Vroom Technical Solution Design

Version: V1.0
Date: 25/09/2019
Sponsor: RMIT University
Author:

- Ignatia Stellarista
- Sarah Nurwidhiafitri Sukamto
- Sefira Karina
- Supun Kwan
- Thach Nguyen

Document Control

Distribution

Version	Issued	Recipient	Position
V 1.0	12/08/2019	Melina Vidoni	Sponsor
V 1.1	05/11/2019	Melina Vidoni	Sponsor

Amendment History

Section	Page	Version	Comment
<Enter Doc. Section No.>	<Enter Page No.>	1.1	Updated after client's draft comment.

.

Staff or Entities Consulted

NAME	Position / Organization
Melina Vidoni	Client, Sponsor

Related Documents

Name	Author	Description
Project Folder	Stella, Sarah, Robin, Sefira, Tyler	This is the project's folder. Within it, there are files needed for the documentation of this project. The folder can be found here.

Preface

The purpose of this document is to give a detailed overview of the technical specification of the product.

Table of Contents

1 Introduction

The purpose of this project is to develop a web application for a car share scheme. As the business owns a number of cars for rent located around the CBD, the project is intended to provide features of the system to allow their customers to look up and book for a car, handle transactions and data management which makes the business more structured, convenient and expandable. The ideal application should be easy to use, robust and well-designed. In the end, the application should meet the requirements as follows:

- To provide a platform where the customer can see the list of cars and choose a car to be rent.
- To let the customer book the car they want and proceed with the payment.
- To let the customer, keep track of their booking and edit the booking information.
- To let the admin, see customer's order and keep track of the orders.
- To let the admin, add, edit, and delete cars' information.
- To let the admin and customer see the booking histories.

In this project, the team decided to use PHP Laravel with MySQL DB as the backend, and Angular Typescript for the frontend. The deployment for both frontend and backend are done with Heroku with ClearDB MySQL (Punch plan) for the production database with a total storage of 1GB.

2 Technical Environment

For the server side, PHP Laravel was chosen not only because one of the backend members has some experiences in using the framework and can teach the other members to understand the framework, but also because Laravel allows the team to make the application well-structured and organize automatically. Laravel also has several useful features such as seeder, factory, feature, and unit testing that are necessary to this project.

For the client side, Angular typescript was chosen because the frontend team members are familiar with the framework, and therefore the frontend team could immediately star designing because no training is needed.

MySQL is used as the database because not only it is easy to use and most members are more familiar with using a relational database, the integration part for MySQL is quite simple. On the server, ClearDB, which is also MySQL is used. Originally, the team uses the free version of ClearDB with 500mb of storage. Turns out the storage is too small, so the team decided to upgrade the DB plan to 1GB for $9.99/month. In the real release version, the storage can be updated as needed.

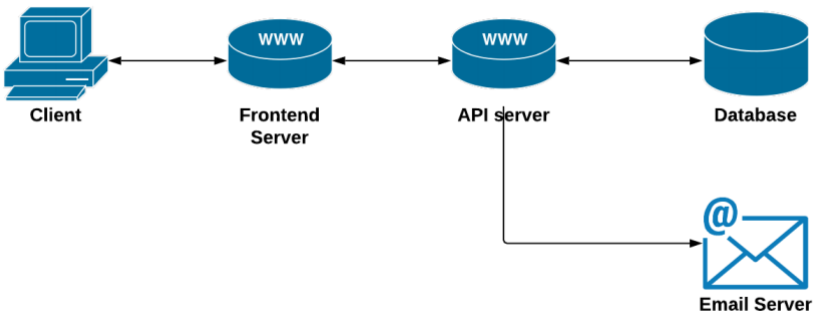
To track the coding the team used GitHub for both frontend and backend. For the backend, there are 2 branches, master and development. All members work on the development branch and the code will be merged once someone has reviewed it.

3 Overall Architecture

The overall architecture can be seen here (the diagram will be too small if we input it in this document).

4 System Architecture

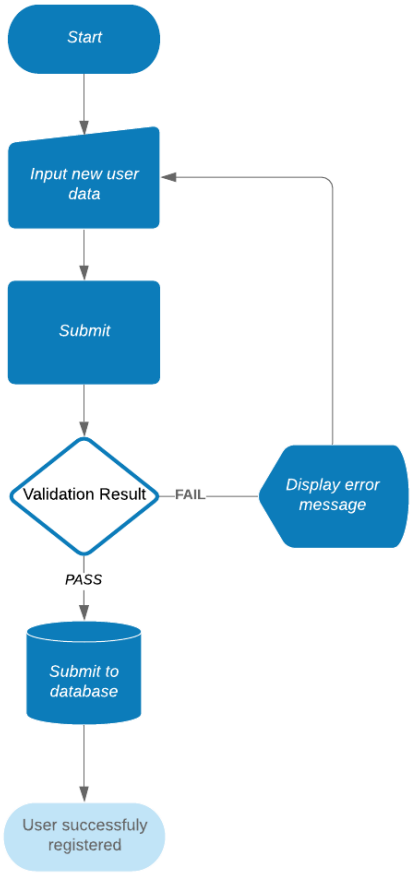
<Detail the system that was built/completed. Explain each component thoroughly. A architecture diagram is essential. >



The system is a relatively straightforward system, where pages are served from a separate server from business logic and database and email capability are on separate services. A frontend server serves application pages to the client, in which pages will make calls to features on a separate API server and receives responses from the same API server. The API server is a server where business logic and server-side validations are performed. This server also stores data and read data from a database. This server also orchestrates email sending functionality request to a separate email server. The email server is a service that sends email with specified template to recipients. This server is connected with the API server to provide the password reset functionality.

