HIGHER DIPLOMA IN SCIENCE IN COMPUTING

# FINAL PROJECT

# EGarage (Ger's Garage Brief)

Student Name: Jorge Alberto

Student Number: 2019070

Dublin, Ireland 2019

# Contents

Chapter 1 – Introduction ..... 4
A brief synopsis of the project context ..... 4
General areas of computing that project context covers / requires knowledge of ..... 5
Brief summary of your initial proposed plan for addressing the project context ..... 5
Why is this a good project? ..... 6
Novel aspects – a real world business or organization or taking advantage of new technology ..... 7
Chapter 2 – Literature Review ..... 8
Chapter 3 - System Analysis and Design ..... 10
Functional Requirements ..... 10
Use case Diagram ..... 10
Use case description ..... 12
Wireframes ..... 21
Register and Login Wireframes ..... 22
Wireframes for user ..... 23
Wireframes for admin ..... 24
Functional Design ..... 35
Domain Model ..... 35
Sequence Diagram ..... 37
State Machine Diagram ..... 39
Class Diagram ..... 40
Database design (Firestore) ..... 44
Chapter 4 - Implementation of the system ..... 48
How was the process? And What do every functionality need to be finished? ..... 51
Register Functionality / RegisterActivity Class ..... 51
Login Functionality / LoginActivity Class ..... 56
UserHomeActivity Class ..... 57
Make a Booking functionality / MakeABookingActivity Class ..... 60
Change Status and Allocate Mechanic functionalities ..... 66
AdminDisplayBookingsActivity Class ..... 69
Listeners on ListOfBookingsActivity, AllocateMechanicsActivity and DisplayBookingsActivity Class ..... 71
Allocate Cost Functionality ..... 73
Generation PDF Print in ListOfBookingsActivity ..... 74
Print Invoice ..... 75
Chapter 5: Testing and Evaluation ..... 76

Conclusion ..... 82
Appendix A ..... 82
List of References ..... 82

# Chapter 1 – Introduction

## A brief synopsis of the project context

My project is based on the Ger's Garage scenario provided by CCT College. The brief provided asks us to build a Web-based or Mobile application which allow Ger, the owner of the Garage, to automate the processes performed by him. In other words, he needs an application to register the most important activities that he performs daily.

In the following paragraphs I'm going to explain the minimum requirements, which Ger wants to fulfil.

Firstly, cars, motorbikes, small vans and small buses can be booked in for a check-up or service in the garage. He keeps a stock of supplies which could be sold to customers if they needed them to service or fix a vehicle, so he is also able to sell parts or other supplies to customers.

Secondly, he has a small group of mechanics that work with him. There are four mechanics which can carry out any type of service / repair, and each mechanic could either service or repair at most four vehicles in one day.

Thirdly, regarding customers, they should be able to register and login on the application. They also need to be able to book a vehicle to fix or repair by selecting a date, as well as view details about their bookings. To perform these tasks on the application customer must be able to provide personal details and information about the vehicles such as customer name and contact details, vehicle type and make, vehicle license details and vehicle engine type. Customer cannot book on Sunday because the garage is closed.

Fourthly, the different types of bookings are:

- Annual Service,
- Major Service,
- Repair / Fault,
- Major Repair

The type Major repair takes two shifts to repair the vehicle against the other types which take a shift. The application should limit the number of booking allowed per day.

The different statuses of a booking are:

- Booked: this status is set when a booking is made
- In service: When the vehicle arrives at the garage
- Fixed/Completed: When the vehicle is ready for collection
- Collected: When the customer has collected the vehicle and paid the bill
- Unrepairable/Scrapped: when the vehicle cannot be repaired

Lastly, Ger wants to have an admin user to perform some tasks on the application such as view bookings, print schedules for any date to plan his work, allocate a mechanic to a vehicle, allocate costs to a booking, change status of a booking and print an "invoice" to give it to the customers when the service/repair is completed. When Ger allocates a cost to a booking, he needs to be able

to add items/tasks to the final cost because mechanics could need add a part or performed an additional task to fix/service the vehicle. This cost should be used to print the “invoice”.

