

Sprint Ceremonies and Database design

Sprint Planning

The project week began with an agile approach, starting with sprint planning. This facilitated collaborative discussions among the group to break down the user requirements presented into user stories (Fig 2-4). Each user activity and story were thoroughly discussed and assigned a size using dot voting (Fig. 1).

Some of the identified user activities, tasks, and their corresponding user stories are presented below:

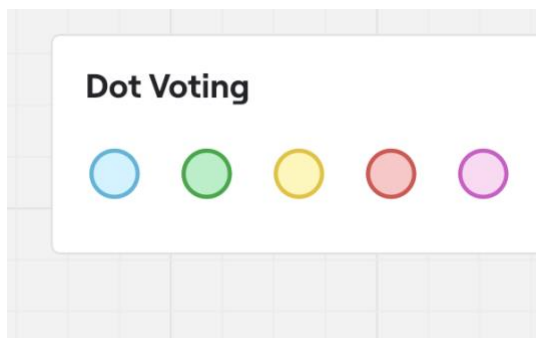


Figure 1: Dot voting (sizes arranged from smallest to largest, left to right). Appendix C.

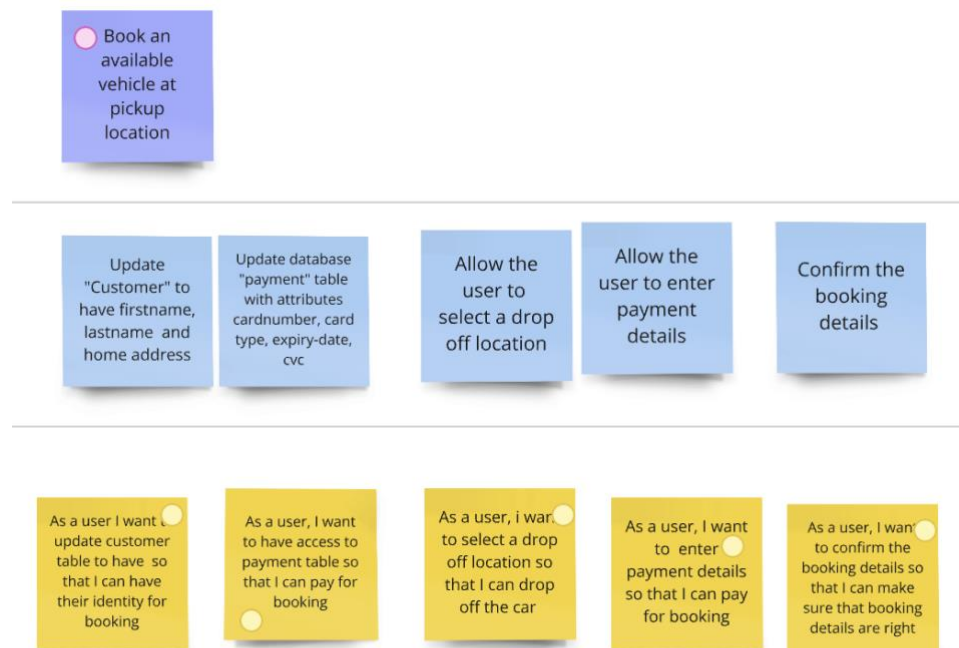


Figure 2: User Story Mapping for booking a rental vehicle. Purple card represents a user activity, blue cards represent user tasks, and yellow cards represent user stories. Appendix C.

As a group, we decided that booking a rental vehicle from a pickup location would involve several key steps. First, a customer must be interested in booking a vehicle, which requires a system to track customer information. This led to the creation of a user story to add a Customer table to the database.

To confirm the booking, the user must first select a pickup location, choose a vehicle from the available options, and then enter payment details. Each of these actions requires specific data to be stored and retrieved from the database. The user stories for the Car and Location tables have already been created, which address the need for vehicle and location information, respectively.

However, in order to complete the rental booking process, a Payment table is needed to store payment information and confirm the transaction. This new table would fulfil the requirements for processing payments and finalizing the rental booking. Because this user activity was one of the largest which involved several user stories and interactions between different tables, we decided that this would constitute core part of our minimum viable product (MVP).

