testing; these test cases ensure there are workflows in place to allow the software or system to be used. This should include workflows for backup plans, user training, and various maintenance processes and security checks.
   2. This type of test cases helped us in making sure that the system work was in place and every screen was loaded within correct order and in timely manner.
   3. This helped us in understanding the quality of the system.
4. Black Box Testing is often categorized as functional testing, but can, to some extent, be seen as a type of User Acceptance Testing.
   1. It’s a method of software testing which analyzes certain functionalities without letting testers see the internal code structure. Black Box Testing is part of User Acceptance Testing, because Black Box Tests share the same principles as UAT.
   2. During Black Box Tests the user isn’t aware of any code base, but only about the requirements which the software should meet.

**UML Diagrams**

Class diagram



Use case



**Planning Development Task**

|  |  |
| --- | --- |
| Task No | Description |
| 1 | Selecting management system to be developed |
| 2 | Allocating development task |
| 3 | Designing UI screens |
| 4 | Developing files related to Car Entity |
| 5 | Developing files related to Owner Entity |
| 6 | Developing files related to Parking Lot Entity |
| 7 | Database schema creation |
| 8 | Creating unit testing cases |

**PROGAMMING AND DEVELOPMENT**

**Programming and wiring modules and layers**

There are 4 modules in our system

1. Controller:
   1. A controller is responsible for processing incoming requests. It invokes business logic, updates the model and returns the view that should be rendered. An MVC Controller is a JAX-RS resource method annotated with @Controller. If a class is annotated with @Controller, then all resource methods of this class are regarded as controllers.
   2. The controllers in the system are depend on their respective entity class and Service. Methods for getting data, putting new data and updating data are written in this class
2. DAO
   1. The DAO pattern is a structural pattern that allow us to isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API.
   2. DAO pattern is based on design principles such as **abstraction** and **encapsulation**. It will protect the rest of application from many operations in persistence layer. For example, change database from MySQL to PostgreSQL, change storage by file to database.
   3. The DAO pattern is commonly used pattern to persist domain objects into a database. The most common form of a DAO pattern is a class that contains CRUD methods for a particular domain entity type.
   4. So, DAO class is an intermediate layer to help other layers to communicate with persistence layers regardless of any storage mechanism such as file, database management, …, then, performs some operations such as CRUD.
   5. The DAO in our system has the following components:
      1. A DAO factory class.
      2. A DAO interface which is called repository class.
      3. A concrete class that implements the DAO interface.
      4. Data transfer object (sometimes called Value Objects)
3. Entity
   1. An entity is a lightweight persistence domain object. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The primary programming artifact of an entity is the entity class, although entities can use helper classes.
   2. Admin, Car Owner and Parking-Lot are the entities of our system.
4. Service
   1. A service is defined by a set of interfaces and classes. The service contains an interface or an abstract class that defines the functionality provided by the service. The consumer which is client knows only about the service interface.
   2. Services in our system are dependent on the DAO module and the Entity classes

**Unit Testing**

Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers

We have used to Mockito and Junit frameworks to write unit test cases for our system.

Test Cases

1. Login validation: Check whether admin data is validated.
2. Creating CAR: Checking if the Posting i.e., is inserting car info into the database is successful.
3. Update Car: When the car is exiting the parking space, Car information needs to be updated into the database. This test checks if the put method i.e., updates work correctly.
4. Create Owner: Checks if owner info is successfully added to the database.
5. Update parking lot: This test case finds if the parking lot is empty or full before it is allocated to the car.