## General areas of computing that project context covers / requires knowledge of

The project requires knowledge of the application development life cycle to plan, analyze, design, coding, testing and deploy a system. Additionally, each one of these steps also involved some tasks which can be performed using tools and/or diagrams. For example, to analyze, gather customer requirements and design software we can use Use Case Diagram, Sequence Diagrams, Class Diagrams, WireFrames, LucidChart (online tool), etc. Therefore, this project covers all the steps and its tasks that we must do to develop an application, from the beginning when we analyze the requirements until we deploy the application in a store, server, etc.

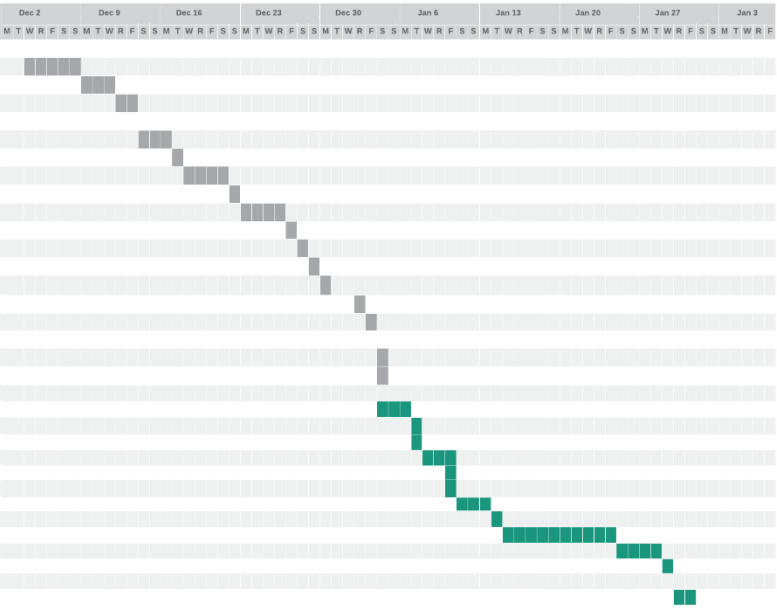
The project also covers aspects related to software development methodologies because although the project must be done individually, we have to define how we are going to do the project. For example, we could analyze, design, coding and testing by iterations, and thus to improve the application by iterations. In my case, first I’m going to analyze and design the minimum requirements. Second, when I complete these stages, I’m going to start to code the solution by functionalities using the information analyzed and designed in the previous step. Third, I’m going to test every functionality when this one is finished. In other words, I’m going to perform unit tests and integration tests among classes which work together. In every step, I’ll work in the documentation. Lastly, If I have time, I’ll add more functionalities or apply techniques of refactoring to improve the coding (repository pattern and MVP architecture). My plan is to produce maintainable software.

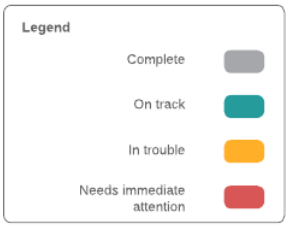
From a technical point of view, this project requires knowledge about:

1. Databases (No-SQL Database, Relational Database, Cloud Databases, Real time databases)
2. Database Management System (MySQL, Firebase Console)
3. Software Architecture (Client – Server) and protocols (HTTP, HTTPs)
4. Libraries and/or Software Development Kit (FireStore SDK)
5. Algorithms, constructs and data types (simple and compound)
6. Programming paradigms (object-oriented, procedural)
7. Programming Languages (Java, PHP, JavaScript)
8. Standard Markup Language (HTML)
9. Style Sheet Language (CSS)
10. Integrated development environment (Visual Studio, Android Studio, NetBeans, Eclipse, etc.)