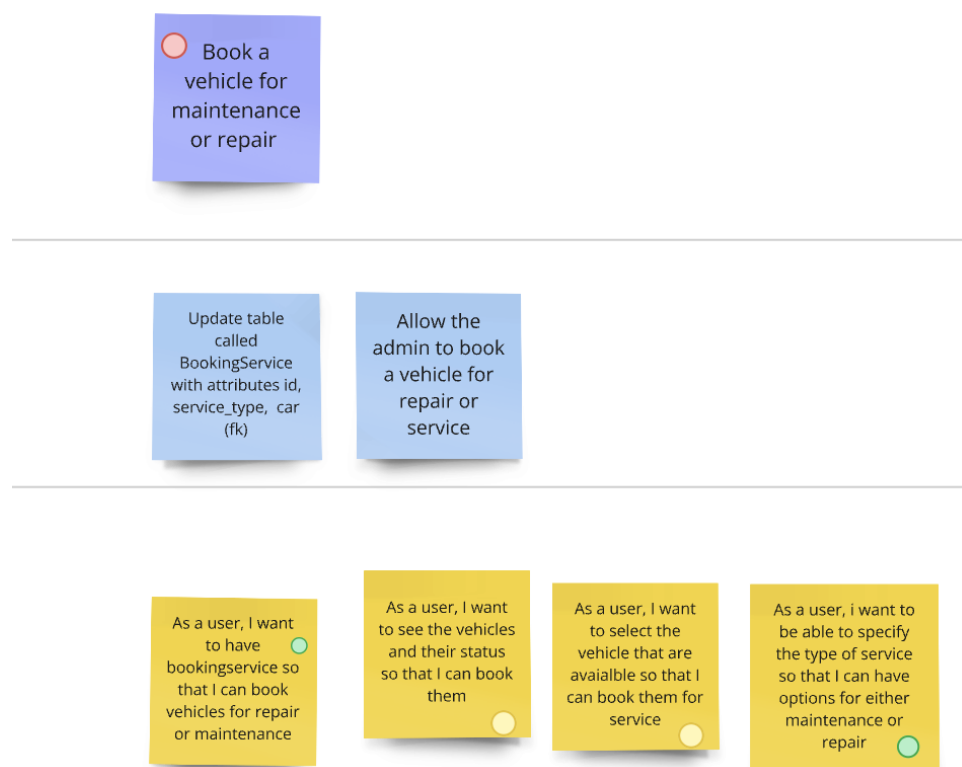


Figure 3: User Story Mapping for booking a vehicle for maintenance and repair. Purple card represents a user activity, blue cards represent user tasks, and yellow cards represent user stories.

The booking of a vehicle required its own table in the database, which led to the creation of a user story to manage and track these records. Rather than having separate tables for repair and maintenance, our team decided to consolidate them into one table called "Booking Service." This table would specify the type of service a car is being booked for, ensuring better organization and accessibility.

The user activity for booking a vehicle service involves several steps. First, the user needs to be able to view all available vehicles. This is crucial, as cars that are already booked, either for rental or service, or cars that are otherwise unavailable, should not be available for booking. Once the user has browsed the available options, they should be able to select the desired vehicle and specify the type of service they wish to book for that vehicle. This activity was one of the more sizable tasks in our project, and therefore it was considered a "good-to-have" feature in case we were able to complete our MVP.

