Riley_Croucher_Project_Reflection

by Riley Murphy Croucher

FILE	25362_RILEY_MURPHY_CROUCHER_RILEY_CROUCHER_PROJECT_REFLECTION_312866_1645076507.PDF (829.61K)		
TIME SUBMITTED	11-DEC-2020 04:39PM (UTC+0000)	WORD COUNT	3085
SUBMISSION ID	139289554	CHARACTER COUNT	15056

ACCA7000 – ACCA7001- ACCA7002

Evaluation Report on the Design and Implementation of a Car Rental Database Application

Author: Riley Croucher

Student Number: 2013055

Degree Programme: Software Engineering (Web Development) MSc

Lecturer: Dr Michael Dacey /Dr Tim Bashford /Dr Beverley Holland

Team Members: Rokas Rudys, Zack Costello and Ridwan Chowdhury

1. Introduction

This report focuses on the evaluation and reflection of the design and implementation process for a text-based database application. The application was designed and produced during a week-long (07/12/2020 – 11/12/2020) group project whereby the group had to use Agile principles to design a Database and implement it into a Java console application. One of two possible scenarios was selected by the group to be the goal of the project.

The scenario chosen by the team was to design a text-based console application written in Java that incorporates a database design for a car rental company. The company required the application to allow customers to rent cars while also being able to track the status of cars within their inventory. It must also be able to respond to car management requests such as adding new cars to the database when they are bought and removing old cars when they are sold. The full list of requirements the app should aim to implement is shown below.

1.1 Scenario Requirements

The scenario detailed a list of operations that the customer should be able to perform. The team further discussed this list and came to the conclusion that some of the required operations seem more suited to a rental car manager rather than a customer. Therefore, as a team the list of operations was split into two categories, a category for the operations the customer would use and one category for the operations by the rental car manager as stipulated below.

Customer Operations:

- Register as a new user and gain a username and password.
- Login to the system using a registered username and password.
- Be able to select the rental car pickup location.
- The ability to browse available cars at the pickup location and employ the use of search filters to show cars according to type, make and model.
- To book an available vehicle at the pickup location by selecting a drop off location, by entering their name and home address, confirming booking details and then entering payment details.
- View existing bookings.
- Change an existing booking.
- Cancel an existing booking.

Manager Operations:

- View all existing bookings.
- Be able to add new vehicles.

- Be able to remove old or sold vehicles.
- Book a vehicle for maintenance or repair.
- View all vehicles and their status.
- Suspend bookings.
- View all vehicle rentals.
- Search for a rental.
- Record a vehicle pickup and dropoff.

2. Agile Sprint Process

The product was conducted using an Agile sprint framework which was employed for the design and implementation of the required scenario. Overall, the project consisted of one week-long (5 day) sprint, the work for the project was separated into user stories and contained within a sprint backlog. The overall sprint board was conducted on Trello which is a web-based project management tool. As a team we also created a definition of done (DoD) and a definition of ready as displayed in the GitHub read.me or below:

2.1 Sprint Backlog

As this project was only one, one-week sprint there was a merge and overlap between the product backlog and the sprint backlog. Overall, there was no product backlog as all of the user stories within the product backlog were brought into the sprint backlog as they had to be completed and although estimation of the user stories was conducted as a group, we had no prior working sprint velocity to base off how many story points we would be able to complete and thus how many stories to bring into the sprint.

Below in Figure 1 is a screenshot of the Kartastic Trello story board showing the Sprint backlog on the left within the TODO section while user stories which have been completed are located in the DONE section.