## Brief summary of your initial proposed plan for addressing the project context

| Task name | Duration | Start date | Finish date |
|---|---|---|---|
| **Research** | 10 | 12/4/2019 | 12/13/2019 |
| Databases (Firebase, SQLite) | 5 | 12/4/2019 | 12/8/2019 |
| Layouts in Android Studio | 3 | 12/9/2019 | 12/11/2019 |
| Software Development Methodologies | 2 | 12/12/2019 | 12/13/2019 |
| **Analisys and Design** | 21 | 12/14/2019 | 1/3/2020 |
| Use cases and its Descriptions | 3 | 12/14/2019 | 12/16/2019 |
| Document User cases and its Descriptions | 1 | 12/17/2019 | 12/17/2019 |
| Secuence diagrams and State diagram | 4 | 12/18/2019 | 2/21/2019 |
| Document Secuence and State diagrams | 1 | 12/22/2019 | 12/22/2019 |
| WireFrames Design | 4 | 12/23/2019 | 2/26/2019 |
| Document WireFrames | 1 | 12/27/2019 | 12/27/2019 |
| Class Diagram | 1 | 12/28/2019 | 12/28/2019 |
| Document Class diagram | 1 | 12/29/2019 | 12/29/2019 |
| Design Data structure of the DB | 1 | 12/30/2019 | 12/30/2019 |
| Document Data structure | 1 | 1/2/2020 | 1/2/2020 |
| Write Chapter 1 | 1 | 1/3/2020 | 1/3/2020 |
| **Installation and Customization** | 1 | 1/4/2020 | 1/4/2020 |
| Installation of Android Studio | 1 | 1/4/2020 | 1/4/2020 |
| Installation and customization of Firebase | 1 | 1/4/2020 | 1/4/2020 |
| **Coding and Testing** | 26 | 1/4/2020 | 1/29/2020 |
| Write code (Model) | 2 | 1/4/2020 | 1/6/2020 |
| Test model with the Database (Unit test) | 1 | 1/7/2020 | 1/7/2020 |
| Document testing and design decisions | 1 | 1/7/2020 | 1/7/2020 |
| Write code (Login and Register screen) | 3 | 1/8/2020 | 1/10/2020 |
| Test Login and Register Screen | 1 | 1/10/2020 | 1/10/2020 |
| Document testing and design decisions | 1 | 1/10/2020 | 1/10/2020 |
| Write code (other Screens) | 4 | 1/11/2020 | 1/13/2020 |
| Prepare presentation | 1 | 1/14/2020 | 1/14/2020 |
| Write code (other Screens) | 6 | 1/15/2020 | 1/24/2020 |
| Test other Screens | 4 | 1/25/2020 | 1/28/2020 |
| Document testing and design decisions | 2 | 1/29/2020 | 1/29/2020 |
| **Documentation** | 2 | 1/30/2020 | 1/31/2020 |
| Finish Documentation | 2 | 1/30/2020 | 1/31/2020 |




The proposed plan was divided in 5 stages:

1. Research: First I did a research about databases (SQLite and Firebase) that I could use to develop the application, Layout of the Activities in Android Studio and software development methodologies
2. Analysis and Design: Analysis of the requirements in the brief and object-oriented design of the software. I'm going to use different diagrams in this phase such as Use case diagram, Sequence diagram, Class diagram, wireframes, etc. I'm also going to design the data structure of the database for a No-SQL database. This structure is based in collections and document, and this one looks a lot like JSON.
3. Installation and Customization: it refers to the software environment used to develop the application. Installation and customization of the Android Studio and Firebase
4. Coding and Testing: Develop of the application and testing of the different components of the system (unit testing and integration testing).
5. Documentation: Complete the documentation

The documentation is performed and updated in every stage.

## Why is this a good project?

I have experience working in the IT industry (mostly in SAP as ABAP Developer) so I have knowledge about different software development life cycle and methodologies. Software development life cycle

for SAP is different to other technologies. Thus, with this project I'm going to learn new tools and diagrams, which are very useful nowadays because software development is not only coding, it is a process which help us to produce better software with quality characteristics such as correctness, reusability, reliability, efficiency and maintainability as goals. From my point of view the latter is the most important nowadays because flexible software allow us to change it easily or add new functionalities without so much efforts. Therefore, building software with a good analysis and design lead us to a high-quality software product.

On the other hand, I'm going to use Android Studio and Firebase to develop my program. These technologies are currently the most used, if we talk about mobile development. Therefore, this add me value because I'm learning technologies which are very important for the mobile development market.

## Novel aspects – a real world business or organization or taking advantage of new technology

The Ger's Garage application is a product for a small business which needs to automate daily activities and register bookings to organize the garage. Additionally, this application will help him to have a good time management. Therefore, this system accomplishes only this purpose. Another thing we could think is about reusability because we could reuse this application and/or source code to implement this product in other garages or in similar businesses.