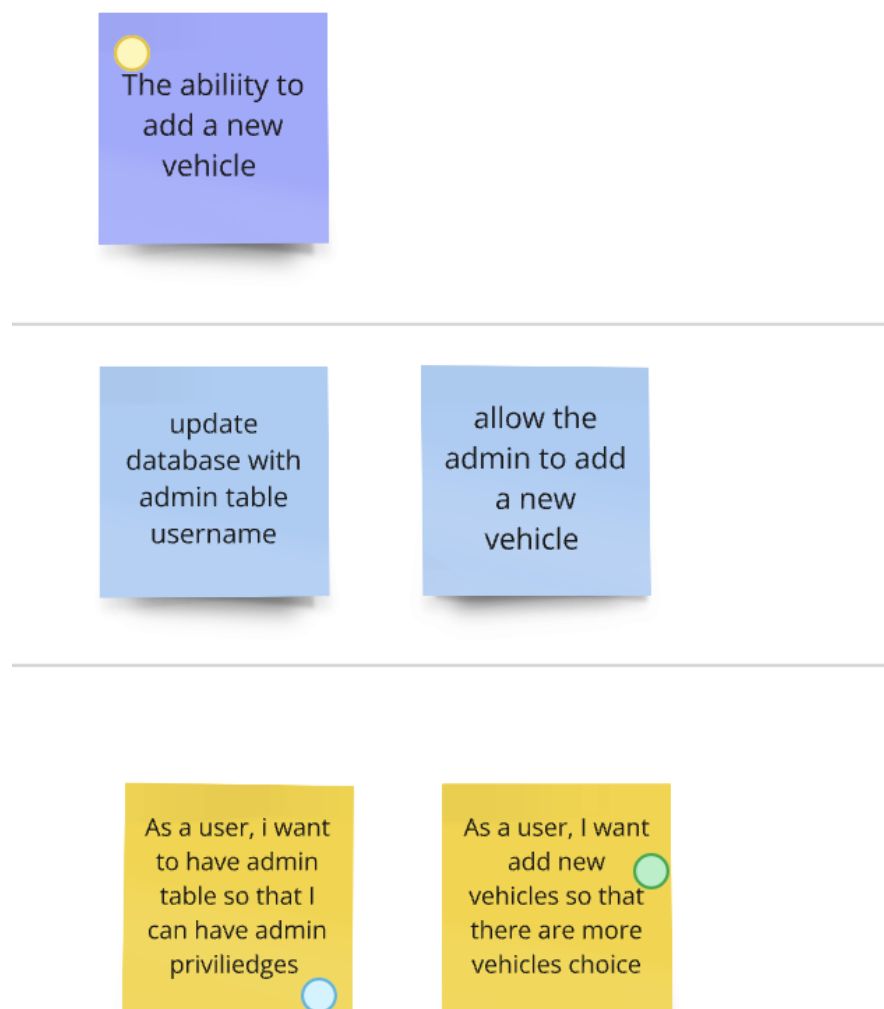


Figure 4: User Story Mapping of adding a new vehicle. Purple card represents a user activity, blue cards represent user tasks, and yellow cards represent user stories.

The user activity of adding a new vehicle prompted our team to consider the need for different types of users in the application. For this activity, we decided that an "Admin" user role would be necessary, as it would have additional privileges. As a result, a user story was created to implement an Admin table. The Admin user would have access to extra features that a regular user would not, such as the ability to add new vehicles to the database.

Database Design

After the user stories were assigned and added to the backlog in Trello (Appendix B), my team and I decided to form relationships between the different entities in our user stories by creating an Entity Relationship Diagram (ERD) (Fig. 5). This diagram helped us visualize and organize the connections between various entities, ensuring that our database structure would be logical and well-aligned with the user requirements. By mapping out these relationships, we were able to identify potential issues early on and ensure a smoother development process.