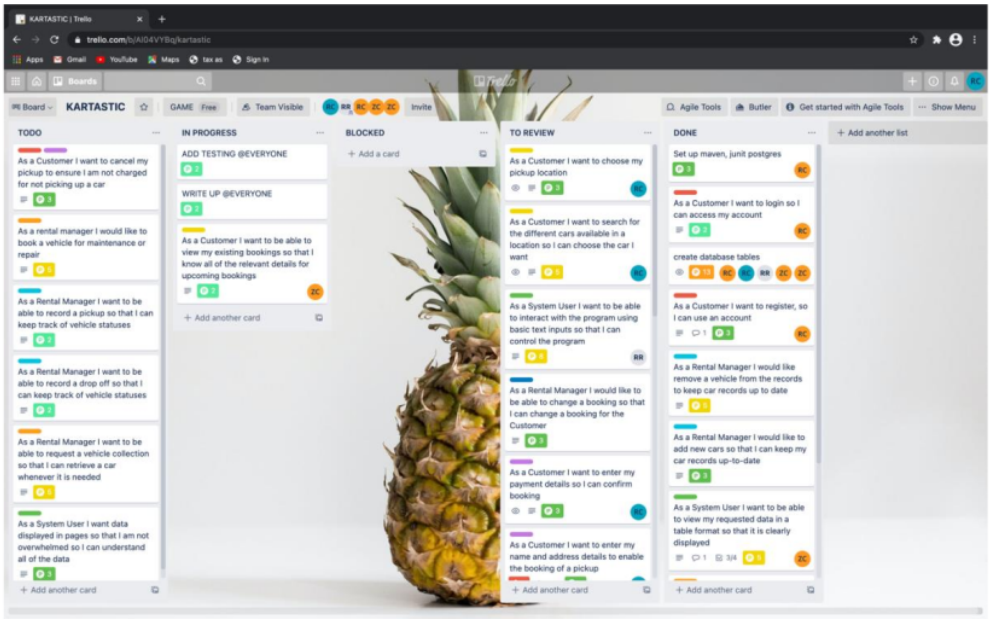


Figure 1: Showing a screenshot of the Kartastic Trello story board showing the Sprint backlog.

2.2 User Story Estimation

Within the Sprint Planning like session held on Day 0 of the sprint (07.12.20) user story estimation took place. Firstly, a user story was written based on the groups review of the success scenario criteria (refer Section 1.1) the group then discussed the user story and added acceptance criteria which reflects the groups pre-assigned DoR and DoD as shown in Figure 2 below. Each story would then be given a story point estimate based on the perception of the user story difficult and how much time and capacity should be allocated to it. This process was done by each team member giving an independent uninfluenced reflection on each member estimation, these estimations were then discussed, and a story point estimate was agreed upon. The story point estimate for the story shown in Figure 2 can be seen and thus this story was estimated by the group in Sprint planning as needing a 5 story point capacity.