# Chapter 2 – Literature Review

The technologies that I chose were:

1. Android Studio 3.5.3 (IDE)

2. Software Development Kit (Api Level 28) – Android 9 (Pie)

3. Java Programming Language version 8

4. Cloud Firestore database and Firebase Authentication (Firebase Console)

First, I chose Android Studio because is the standard IDE to develop native mobile application for android. This tool allows us to create screens quickly using its interactive editor and we can use Java programming language with which I am familiar. After finishing project, I can say I didn't have any problem with Android Studio, everything works properly and quickly, even when I installed libraries they were built correctly.

Another option, it could have been developing a hybrid application using PhoneGap, Cordova and another IDE such as Visual Studio, but I preferred to create a native application because of compatibility - I used firebase as database, which is also developed by google, and java programming language. On the other hand, there are also a lot of information, forums, Facebook groups, tutorials about Java and Android Studio on internet and its online community is big which helped me to find solutions to problems I had.

Second, I chose API Level 28 because Android OS 9 (Pie) was launched in August 2018, which is stable after a little more than a year of working on mobile devices, and its framework API related is level 28. If someone has android OS 10, all the APIs until 29 are available.

When we chose android studio to develop native applications, we can develop using java or kotlin. So, I decided to choose Java because we have been working with this programming language a lot this year. I have solid concepts about this programming language, and I like java because it is static typed which allow us to analyse and find out errors quickly as opposite to JavaScript.

Third, I chose Firebase which is a backend-as-a-service app development that provides Realtime database, Firestore database and an authentication service. I used Firestore database because:

1. Compatibility: As I thought Firestore worked with android perfectly.
2. Console user-friendly
3. We can create listener to update screen quickly and without so much effort
4. Queries required are not complex (if we have a query which two or more fields, we must create an index in firestore)
5. Document's fields are indexed automatically by Firebase
6. Atomic operations easy to use (transaction and batch operationes)

Disadvantage which was information repeated in the database and it is so dynamic, we may have two documents at the same collection with different fields.

In conclusion, I'd like to say I'm seeking a job in Ireland and from my point of view these technologies will help me because Java is a stable programming language which is used by a lot of companies, Firebase is a new product which can be used by many programming language (Java

Androd, Java, Kotlin, Swift, JavaScript, Python, etc) and Android Studio is an excellent tool if we want to develop mobile applications.

# Chapter 3 - System Analysis and Design

I'm going to use different diagrams and tools to analyse the requirements and design the software. The core diagrams are UML use case diagrams, UML secuence diagrams and UML class diagrams but I'm going to use others such as domail model, state machine diagrams, etc. The use case diagram to overview usage requirements, secuence diagrams to analyze the behaviour of this application and to discover the objects/entities involved in this system and how they interact. The class diagram helps us to design the structure of the object-oriented software. I'm also going to use Wireframes to define the layouts and content of the screens, and thus discover new functionalities.

## Functional Requirements

### Use case Diagram

The Use Case diagram is an external view of system. It describes the behaviour of the system without reference its internal structural or internal processing.

**Why did I use Use Case diagram?**

We can create the use case to show customers a high-level description of what the system is supposed to do. Its goal is to model the customer requirements in an understandable diagram, and thus can work with clients using diagrams which can be understood by not technical people. We also construct the Use Case diagram to have a better view of the different functionalities of the system and to communicate the scope of a development project.

**How did I construct my use-case diagram?**

1. I defined the actors of the systems. They represent a person, an organization or an external system which interact with our system. In my diagram, I have 3 actors:
    a. User: represent a customer of the Ger's garage
    b. New User: represent a new customer which is used to indicate that a new user must register in the app before using this one. The new user is a user (generalization relationship) and the new user can do whatever a user can do. The generalization relationship is shown as a solid line with the label IS-A.
    c. Admin: This actor represents the owner of the garage
2. I drew a box to put use cases and every element of a use case diagram inside. This box represents the boundary of the system. Then, I added the name Ger's garage at the top center of the box.
3. I identified the use cases for the user and new user. "A use case describes a sequence of actions that provide a measurable value to an actor and is drawn as a horizontal ellipse"(http://www.agiledata.org/essays/objectOrientation101.html#DifferentCoreDiagrams). The use cases "view bookings" and "make a booking" deliver a value added. The first one allows a user to see the status of the bookings which is crucial from a point of view the user. The second one is the most important one because the app is based in the concept of booking, so this use case deliver an added value to the user. Register are login use cases were added because I think they represent important issues in the technical environment. Lastly, I added the binary associations between actors and use cases with a solid line.