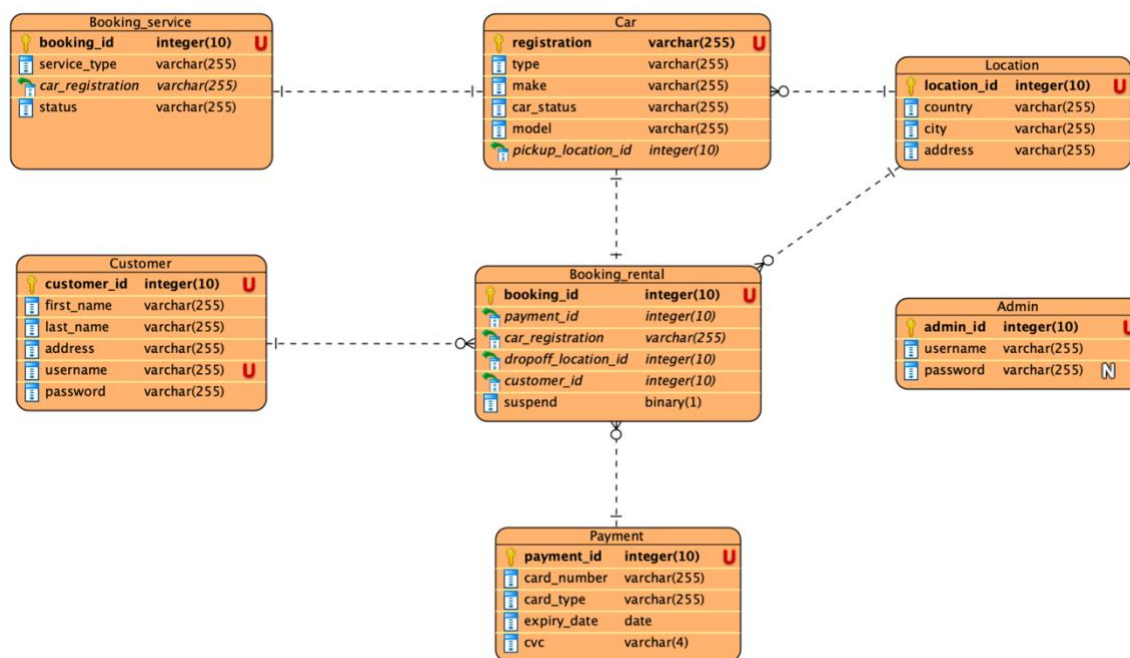


Figure 5: Entity Relationship Diagram (ERD) of Car Rental Application.

The Car Rental Application is designed with seven key tables: Car, Booking Service, Booking Rental, Payment, Location, Customer, and Admin (Fig. 5). This structure was initially conceptualized in alignment with our Minimum Viable Product (MVP) requirements. At the core of the application is the Booking Rental table, which contains all the necessary attributes to facilitate the booking of a rental.

The Booking Rental table has a many-to-one relationship with the Payment table, as we decided that a payment card could be used for multiple booking rentals. Similarly, the Booking Rental table also has a many-to-one relationship with the Customer table, allowing a customer to make multiple booking rentals.

There is a one-to-one relationship between the Booking Rental table and the Car table, as a car can only be booked for one rental at a time. The Car table also has a one-to-one

relationship with the Booking Service table for the same reason: each car can only be associated with one service booking at a time.

The Location table has a many-to-many relationship with both the Car and Booking Rental tables. Each car has a pickup_location_id attribute, as it must be picked up from a specific location. Additionally, each booking rental includes a dropoff_location_id, where the customer specifies the location to drop off the car. Since a locations can serve as both pickup and drop-off points for various cars at a time, a one-to-many relationship is necessary.

The Admin table was initially intended to handle administrative tasks, such as assigning services like booking services. However, since the focus was initially on the MVP, we decided not to have the Admin table directly relate to the Booking Service table at this stage.

Daily Standups

The use of daily standups, aligned with the Trello board (Appendix B), not only allowed the team to collaborate effectively but also facilitated a flexible and adaptive working environment. Each standup focused on tasks from the previous day, enabling team members to communicate their progress and any obstacles they encountered. This constant communication helped us adjust our tasks as needed.

For instance, when one team member encountered an issue with their database, preventing them from working on most of the app's functionality, we quickly reassigned them to a task that didn't require database connectivity: creating the menu for the console. This ensured that they remained productive despite the database issue. In the meantime, I took over their ticket for customer registration, ensuring that progress continued on all fronts.

Sprint Retrospective and Agile

During the Sprint Retrospective, we discussed the effectiveness of our collaboration, the major hurdles we encountered, and areas where improvements could be made. The team's collaboration was excellent, with a strong emphasis on adaptability. We also reflected on ways to improve, such as breaking down tasks into smaller, more manageable user stories to enhance clarity and efficiency.

Personally, the retrospective made me realize how unforeseen challenges, like database connectivity issues, can arise despite thorough preparation. However, it also highlighted the value of agile practices, such as pair programming, which can be extremely helpful in overcoming such problems. By working closely together, we were able to solve issues more efficiently and maintain progress even when faced with unexpected setbacks.

Individual Code and testing - Appendix A

Day: 17/12/24

I worked on developing MVP functionality, which included creating booking rentals, retrieving booking rentals by customer ID, and adding payment methods.