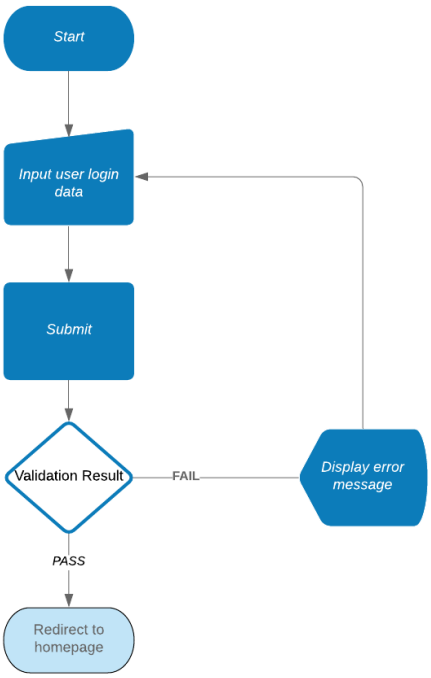
4.1 FUNCTIONALITIES/FEATURES

4.1.1 User Registration



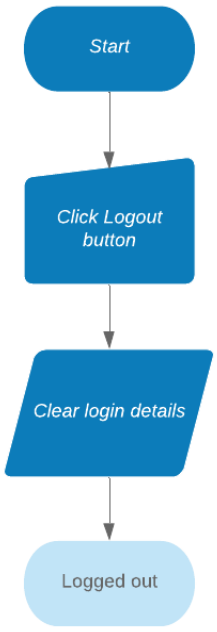
The user registration functionality starts when user is at the register page or when adding new user and inserted the data required by the business logic. After the data input process is done, the entry will be validated on the database for duplicate usernames. If the validation process fails, an error message will be displayed to the user. Otherwise, the new user details will be stored on the database.

4.1.2 *User Login*



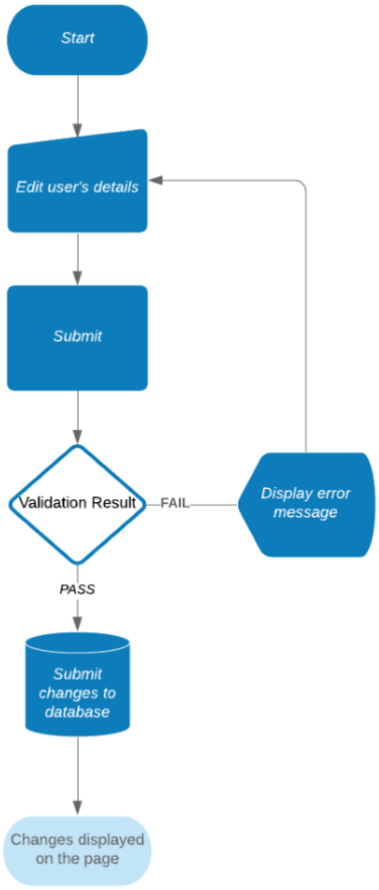
The user login functionality starts when user is at the login page, and inserted the data required by the business logic. After the data input process is done, the entry will be validated on the database for the availability. If the validation process fails, an error message will be displayed to the user. Otherwise, the user will be directed to the main page

4.1.3 *User Logout*



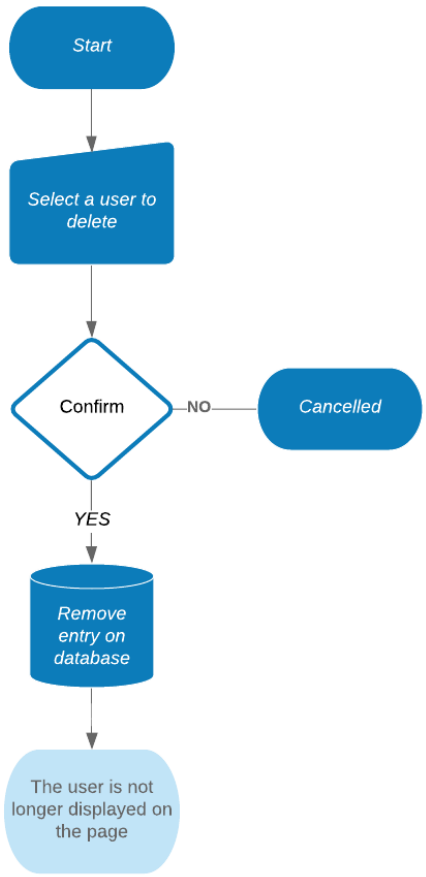
The user logout functionality starts when user is at a login session. When the user click logout, it will removed the login details from session storage and user is successfully logged out.