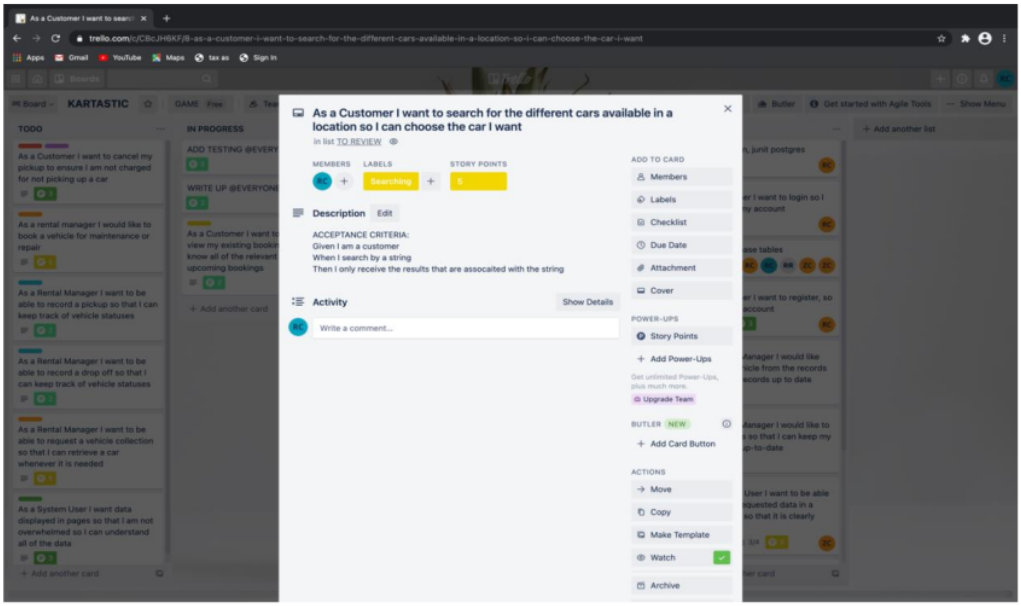


Figure 2: Showing a screenshot of a user story with a story point estimate and acceptance criteria.

2.3 Agile Sprint Evaluation

Overall, the success of the project relies fairly heavily on the Agile framework that the team employed during the project. The use of Git within the team allowed for a very high degree of team collaboration and allowed local code to be continually integrated into the Repo following the continuous Integration subset of the Agile framework. Another aspect that allowed a large degree of team collaboration was the use of daily stand-ups. Each day the team would meet and discuss the work they had completed the day prior, the work they were aiming to complete on that day and if there were any problems that they needed to flag to the wider team. These meeting were vital to allowing collaboration between functions as each team member was up to date as to what each member was doing. This concept was also highlighted through the continual updating of the Trello sprint board which meant that each time a team member started a new user story or finished an old one they would assign themselves and move it appropriately. These Agile Sprint processes ensured team transparency and prevented the team from overlapping work on user stories. Other factors of Agile that were employed that had a positive contribution to the project was the use of the similar Sprint Planning session which led to the formation of a sprint backlog of multiple stories which had already been defined and estimated in this meeting. The use of acceptance criteria and the pre-defined DoR and DoD allowed each team member to know what was expected when delivering a user story allowing conformity and continuity while also ensuring code quality.

There are several aspects of the Agile framework used in this project which could be improved to heighten the team's velocity if this project was to occur again. One large factor was the omission of Three Amigo meetings between the product owner and the developers/testers. Although the product owner set the scenario criteria these criteria are fairly ambiguous which led to some confusion as to what work was required. For example, one criterion was the ability to "Request vehicle collection", this led to confusion as each team member had their own interoperation of the criterion, one team member thought it was when a rental manager wanted to collect a car before the scheduled drop off date for while another thought it was an additional extra that the customer could pay for to allow someone from the car rental company to pick the car up from them direct rather than at a Kartastic car rental location. Therefore, having a meeting directly with the product owner as would have occurred in the three amigo meeting would lead to a conclusive unambiguous dominion of this user story. Another aspect which could be improved is the ability to estimate the squads working velocity as within this project there was no prior team experience to base story point estimations on, this would also lead to a heightened refinement of what would be capable within a one-week sprint. This could lead to the formation of a distinct product backlog which would in turn make the sprint backlog seem less daunting.

3. Database Design and Implementation

3.1 Database Design

On Day 0 of the sprint the design and implementation of the required database was decided as a priority of the project as without it starting any meaningful Java console code would be redundant. Therefore, the first day was spent designing the required database using Draw.io (a free online diagram software) to produce an ER diagram for the database. The ER diagram produced is shown below in Figure 3 and can also be downloaded as a PDF from the documents folder in the GitHub repo. The diagram shows quite an expansive database design made up of 11 different tables. This ER diagram also shows the data models between the tables (e.g. The one to many relationships). The group spent time designing this diagram to ensure once the database was created it followed the ER diagram and thus the overall database design to allow for team transparency as to how the different tables and inputs relate to one another within the application. The team also discussed and implemented the design using normalisation techniques. The database design employed is within the Third Normal Form (3NF) as the tables within the database do not include Transitive Dependency meaning there were no indirect relationships between values in the same table that could cause functional dependency.