Excerpt of code work for 17/12/24 , Tuesday

```
public static boolean createBookingRental(int payment_id, String registration_plate, int dropoff_location_id, int customer_id) {
    String insertQuery = "INSERT INTO booking_rentals (payment_id, registration_plate, dropoff_location_id, customer_id, suspend) VALUES (?, ?, ?, ?, ?)";
    try (Connection conn = Main.establishConnection()) {
        PreparedStatement stmt = conn.prepareStatement(insertQuery);
        stmt.setInt(1, payment_id);
        stmt.setString(2, registration_plate);
        stmt.setInt(3, dropoff_location_id);
        stmt.setInt(4, customer_id);
        stmt.setBoolean(5, false);

        int rowsAffected = stmt.executeUpdate();

        return rowsAffected > 0;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 6: Create Booking Rental

```
@Test
public void testCreateBookingRental() throws SQLException {
    // mock the Connection and PreparedStatement
    Connection mockConn = mock(Connection.class);
    PreparedStatement mockStmt = mock(PreparedStatement.class);

    when(mockConn.prepareStatement(anyString())).thenReturn(mockStmt);

    // test the method with mocked connection
    String insertQuery = "INSERT INTO booking_rentals (payment_id, registration_plate, dropoff_location_id, customer_id, suspend) VALUES (?, ?, ?, ?, ?)";
    PreparedStatement stmt = mockConn.prepareStatement(insertQuery);
    stmt.setInt(1, 1);
    stmt.setString(2, "ABC123");
    stmt.setInt(3, 4);
    stmt.setInt(4, 7);
    stmt.setBoolean(5, false);
    stmt.executeUpdate();

    // verify
    verify(mockStmt, times(1)).executeUpdate();
}
```

Figure 7: Testing for create Booking Rental

```
/Users/parjalrai/Library/Java/JavaVirtualMachines/openjdk-23.0.1/Contents/Home/bin/java ...
WARNING: A Java agent has been loaded dynamically (/Users/parjalrai/.m2/repository/net/bytebuddy/byte-buddy-agent/1.14.12/byte-buddy-agent-1.14.12.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

org.mockito.exceptions.base.MockitoException:
Mockito cannot mock this class: interface java.sql.Statement.

If you're not sure why you're getting this error, please open an issue on GitHub.

Java
  vendor name : Oracle Corporation
  vendor version : 23.0.1+11-59
  name : OpenJDK 64-Bit Server VM
  version : 23.0.1+11-59
```

Figure 8: Error running tests

The booking rental creation was a core feature of the MVP. This function accepts all necessary input for booking a rental and inserts it into the database using an SQL query. When a new booking rental is successfully created, `stmt.executeUpdate()` should add a row to the database, returning true. If no row is added, it returns false (see Fig. 7).

The testing focused on verifying the behaviour of the booking rental creation process. Due to issues with running the tests as intended (see Fig. 8), I implemented a workaround to test the method's internal behaviour. Specifically, the test confirmed that executing the insert

query triggered `stmt.executeUpdate()` exactly once, indicating that the functionality was working correctly.

Day: 18/12/24

With the MVP essentially complete, I began working on additional tickets from the backlog. This included implementing customer registration, developing admin functionalities such as adding new cars, suspending and unsuspending cars, and creating the menu function (see Fig. 9).

Excerpt of code work for 18/12/24 , Wednesday

```
Admin Login:
Please use Username: 'admin01' and Password: 'adminpass321' as an example.
Please enter your Username (or 'x' to cancel) :
admin01
Please enter your Password (or 'x' to cancel) :
adminpass321
Login Successful
Please select one of the following options:
1. View all Cars
2. Add a Car
3. Suspend a Car
4. UnSuspend a Car
5. Book a Car for Servicing
6. Go Back
7. Exit
```

Figure 9: Menu functionality on console

```
public static void viewAllBookingRentalsBySuspend(boolean suspendStatus) {
    System.out.println("Please enter the booking ID:");
    try (Connection conn = Main.establishConnection()){
        Statement stmt = conn.createStatement();

        // query to select all data from the booking_rentals table
        ResultSet rs = stmt.executeQuery("SELECT * FROM booking_rentals where suspend = " + suspendStatus);

        // print a header for the output
        System.out.println("Booking Rental ID | Payment ID | Registration Plate | Dropoff Location ID | Customer ID | Suspend");

        // process each row in the result set
        while (rs.next()) {
            // retrieve each column value
            int bookingRentalId = rs.getInt("booking_rental_id");
            int paymentId = rs.getInt("payment_id");
            String registrationPlate = rs.getString("registration_plate");
            int dropoffLocationId = rs.getInt("dropoff_location_id");
            int customerId = rs.getInt("customer_id");
            boolean suspend = rs.getBoolean("suspend");

            System.out.printf("%d | %d | %s | %d | %d | %s\n",
                bookingRentalId, paymentId, registrationPlate, dropoffLocationId, customerId, suspend);
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