4.1.4 Edit User's Details (for customer only)



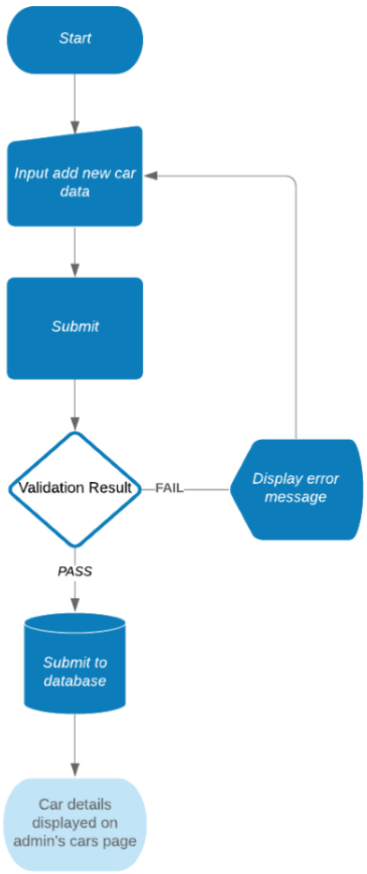
The edit user's details functionality starts when user is logged in and at user details page user inserted the data changes required by the business logic. After the data input process is done, the entry will be validated on the database for accuracy. If the validation process fails, an error message will be displayed to the user. Otherwise, the user's details will be changed

4.1.5 Delete User (for super-admin to admin)



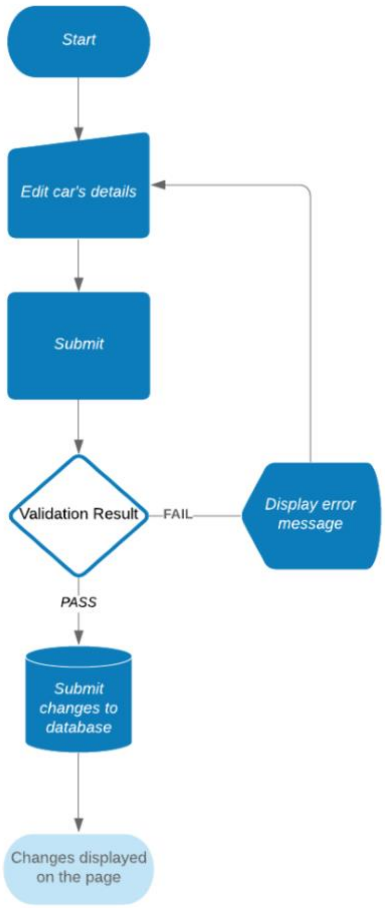
The delete user functionality starts when user is logged in and at user list page user deleted the selected available data. If the user confirms the deletion, the user with their details will be deleted from database and no longer displayed on the page. Otherwise the details will still be there.

4.1.6 Add Car



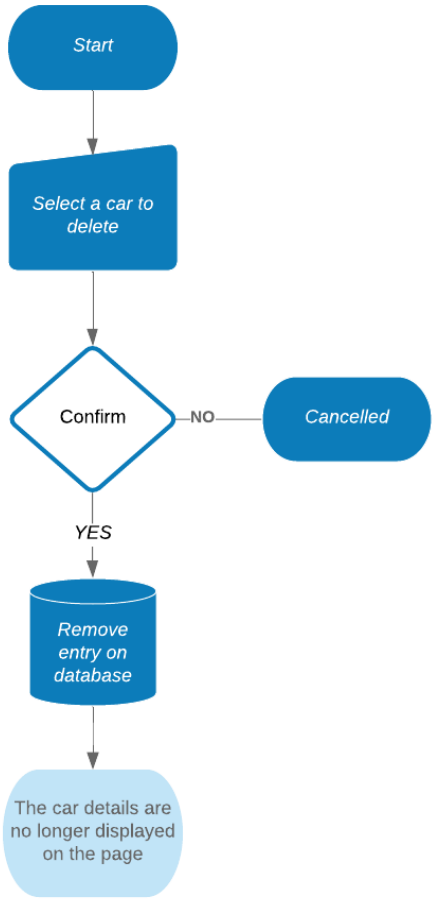
The add car functionality starts when user is on the new car page, in which user will manually input the data required by the database. After required fields are entered, a validation process will take place that will validate input with the business requirements. If the validation process fails, an error message will be displayed to the user. Otherwise, the new car will be stored on the database and displayed on the cars page.