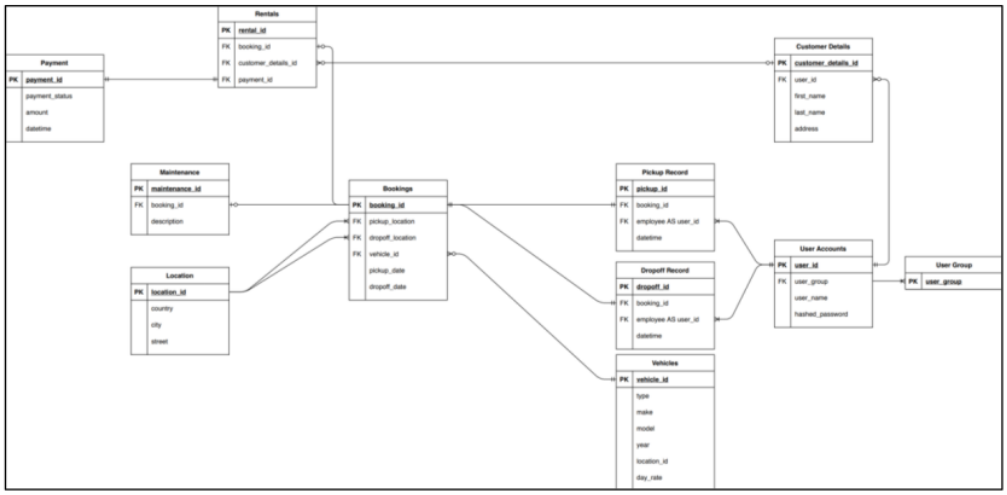


Figure 3: Showing the finalised ER diagram showing the database design.

3.2 Database Implementation

The team made the agreement that PostgreSQL would be used as the database system within this project. PostgreSQL is an open-source object-relational database system which can easily integrate into text code editors and is cross-platform which allowed team members to work on Mac, Windows and Linux.

3.2.1 User Story 1 – Database Implementation

As discussed, the database implementation was highlighted as an initial priority for the project therefore all team members assigned themselves to the user story as shown below in Figure 4.

Figure 4: Showing the initialise Database Table user story.

Each of the team then selected a couple of database tables to initialise and create and also seed and test. The following description is of the tables completed by the author and show how the team implemented the database. As this was mob programming on one user story the team moved the story as a group from the TODO section of the sprint board into the IN PROGRESS section.

The selected table that will be discussed is the creation of the Pickup Record table and how this was seeded. Figure 5 below shows the function that was used to create this table. The first step is to to connect to the database destination in this case the Kartastic database. Once connected the use of SQL queries were used as can be shown between lines 637 and 647 in figure. These quires first create the table and its name before adding the columns of the table with their constraints (E.G INT/NOT NULL). To add the correct foreign keys to the table a further constraint must be added to the end of the quire and reference the key location.