Figure 10: View All Booking Rentals By Suspend

```

@Test
void testAllActiveBookingRentals() throws SQLException {
    Statement stmt = mock(Statement.class);
    ResultSet rs = mock(ResultSet.class);

    // behavior of the mock ResultSet
    when(stmt.executeQuery( sql: "SELECT * FROM booking_rentals where suspend = true")).thenReturn(rs);
    when(rs.next()).thenReturn( t: true, ...ts: false); // Simulate one row in the result set

    // mock the data for each column in the result set
    when(rs.getInt( columnLabel: "booking_rental_id")).thenReturn( t: 101);
    when(rs.getInt( columnLabel: "payment_id")).thenReturn( t: 202);
    when(rs.getString( columnLabel: "registration_plate")).thenReturn( t: "XYZ 1234");
    when(rs.getInt( columnLabel: "dropoff_location_id")).thenReturn( t: 303);
    when(rs.getInt( columnLabel: "customer_id")).thenReturn( t: 404);
    when(rs.getBoolean( columnLabel: "suspend")).thenReturn( t: true);

    // simulate the logic that uses the Statement and ResultSet
    ResultSet result = stmt.executeQuery( sql: "SELECT * FROM booking_rentals where suspend = true");

    // assert
    assertNotNull(result);
    assertTrue(result.next()); // Should be true for the first row

    assertEquals( expected: 101, result.getInt( columnLabel: "booking_rental_id"));
    assertEquals( expected: 202, result.getInt( columnLabel: "payment_id"));
    assertEquals( expected: "XYZ 1234", result.getString( columnLabel: "registration_plate"));
    assertEquals( expected: 303, result.getInt( columnLabel: "dropoff_location_id"));
    assertEquals( expected: 404, result.getInt( columnLabel: "customer_id"));
    assertTrue(result.getBoolean( columnLabel: "suspend")); // Expecting true for suspend

    // verify
    verify(stmt).executeQuery( sql: "SELECT * FROM booking_rentals where suspend = true");

    // clean up
    result.close();
}

```

Figure 11: Test for view all books by rentals where suspend is true

The `viewAllBookingRentals()` method dynamically takes a boolean parameter, `suspend`, and retrieves all booking rentals that are either active or suspended based on its value. The test primarily verifies the behaviour when the `suspend` status is true. All database interactions are mocked to avoid direct interaction with the database. The test confirms that the method queries `booking_rentals` on the backend and returns the expected values, which are validated using `assert` and `verify` methods.

Day: 19/12/24

Excerpt of code work for 19/12/24, Thursday

Finally, I focused on enhancing the menu, implementing the functionality for booking services, and creating the README file.

```

public static boolean checkForValidCar(String regplate){ 3 usages  Parjal Rai *
    if(regplate == null || regplate.isEmpty() || regplate.length() != 7){
        return false;
    }

    try(Connection conn = Main.establishConnection()){
        String query = "SELECT * FROM cars WHERE lower(registration_plate) = ?";
        PreparedStatement stmt = conn.prepareStatement(query);
        stmt.setString(1, regplate.trim().toLowerCase());

        ResultSet rs = stmt.executeQuery();

        return rs.next();
    }catch(SQLException e){
        throw new RuntimeException(e);
    }
}

```

Figure 12: Check for valid car method. Used for booking service for valid car.

```

@Test  Parjal Rai *
void testCheckForValidCarRegPlate() throws SQLException {

    Statement stmt = mock(Statement.class);
    ResultSet expected = mock(ResultSet.class);

    String reg_plate = "XY231BA";

    when(stmt.executeQuery( sql: "SELECT * FROM cars where registration_plate = " + reg_plate)).thenReturn(expected);
    when(expected.next()).thenReturn( true, false);
    when(expected.getString( columnLabel: "car_type")).thenReturn( "Sedan");
    when(expected.getString( columnLabel: "make")).thenReturn( "Toyota");
    when(expected.getString( columnLabel: "model")).thenReturn( "Corolla");

    ResultSet result = stmt.executeQuery( sql: "SELECT * FROM cars where registration_plate = " + reg_plate);

    assertNotNull(result);
    assertTrue(result.next());
}

```

Figure 13: Check for valid car method test.

The checkForValidCar() method enables users to check if a car is available before booking it for service (see Fig. 12). The test validates that if the registration plate exists in the car database, the method returns true.