4.1.7 Edit Car's Details



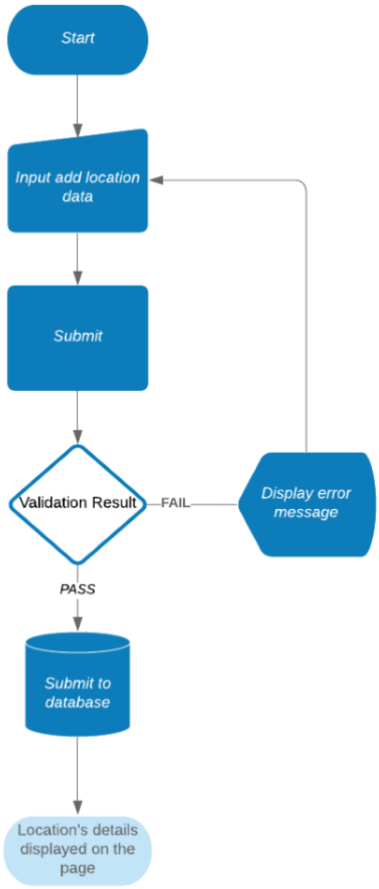
The edit car's details functionality starts when user is logged in and at car details page user inserted the data changes required by the business logic. After the data input process is done, the entry will be validated on the database for accuracy. If the validation process fails, an error message will be displayed to the user. Otherwise, the car's details will be changed.

4.1.8 Delete Car



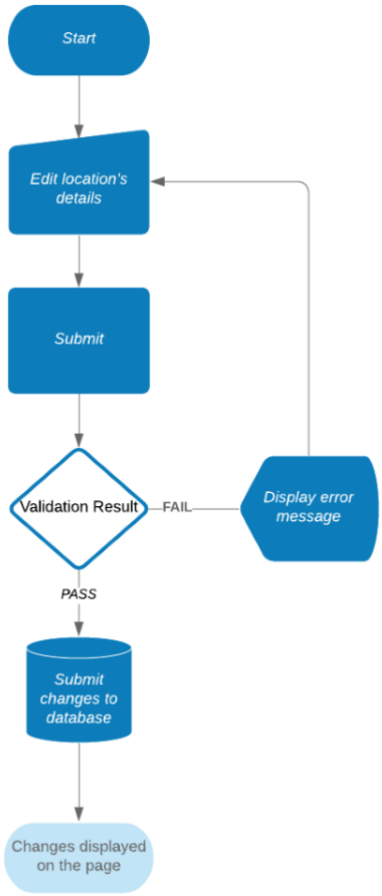
The delete car functionality starts when user is logged in and at car list page user deleted the selected available data. If the user confirms the deletion, the selected car with their details will be deleted from database and no longer displayed on the page. Otherwise the details will still be there.

4.1.9 Add Parking Locations



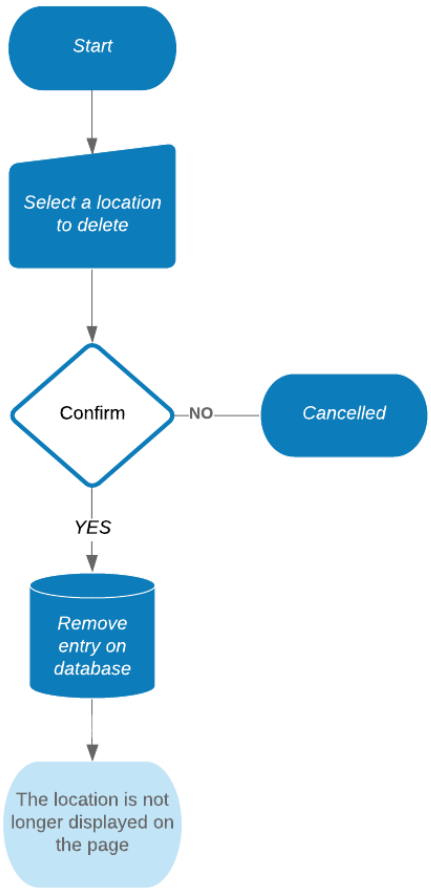
The add location functionality starts when user is on the location page, in which user will manually input the data required by the database. After required fields are entered, a validation process will take place that will validate input with the business requirements. If the validation process fails, an error message will be displayed to the user. Otherwise, the new locations will be stored on the database and displayed on the location page.

4.1.10 Edit Parking Locations



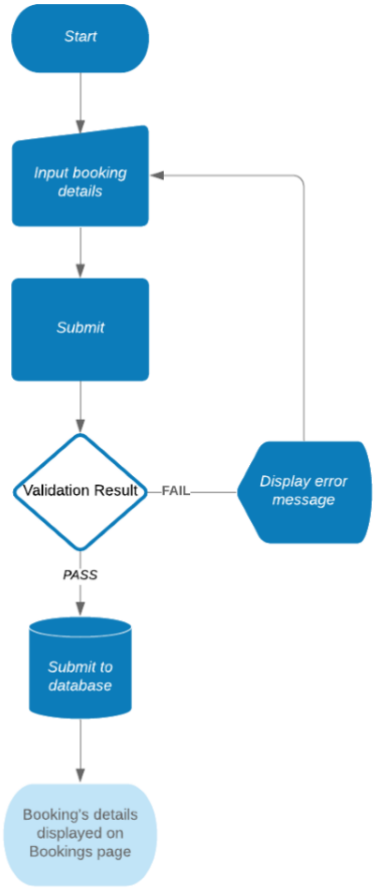
The edit location's details functionality starts when user is logged in and at location details page user inserted the data changes required by the business logic. After the data input process is done, the entry will be validated on the database for accuracy. If the validation process fails, an error message will be displayed to the user. Otherwise, the location's details will be changed.