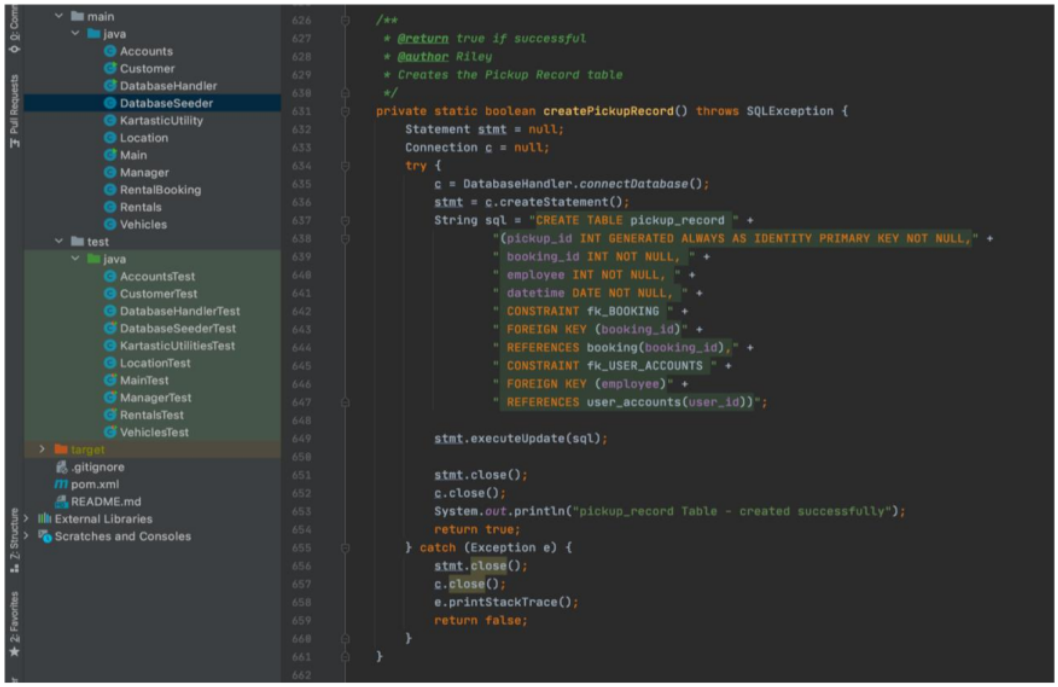


Figure 5: Showing the code that creates the Pick-up record table.
(src/main/java/DatabaseSeeder)

To then fully implement a realistic database this table must then be seeded with data. This code is shown within Figure 6 below. To seed the pickup table the connection is made with the database again and the amount of data to be seeded and generated is selected by a for loop. It then selects the information needed through SQL queries and constraints before entering the required data which it extracts from the other database tables. The data found within the other tables is generated using the fake data generator library.

```
/**
 * @return true if successful
 * @author Riley
 * Seeds the pickup record table
 */
private static boolean seedPickupRecord() throws SQLException {
    Statement stmt = null;
    Connection c = null;
    try {
        c = DatabaseHandler.connectDatabase();
        stmt = c.createStatement();
        //gets a booking record id from registered rentals and checks finds those that havent been registered yet, if found will create new pickup record fo
        for (int i = 0; i < NUMBER_OF_RENTALS_PICKUP_RECORDS * RATIO_OF_PICKEDUP; i++) {
            ResultSet r = stmt.executeQuery( sql: "SELECT booking_id, pickup_date FROM booking WHERE booking_id NOT IN (SELECT booking_id FROM pickup_record)
            r.next();
            int booking_id = r.getInt( columnIndex: 1);
            Date date_pickup = r.getDate( columnIndex: 2);

            r.close();
            String sql = "INSERT INTO pickup_record (booking_id,employee, datetime) "
                    + "VALUES (" + booking_id + ",'" +
                    "            (SELECT user_id FROM user_accounts" +
                    "                WHERE user_group = 'employee' " +
                    "                    or user_group = 'manager' ORDER BY RANDOM() LIMIT 1), '" + date_pickup.toString() + "');";
            stmt.executeUpdate(sql);
        }
        stmt.close();
        c.close();
        System.out.println("pickup_record Table - seeded successfully");

        return true;
    } catch (Exception e) {
        stmt.close();
        c.close();
        e.printStackTrace();
        return false;
    }
}
```

Figure 6: Showing the code that seeds the Pick-up record table (src/main/java/DatabaseSeeder)

Overall, this user story was completed through mob programming the different database tables independently and the completion of this user story led to the completion of the database implementation.

4. Coding and Testing

The ultimate aim of this project was to design a text-based console application, and this was achieved through the implementation of Java programming. This section will introduce two further coding User stories achieved within the project.