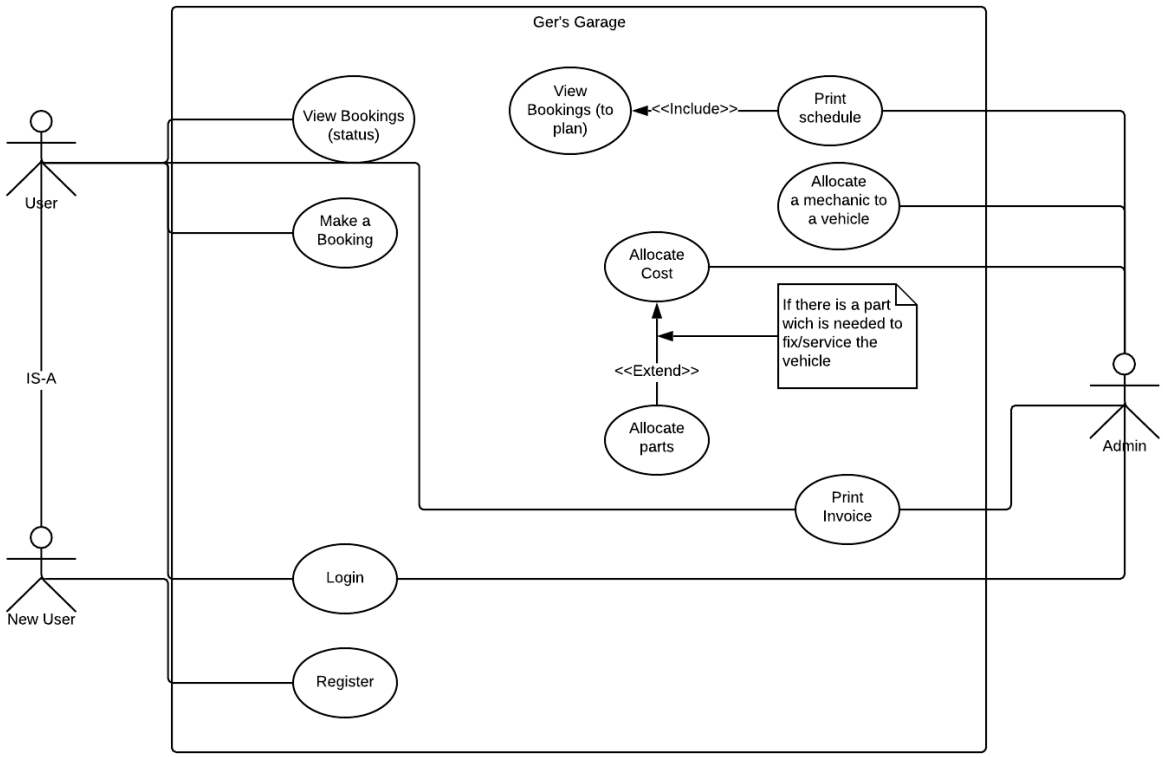
4. I identified the use cases for the admin. Print schedule was added because the admin needs to plan the week, so this use case is of value to the owner. Allocate mechanic to a vehicle is used by the admin in order to fix/service a vehicle, so this use case provides value to the admin. Allocate cost has a business purpose for the admin, because for example the admin may allocate cost to print an invoice or register the cost of a fix/service.
   There is an extending use case which extends Allocate cost. The extending use case is Allocate parts. This extending use cases has a business goals and does have an observable value to the admin because the admin can allocate the cost of items/parts used to fix/service the vehicle. I used the extend relationship because I wanted to specify optional behaviour which can be inserted into the behaviour defined in the extended use case (Allocate cost). This relationship is shown in the diagram as a solid line with the label <<extend>> because the web tool does not allow me to create a dashed line.

   The admin also can log in to the app (this use case was added following the same reason that login and register for user and new user).

   The purpose of the use case print invoice is printing an "invoice" to give customer a receipt of work performed. This is really of value for owner of the garage and the customer. The user is also connected with Print invoice use case because the user actor is involved with the interaction described by the Print invoice use case.

   Lastly, I added the binary associations to the diagram for the admin actor

5. First, this diagram was done in a paper, then was checked and created using the tool offered by https://www.lucidchart.com to build use case diagrams.