4.1.11 Delete Parking Locations



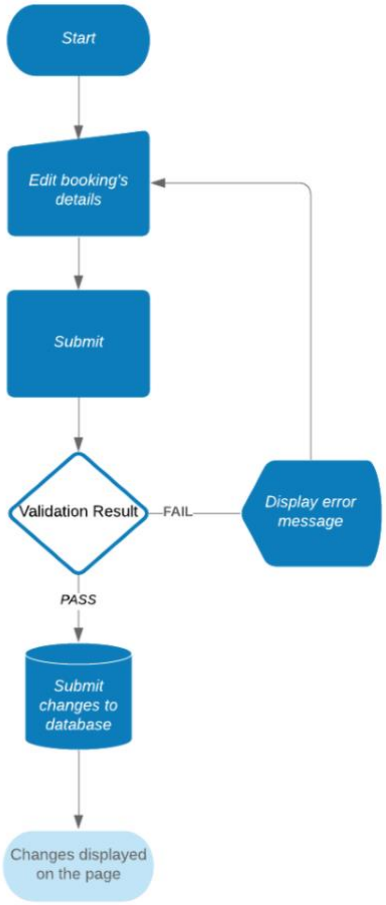
The delete location functionality starts when user is logged in and at location page user deleted the selected available data. If the user confirms the deletion, the selected location with their details will be deleted from database and no longer displayed on the page. Otherwise the details will still be there.

4.1.12 Add Booking for Customer



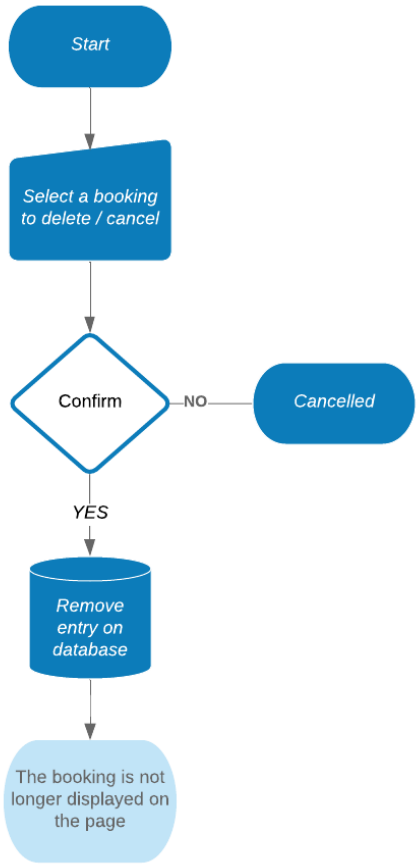
The add booking functionality starts when user is on the car page the click book car, in which user will manually choose the data required by the database. After required data are chosen, a validation process will take place that will validate input with the business requirements. If the validation process fails, an error message will be displayed to the user. Otherwise, the booking data will be stored on the database and displayed on the bookings page.