4.1 User Story 2 – Searching

Figure 7 below shows the first user story that will be discussed within this section and highlights that the story aimed to implement the ability for a customer to see all the cars available at a specific location.

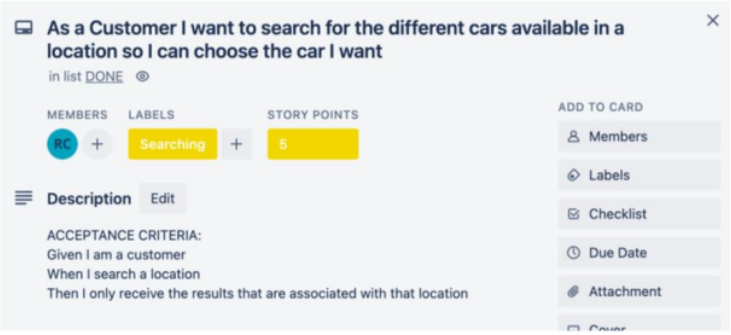
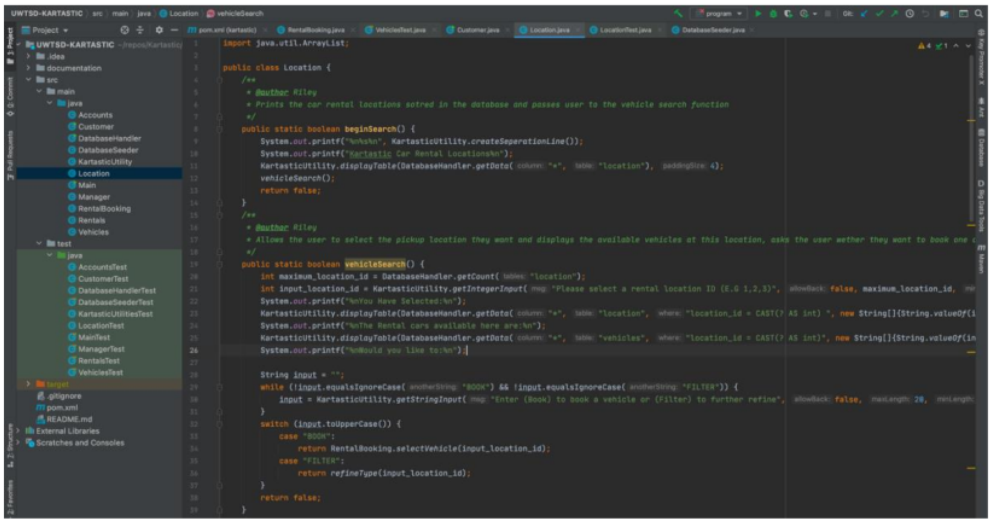


Figure 7: Showing the location search user story.

This story was assigned to the author and moved from the sprint backlog into the sprint. The code was then designed to enable the customer to input a location ID number to specify their location before being able to see the cars available at this location before asking whether they want to select a car. This code can be seen in Figure 8.



```
import java.util.ArrayList;

public class Location {
    /**
     * @author Riley
     * Prints the car rental locations sotred in the database and passes user to the vehicle search function
     */
    public static boolean beginSearch() {
        System.out.printf("%n%s%n", KartasticUtility.createSeperationLine());
        System.out.printf("Kartastic Car Rental Locations%n");
        KartasticUtility.displayTable(DatabaseHandler.getData( column: "*",  table: "location"),  paddingSize: 4);
        vehicleSearch();
        return false;
    }
    /**
     * @author Riley
     * Allows the user to select the pickup location they want and displays the available vehicles at this location, asks the user wether they want to book one
     */
    public static boolean vehicleSearch() {
        int maximum_location_id = DatabaseHandler.getCount( table: "location");
        int input_location_id = KartasticUtility.getIntegerInput( msg: "Please select a rental location ID (E.G 1,2,3)",  allowBack: false, maximum_location_id,
        System.out.printf("%nYou Have Selected:%n");
        KartasticUtility.displayTable(DatabaseHandler.getData( column: "*",  table: "location",  where: "location_id = CAST(? AS int) ", new String[]{String.valueOf(i
        System.out.printf("%nThe Rental cars available here are:%n");
        KartasticUtility.displayTable(DatabaseHandler.getData( column: "*",  table: "vehicles",  where: "location_id = CAST(? AS int)", new String[]{String.valueOf(in
        System.out.printf("%nWould you like to:%n");

        String input = "";
        while (!input.equalsIgnoreCase( anotherString: "BOOK") && !input.equalsIgnoreCase( anotherString: "FILTER")) {
            input = KartasticUtility.getStringInput( msg: "Enter (Book) to book a vehicle or (Filter) to further refine",  allowBack: false,  maxLength: 20,  minLength:
        }
        switch (input.toUpperCase()) {
            case "BOOK":
                return RentalBooking.selectVehicle(input_location_id);
            case "FILTER":
                return refineType(input_location_id);
        }
        return false;
    }
```