## Use case description

Each use case is accompanied with a detailed use case description. It is useful to describe the main flow and alternative flows of a use case, in other words, it is a written description of how users will perform tasks on mobile app. This is built using the use case diagram and the information gathered from brief and enquiries answered at the supervision/guidance classes.

| | |
|---|---|
| **Id** | 1 |
| **Title** | **Login** |
| **Description** | This use case allows the user to log in to the mobile app to access to the different functions according to user's role. **The users have a maximum of 3 attempts to login, after their accounts will be blocked and they must reset their password (optional)** |
| **Primary Actor** | User, New User and Admin |
| **Preconditions** | The device is connected to internet. The user must be registered |
| **Postconditions** | The user is logged in the system and can start to use the functionalities of the system |
| **Main Success Scenario** | 1. The user selects log in<br>2. The user enters an email and a password<br>3. The user submits the email and password<br>4. The app validates the email and password<br>5. The app verifies the email and password<br>6. The user is logged in and the system display the first screen of the app (user/admin home screen)<br>7. Use case ends |
| **Extension** | **Step 4a. Missing email and/or password and email format validation**<br>4.1 The app prompts for email and/or password<br>4.2 The use case resumes at main success scenario step 2<br>**Step 4b. Maximum 3 attempts exceeded (optional)**<br>4.1 The app displays "Maximum attempts exceeded, contact administrator" message<br>4.2 The app locks the user account<br>4.3 The use case ends, it resumes at main success scenario step 7<br>**Step 5. Invalid email and password**<br>5.1 The app displays "Invalid email and/or password" message<br>5.2 The app prompts for email and password<br>5.3 The use case resumes at main success scenario step 2<br>**Step 5. No internet connection** |

| | |
|---|---|
| | 5.1 The app displays “No internet connection” message<br>5.2 The app prompts for email and password<br>5.3 The use case resumes at main success scenario step 5 |
| Priority | High |

| | |
|---|---|
| **Id** | 2 |
| **Title** | **Register** |
| **Description** | This use case allows the user to register in the app to access to the different functions of the app mobile |
| **Primary Actor** | New User and User |
| **Preconditions** | The device is connected to internet. The user must have a valid email |
| **Postconditions** | The user is registered in the app for future use or access. The user can log in to the app |
| **Main Success Scenario** | 1. The user selects Register<br>2. The user entered the following personal details and vehicle details:<br>email, name, mob. phone, vehicle (make and type), vehicle license details, vehicle engine type, password<br>3. The user submits the details<br>4. The app validates the details<br>5. The app verifies the details<br>6. The user is registered successfully and redirect to the user/admin home screen<br>7. The app displays “the user was registered successfully. You will receive an activation email at the email address you entered during registration. Please follow the instructions to activate your account” (optional)<br>8. The user must confirm the email received by the system to log in to the app (optional)<br>9. Use case ends |
| **Extension** | **Step 4. Missing detail/s and data format validation**<br>4.4 The app prompts for invalid data<br>4.5 The use case resumes at main success scenario step 2<br>**Step 5. Invalid email and/or invalid data**<br>5.1 The app displays “The field/s X/s is/are invalid” message<br>5.2 The app prompts for the data<br>5.3 The use case resumes at main success scenario step 2<br>**Step 5. No internet connection**<br>5.1 The app displays “No internet connection” message |

| | |
|---|---|
| | 5.2 The app prompts for the data<br>5.3 The use case resumes at main success scenario step 5 |
| **Priority** | High |

Although every system has the functionality "view X", where X could be an item, booking, etc; I added View Bookings to the use case because the user/customer can see the status of the booking, and this represent an important characteristic of value for the user.

| | |
|---|---|
| **Id** | 3 |
| **Title** | **View Bookings (status)** |
| **Description** | This use case allows the user to see the bookings created and its statuses |
| **Primary Actor** | New User and User |
| **Preconditions** | The user is already registered and logged in to the app. The user must have booked a/ vehicle/s in for a service or a repair in the mobile app previously. The device is connected to internet |
| **Postconditions** | The user can see the status and the details of the bookings |
| **Main Success Scenario** | 1. The user selects View bookings<br>2. The app displays the bookings selected belonging to that user<br>3. The user can see the details of every booking<br>4. Use case ends |
| **Extension** | **Step 4. No internet connection**<br>2.1 The app displays "No internet connection" message<br>2.2 The use case resumes at main success scenario step 4 |
| **Priority** | High |