4.1.13 Edit Booking Details for Customer



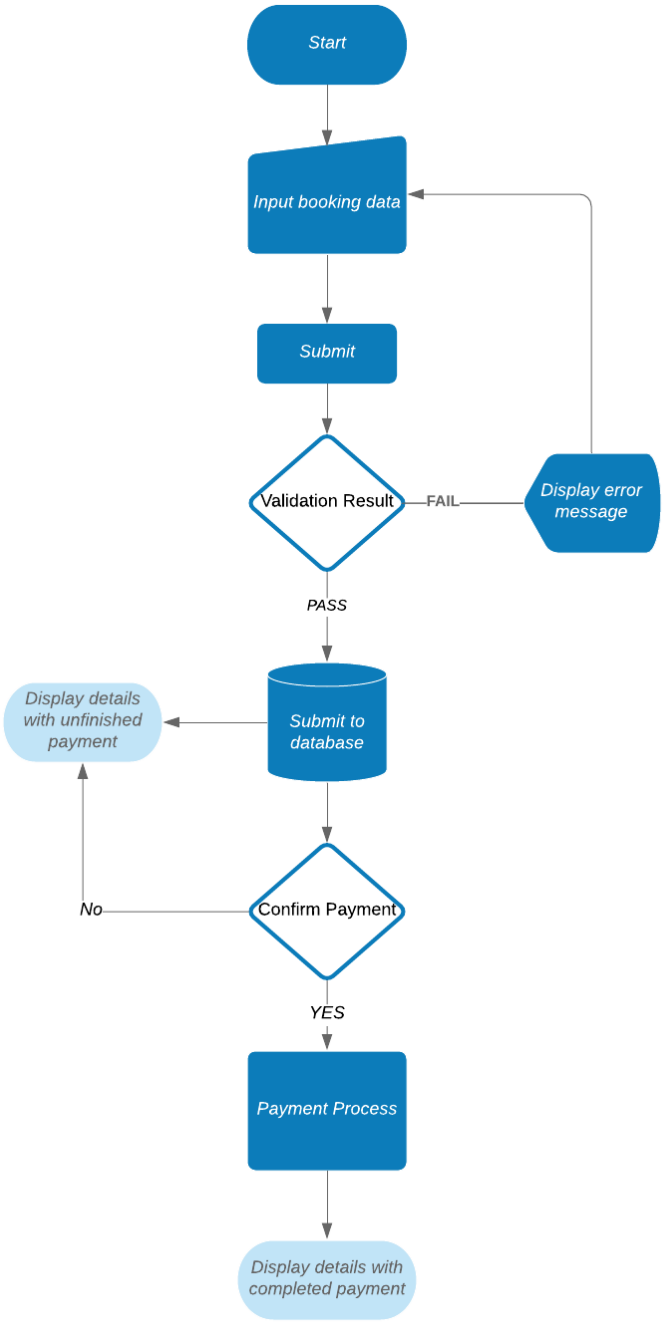
The edit booking's details functionality starts when user is logged in and at booking details page user inserted the data changes required by the business logic. After the data input process is done, the entry will be validated on the database for accuracy. If the validation process fails, an error message will be displayed to the user. Otherwise, the booking's details will be changed.

4.1.14 Delete / Cancel Booking



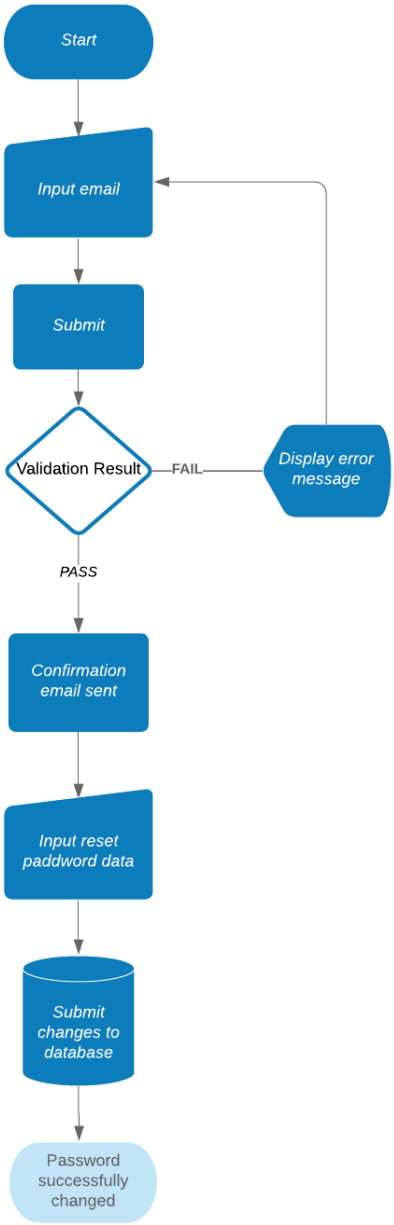
The delete location functionality starts when user is logged in and at location page user deleted the selected available data. If the user confirms the deletion, the selected location with their details will be deleted from database and no longer displayed on the page. Otherwise the details will still be there.

4.1.15 Complete Payment



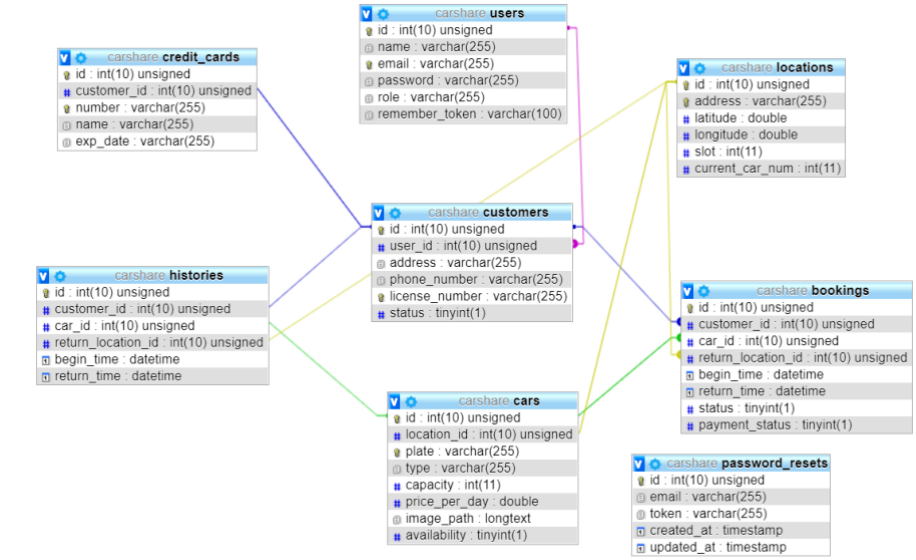
The payment functionality starts when the user is loggeD in and already made or add a booking. After the details are stored in the database, the user was suggested to complete the payment. If the user completes the payment, the booking details will be updated. Otherwise, the payment status will remain undone.