Figure 8: Showing the code completed for the story (src/main/java/Location).

To best complete this story two functions were made, one called the beginSearch function that will be called if the customer selects this pathway from the customer menu and two the vehicleSearch function that allows the user to search the vehicles. The beginsearch function prints all of the Kartastic car rental locations stored in the database before passing the user to the vehicleSearch function. The user is then prompted to input the location_ID of the pickup location they would like to rent from. This variable is stored as an integer variable and validation is completed to ensure that the customer enters a correct location. This variable is then used to print the address of the selected location to ensure the customer knows where they have selected before being used to print all available cars at this location. This printed table shows the complete vehicle table related to that location_ID and therefore the customer can then have the option to book one of the displayed vehicles by entering BOOK and being passed to the selectVehicle function in the RentalBooking class or to filter the search which is the subject of the next example user story.

4.2 User Story 3 – Filtering

One of the stories worked on was adding the ability for the customer to filter cars by their types, makes, models and availability as documented within Figure 9 below. This story is a step process function whereby the customers filters are applied one by one until they see the car they are looking for.

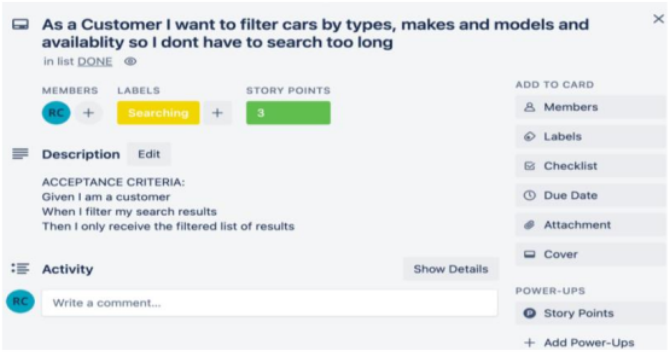


Figure 9: Showing the filtering User story that was implemented.

Some of the code completed for this user story is shown within Figure 10 below. It starts by creating a function called refineType() and parsing in the input_location_ID variable from the above story. This code works by scanning the database for the types of cars that are available and located in that ID, it then makes an array of the DISTINCT vehicle types using an SQL query and prints this as a table for the customer to see. The customer is then able to enter a string variable that is the name of the vehicle type they are looking for, this variable is validated and checked to ensure they have entered a vehicle type and that it is located at that location. The code then prints a table of all the vehicles that are that specific type at that location. The customer is then able to select to further refine the search or to book a vehicle they have seen. The next functions (refer to src/main/java/Location) refineMake and refineModel do the same process but further filter the search results to include all the previous filters. For example, the refineMake function filters for cars at the inputted location that are the inputted type and are made by the inputted manufactures.