In total estimation, I have done around 50 story points.

Appendices

Appendix A-

Sprint Planning Meeting

Date: 16/12/24

Time: 11:00 AM

Agenda:

1. Develop a user story map.
2. Write up user stories from requirements and set them on the Kanban board.
3. Allocate size estimates for each task.
4. Identify versions of the code for the Minimum Viable Product (MVP).

Action Items:

Completed:

- Developed the user story map as a team.
- Broke down user stories into individual tasks.
- Created an Entity Relationship Diagram (ERD) to represent the database.

Pending:

- Allocate size estimates for tasks.
- Identify the MVP.

Stand-Up Meeting 1

Date: 17/12/12

Time: 10:00 AM

Agenda:

1. Review story map and rate user stories.
2. Identify MVP.
3. Allocate tasks.

Action Items

Jay:

- Implement functionality to view all locations.
- Implement functionality to view all cars by location ID.
- Implement functionality to view cars by location and availability.

Rid:

- Develop login functionality.
- Create a customer table.
- Implement logic to save the ID of the customer.

PJ:

- Create a Payment table and implement the createPayment method.
- Create a Booking table and implement the createBookingMethod.

Blockers and Updates:

- Completed allocation of size estimates for tasks.
- Identified MVP user stories.

Stand-Up Meeting 2

Date: 18/12/24

Time: 10:00 AM

Format: Daily Stand-Up Ceremony

Agenda:

1. Run through previous tasks and status.
2. Discuss blockers.
3. Review customer and admin user flow.

Action Items

Jay:

- Allow users to update vehicle booking drop-off locations.
- Allow users to select a car to book from a location.
- Add functionality to filter and browse cars by “type” at the pickup location.
- Enable viewing of individual cars.

Rid:

- Continue developing login functionality.
- Create a menu for customer and admin flow.

PJ:

- Create a function to cancel a booking.
- Develop the admin login functionality.
- Allow the admin to add new vehicles.
- Implement booking and suspending functions.

Blockers and Updates:

1. PGAdmin was down, so Rid could not progress on backend tasks.

Stand-Up Meeting 3

Date: 19/12/24

Time: 10:00 AM

Agenda:

1. Run through previous tasks and status.
2. Discuss blockers.
3. Review customer and admin user flow.

Action Items

Jay:

- Allow users to update vehicle booking drop-off locations.
- Allow the user to view each car
- Testing

Rid:

- Check Flow map is working in the correct flow
- Testing

PJ:

- Allow the admin to add new vehicles.
- Implement booking and suspending functions.
- Testiing

Blockers and Updates:

2. PGAdmin was down, so Rid could not progress on backend tasks.

Retrospective

Date: 19/12/24

Time: 10:00 AM

What Helped Us Move Forward?

1. **Paired Programming:**
 - a. Collaborating effectively to solve problems faster and share knowledge.
2. **Planning Database and User Store Together:**
 - a. Ensured alignment on key data structures and requirements.
3. **Identifying What We Can Do with Our Current Knowledge:**
 - a. Helped focus efforts on achievable goals and built confidence.
4. **Support from Team Members When the Database Was Down:**
 - a. Team collaboration minimized the impact of unexpected issues.

What Helped Us Move Forward?

1. **Database Access Issues:**
 - a. Significant time was wasted attempting to fix access problems, delaying progress.
2. **Adapting to New Processes:**
 - a. Adjusting from the DVLA application workflows to the new project environment took time.
3. **Merge Conflicts:**
 - a. Frequent merge conflicts disrupted development and required resolution time.

How Could We Do Things Differently?

1. **Pre-Project Setup:**
 - a. Ensure all Macs and necessary systems are fully operational before the project begins.
2. **Jira Alternative Selection:**
 - a. Decide on an application to replace Jira early to avoid delays during the project.
3. **Smaller User Stories:**
 - a. Break down user stories into smaller, actionable tasks to create a more achievable MVP and track progress better.

Appendix B – Trello Board link: <https://trello.com/b/5j4wosMm/carrentalapp>

Appendix C – Miro Board link:

https://miro.com/app/board/uXjVL2z3-el/?userEmail=bayingray123%40gmail.com&track=true&utm_source=notification&utm_medium=email&utm_campaign=add-to-board&utm_content=go-to-board&lid=lmk46fuygo6b