4.1.16 Reset Password



The reset password functionality starts in the reset password page. When the user submitted their registered email, a confirmation will be sent through that email. Otherwise if the email is invalid or unregistered, the confirmation will not be sent. After clicking the confirmation link, user will be redirected to new password page where user will enter their new password. After the new password is submitted, changes will be stored in the database.

5 Database Architecture



Above is the final version of the database. The customers have their own table because customers have information that staffs do not have (such as license number). The credit card number has its own table with customer's id as a foreign key because there might be a case where one customer wants to have several cards in their accounts.

The location has its own table considering one location can have several cars. In the location table, there are slot and current_car_num columns. Their difference is the slot column is the total car the location can fit and current_car num is the number of cars that currently occupying the location, these two columns are used for calculations in CRUD for car and location.

The difference between status and payment_status is the status is the activation for the booking while the payment_status only marks whether the booking is paid or not. When the customer book, their status will be false/inactive until they take the car they've booked from the location. After the customer return the car, their booking will be deleted from booking table and move to the history table.

The password_resets table stores the token that user will get if they click reset password, and the email the token belongs to. In the future, the email column might change to user_id instead so the table has a relationship with others.

The current capacity for the database is 1GB as the system is still in the development phase. In the future, the storage can be improved as needed.

6 Implementation Instructions

The current environment setting for the server is as following:

```
APP_DEBUG: true
APP_ENV: production
APP_KEY: base64:+VZjWa1yBYRBA2Q44MQY/V8n+QCaN8GBDVD2eOdtIiY=
APP_NAME: Laravel
APP_URL: https://powerful-sea-28932.herokuapp.com/
CLEARDB_DATABASE_URL: mysql://b163389afee1db:106d4858@us-cdbr-iron-east-
02.cleardb.net/heroku_1f2fa5e26a8972c?reconnect=true
```

```
DB_CONNECTION: mysql
DB_DATABASE: heroku_1f2fa5e26a8972c
DB_HOST: us-cdbr-iron-east-02.cleardb.net
DB_PASSWORD: 106d4858
DB_PORT: 3306
DB_USERNAME: b163389afee1db
JWT_SECRET: CQXWmkSfas7DOdimCum6byaG02qqgFUcGtbnCM5UA5449k381RZUmJoZ1WfO6MWN
MAIL_DRIVER: smtp
MAIL_ENCRYPTION: null
MAIL_FROM_ADDRESS: vroomapi123@gmail.com
MAIL_FROM_NAME: "Vroom"
MAIL_HOST: smtp.sendgrid.net
MAIL_PASSWORD: vroom2019
MAIL_PORT: MAIL_PORT
MAIL_USERNAME: rorobinbin
```

To create the tables requires by the system in the production environment, we use the '*php artisan migrate*' command from Heroku bash. If in the future a data migration is about to be done, the approach we will do to ensure there is no data loss is:

1. Identify the details of the current and future data environment
2. Make the plan of how to do the migration once full understanding of both environments are reached
3. Backup the data in case of failing in migration
4. Migrate the data
5. Do testings in the new environment, fix any defects found.

7 Non-functional specifications

- Customer must complete all their credentials to register. They also will not be able to book a car if their account is not active.
- All passwords are stored inside the DB in their hashed forms.
- All users must have a JWT token from successful login to book, view history, pay, and access all admin functionalities
- Admin cannot create or delete their own profile. Super admin needs to do this.
- Customer must complete their payment before they can unlock the car
- Customer who edit their profile will be marked as inactive until the admin review and activate their account again.
- Customer will not be able to make a booking if their account has been marked as inactive.
- Customer will not be able to activate their booking (unlock the car) if the payment is incomplete or if it is not the start date yet.
- The link to reset password will be sent to the email the customer register with.