The next user case descriptions will be checked after wireframe design, because we will have more information about what the user sees when is using the system.

| | |
|---|---|
| **Id** | 4 |
| **Title** | **Make a Booking** |
| **Description** | This use case allows the user to book his vehicles in for a check-up or service |
| **Primary Actor** | New User and User |
| **Preconditions** | The user is logged in to the app. The user has already registered a vehicle in the app. The device is connected to internet |
| **Postconditions** | The user has booked a vehicle to fix/service and the status of the booking is Booked |
| **Main Success Scenario** | 1. The user selects Make a booking<br>2. The app displays the form for booking<br>3. The user selects the vehicle to fix/service and the type of booking |

| | |
|---|---|
| | 4. The user selects the date for the service<br>5. The app verifies if there is available space for that day<br>6. The app displays the available space for that day<br>7. The user select an available space<br>8. The user submits the booking<br>9. The user confirms the booking and add comments to this booking<br>10. The apps verifies the booking details<br>11. The app creates and saves a booking with the data entered by the user<br>12. User case ends |
| **Extension** | **Step 5, 10 and 11. No internet connection**<br>1.1 The app displays “No internet connection” message<br>1.2 The use case resumes at main success scenario step 5, 10 or 11 respectively<br>**Step 5. No space available for that day**<br>5.1 The app displays “No space available for DATE (DD/MM/YY)” message<br>5.2 The use case resumes at main success scenario step 4<br>**Step 9. The user cancels the booking**<br>9.1 The app displays “The booking was cancelled” message<br>9.2 The use case resumes at main success scenario step 12<br>**Step 10. The booking details are incorrect, or the available booking is not available anymore**<br>10.1 The app displays “Mechanic not available anymore” message<br>10.2 The use case resumes at main success scenario step 2 |
| **Priority** | High |

This use case (View Bookings) is another functionality which is of value for the admin because they can see the different booking, plan the day/week and print the schedule for any particular date.

| | |
|---|---|
| **Id** | 5 |
| **Title** | **View Bookings** |
| **Description** | This use case allows the admin to see the booking that were made by users |
| **Primary Actor** | Admin |
| **Preconditions** | The admin is logged in to the app. The device is connected to internet |
| **Postconditions** | The admin can see all details of each booking |
| **Main Success Scenario** | 1. The admin selects View bookings<br>2. The app displays options to select a booking (date, range of dates, id of booking, email) |

| | |
|---|---|
| | 3. The admin enters a date, range of dates, id or email to search the booking and submit the values<br>4. The app displays the bookings selected<br>5. The admin can see the details of every booking<br>6. Use case ends |
| **Extension** | **Step 3, 4 and 5. No internet connection**<br>3.1 The app displays “No internet connection” message<br>3.2 The use case resumes at main success scenario step 3, 4 or 5 respectively<br>**Step 4. No bookings were selected**<br>4.1 The app displays “No bookings were selected” message<br>4.2 The use case resumes at main success scenario step 2 |
| **Priority** | High |

| | |
|---|---|
| **Id** | 6 |
| **Title** | **Allocate a mechanic to a vehicle** |
| **Description** | This use case allows the admin to allocate an available mechanic to carry out a service/repair of a vehicle |
| **Primary Actor** | Admin |
| **Preconditions** | The admin is logged in to the app. The device is connected to internet. The status of the booking is “In Service” |
| **Postconditions** | The admin allocates a mechanic to a vehicle which is associated with a booking “In service” |
| **Main Success Scenario** | 1. The admin selects Allocate mechanic<br>2. The app displays options to select a booking (date, range of dates, id of booking, email)<br>3. The admin enters a date, range of dates, id or email to search the booking<br>4. The app retrieves the bookings with status “In Service” or “Booked”<br>5. The app displays the bookings selected<br>6. The admin selects a booking<br>7. The admin allocates a mechanic to that vehicle<br>8. The admin submit the allocation<br>9. The admin confirms the allocation<br>10. The app verifies the mechanic selected<br>11. The app saves the allocation<br>12. User case ends |
| **Extension** | **Step 4, 10. No internet connection**<br>4.1 The app displays “No internet connection” message |