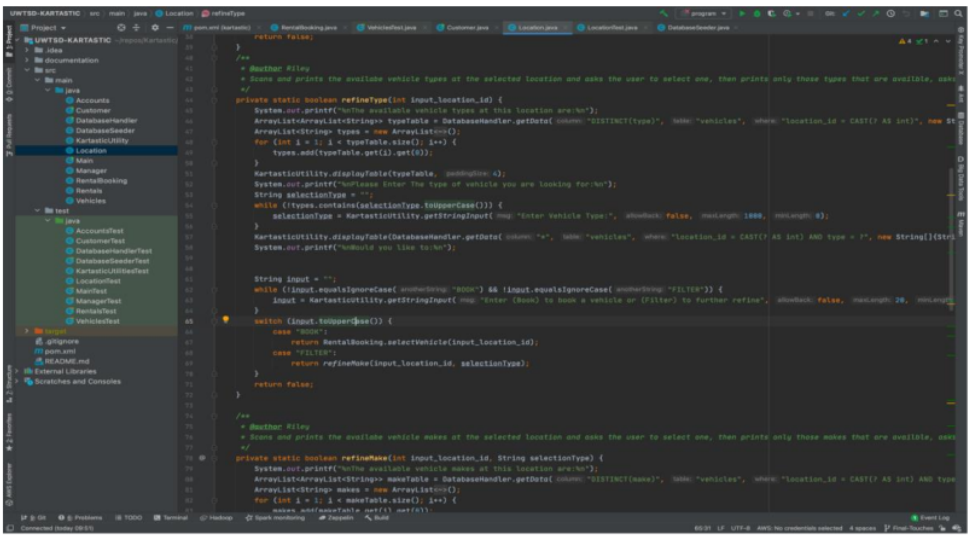


Figure 10: Showing the code completed for the story (src/main/java/Location)

4.3 Testing Reflection

Testing was an area within this project that I struggled with. Due to the time constraints of a five-day project as well as submitting a report I feel like this is one of the areas which could have been expanded on much more. Throughout the code I have implemented I have included validation methods to the input and have tested my functions by eye, but I was unable to implement any unit tests. I think a reason for this is that I spent a lot of time trying to implement console testing which I have no experience in as I thought this would work better for the functions I created.

I was able to get a grasp of the testing required for this project as I pair programmed the VehiclesTest class with Rokas, but this came at the end of the project and although I tried to implement similar testing to the location class I ran out of time. This pair programmed class can be found at (src/test/java/VehiclesTest) and is some of the limited unit tests I managed to contribute to. Another section of the testing within this project I was able to assist in was the coding of the DatabaseTest classes although hindsight now dictates that using a database management tool such as DBeaver would have enabled testing of individual database tables.

Overall, I think that the importance of writing tests as you write functions has been one of the largest learning elements I will take away from this project.

5. My Contributions

The main bulk of my contributions lie within the customer side of the application as I completed the code for both the location class (src/main/java/Location) and the RentalBooking class (src/main/java/RentalBooking). Overall, I was able to finish 8 user stories which equated to an estimated 24 story points although there was very limited testing.

Breakdown Overview:

Number of stories marked done by team :18
Number of stories I completed: 8

Number of story points completed by team: 62
Number of story points I completed: 24

Riley_Croucher_Project_Reflection

GRADEMARK REPORT

FINAL GRADE

84/100

GENERAL COMMENTS

Instructor

Software design evaluation (30): 25

Database evaluation and implementation (30): 23

Coding and Testing (30): 23

Reflection (10): 8

A good, cohesive report which has clearly been well-structure and considered.

Good overall reflection on each element of the process of being a team member.

A well-written and fully featured solution to the problem. The code is clean, though would benefit from a few more comments in places. The structure of the code is logical and well-considered.

The vehicle class appears to be well tested, but the remaining code lacks coverage.

Clear representation of the database requirements represented through the ERD. Be careful to test for data duplication, there's some rationalisation possible around rentals/payments. Otherwise a good solution.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12