8 Summary of test results

Backend: All results for the unit testing can be seen here
Frontend: Frontend test management document can be seen here

9 Known Issues & Risks

Issues

a. Routes that need authorization token can't be accessed in production environment

Impact: Critical

Allocated to: Sefira

Date raised – closed: 16/8/19 - 16/8/19

Solution:

Put JWT_SECRET key manually in Heroku's environment variable instead of using 'php artisan jwt:secret' command like in local environment

b. Heroku can't store image locally to its server

Impact: High

Allocated to: Sefira, Ignatia, Sarah

Date raised – closed: 05/09/19 - 05/09/19

Solution:
Modify the DB to store the image as base64, convert the image to base64 in the server side, convert back to image in client side

c. Database store coordinate the wrong way

Impact: High

Allocated to: Sefira, Sarah

Date raised – closed: 09/09/19 - 09/09/19

Solution:
Find out the correct way to store the coordinate, modify the DB

d. No price stated for every car

Impact: Critical

Allocated to: Sefira

Date raised – closed: 01/10/19 - 01/10/19

Solution:
Modify the car DB and controller so it has a price column

e. Angular can't accept http response 302 from PayPal

Impact: High

Allocated to: Sefira

Date raised – closed: 10/10/19 - 10/10/19

Solution:
Modify the DB to store the image as base64, convert the image to base64 in the server side, convert back to image in client side

f. Google Geocoding API is a paid feature

Impact: Medium

Allocated to: Sarah

Date raised – closed: 18/09/19 - 25/09/19

Solution:
Used Leaflet to show the map, still went with the Geocoding API because it's a requirement to get the necessary information.

g. Production DB storage is too full, lost storing privilege for a moment

Impact: Critical

Allocated to: Sefira

Date raised – closed: 06/09/19 - 06/09/19

Solution:
Update the DB plan from the free version (500MB) to a paid version (1GB)

h. Sendgrid SMTP account suspended due to username and password of the account in .env file was publicly published on Github

Impact: Medium

Allocated to: Robin

Date raised – closed: 16/8/19 - 16/8/19

Solution:
The repository must be set as private or either need to remove the username and password of the account or remove the .env file.
Then change the password of the account and notice them that all the steps have been fixed.

Risk

a. Customer background check

Impact: High

L'hood: High

Risk found by: Client

Solved by: Ignatia, Supun
Mitigation Strategies
The admin must approve the customer's profile information before they can rent a car
Contingency Plan
The client requests the team to do background check on customers

b. Location capacity

Impact: Low

L'hood: Low

Risk found by: Sefira

Solved by: Sefira

Mitigation Strategies

Do some calculations on the backend. However, there is an unsolvable risk: 'What if the admin adds a new car on a location that only have a slot left, and the next day a car that is being rent will be returned to that location?'

One possible solution that can be done in the future is to use WebSocket to update the capacity in real time. Notify the user if the location suddenly becomes unavailable. Now, the application is incapable of real-time notification.

Contingency Plan

The risk is realized when doing the backend query to add and update car.

c. Credit card encryption

Impact: High

L'hood: High

Risk found by: Sefira

Solved by: n/a

Mitigation Strategies

In the current version of the software, the credit card of the customer is not encrypted inside the DB. In the future, this can be solved by using DES, RSA, or SHA256

Contingency Plan

The risk is realized during sprint 3, but no action was taken because we decided to execute the payment with PayPal instead, although the credit card table still exist, in the current version of the software, we haven't made any payment implementation with credit card.

d. Storing image as base64

Impact: Medium

L'hood: Medium

Risk found by: Sefira

Solved by: n/a

Mitigation Strategies

In the current version of the software, the images are stored as the base64 version. However, if the image has a high resolution, the base64 form of the image will be very long and not only it is a waste of storage, the communication between client and server will take too long.

In the future, the image could be stored using external feature such as Amazon S3.

Contingency Plan
The risk is realized during sprint 2, when the storing car image feature was being worked on.

e. Testing mail reaches limit

Impact: Medium

L'hood: Medium

Risk found by: Supun

Solved by: Supun

Mitigation Strategies
Free version of most SMTP services only allows around 100 emails to be sent per month.
To prevent testing mail reaches limit during testing is to switch from regular SMTP to dummy SMTP which allow 5x times more than regular SMTP.

Contingency Plan
The risk was found during sprint 4 when setting up SMTP for reset password function.

10 Other Considerations

Currently customers are unable to make a payment by using their credit card even though they are required to give their credit card number upon registration. In the future payment with credit card will be implemented so the customer can choose between PayPal and credit card.

Aside from that, the database will be upgraded as needed in the released version as there's only 1GB of storage now. The way of storing image might also change (using amazon S3 is a possibility) because storing image as base64 consumes so much storage.

Security feature will be enhanced as well. In this feature the security is overlooked in order to make the system functional first in the given timeline. In the released version, security such as DES or RSA will be used to encrypt some credentials.

A searching feature to search type of car and separating car based on several categories (such as size and price) will also be taken into consideration.

Future version of the app will work on these considerations in order to make the system more user friendly, reliable, and secure.

11 Appendix

Laravel
https://laravel.com/
Heroku
https://www.heroku.com/
Angular
https://angular.io
Leaflet

https://leafletjs.com/
PayPal PHP SDK
https://paypal.github.io/PayPal-PHP-SDK/
PHPUnit
https://phpunit.de/
SendGrid
https://sendgrid.com/
ClearDB
https://www.cleardb.com/
Postman
https://www.getpostman.com/
JWT
https://jwt.io/
XAMPP
https://www.apachefriends.org/index.html
Creately
https://creately.com/