| | |
|---|---|
| | 4.2 The use case resumes at main success scenario step 4 and 11 respectively<br>**Step 4. No bookings "In service" were retrieved**<br>4.1 The app displays "No Booking /"In service" were selected" message<br>4.2 The use case resumes at main success scenario step 2<br>**Step 10. The mechanic is not available anymore for that booking**<br>11.1 The app displays "Mechanic not available anymore" message<br>11.2 The use case resumes at main success scenario step 7<br>**Step 9. The admin cancels the allocation**<br>10.1 The app displays "The allocation was cancelled" message<br>10.2 The use case resumes at main success scenario step 13 |
| **Priority** | High |

**Step number 10. The app verifies the mechanic selected**: We already knew before starting this use case that there are mechanics available for every booking on the date selected at step 3. When a user makes a booking, the app checks if there is available space. Anyway, double checked is performed to avoid another user choose same mechanic (two admins) or not two admin are allowed in that screen.

| | |
|---|---|
| **Id** | 7 |
| **Title** | **Print Schedule** |
| **Description** | This use case allows the admin to print the schedule of bookings for any particular date or a range of dates |
| **Primary Actor** | Admin |
| **Preconditions** | The admin is logged in to the app. The device is connected to internet. There is a printer connected to the same network as the device |
| **Postconditions** | The admin prints the schedule of bookings |
| **Include use cases** | **View Bookings (5)** |
| **Main Success Scenario** | 1. The admin selects View Bookings<br>2. The app triggers View Bookings use case<br>3. The admin selects bookings to print and select Print schedule<br>4. The app displays a pdf with all the bookings to print and allow user to select a printer if this one is detected in the net<br>5. The admin selects a printer and submit the printing<br>6. The admin confirms the printing<br>7. The app sends the data (schedule) to the printer |

| | |
|---|---|
| | 8. The printer prints the schedule<br>9. Use case ends |
| **Extension** | **Step 4 and 7. No internet connection**<br>4.1 The app displays “No internet connection” message<br>4.2 The use case resumes at main success scenario step 4 or 7 respectively<br>**Step 6. The admin cancels printing**<br>6.1 The app displays “Print cancelled” message<br>6.2 The use case resumes at main success scenario step 9 |
| **Priority** | High |

| | |
|---|---|
| **Id** | 8 |
| **Title** | **Allocate cost** |
| **Description** | This use case allows the admin to allocate costs to each type of booking (Annual Service; Major Service; Repair / Fault or Major Repair) |
| **Primary Actor** | Admin |
| **Preconditions** | The admin is logged in to the app. The device is connected to internet. The status of the booking is “Fixed / Completed” |
| **Postconditions** | The costs of the service/fix are allocated |
| **Main Success Scenario** | 1. The admin selects Allocate costs<br>2. The app displays options to select a booking (date, range of dates, id of booking, email)<br>3. The admin enters a date, range of dates, id or email to search the booking<br>4. The app retrieves the bookings with status “Fixed / Completed”<br>5. The app displays the booking/s retrieved<br>6. The admin selects a booking<br>7. The admin enters the cost and the name of service/fix performed (it could be fixed)<br>8. (Extension Point: Allocate parts)<br>9. The admin submit the cost<br>10. The app validates the cost<br>11. The admin confirm the allocation<br>12. The app saves the cost<br>13. User case ends |
| **Extension** | **Step 4 and 12. No internet connection**<br>4.1 The app displays “No internet connection” message<br>4.2 The use case resumes at main success scenario step 4 or 12 respectively<br>**Step 4. No bookings “Fixed / Completed” were selected**<br>4.1 The app displays “No Booking / “Fixed / Completed” were selected” message |

| | |
|---|---|
| | 4.2 The use case resumes at main success scenario step 2<br>**Step 10. Invalid data**<br>10.1 The app displays “Data invalid” message<br>10.2 The use case resumes at main success scenario step 7<br>**Step 11. The admin cancels the allocation**<br>11.1 The app displays “The allocation was cancelled” message<br>11.2 The use case resumes at main success scenario step 13 |
| **Priority** | High |

| | |
|---|---|
| **Id** | 9 |
| **Title** | **Allocate parts** |
| **Description** | This use case allows the admin to allocate cost of the parts/items the mechanic used to fix/service the vehicle |
| **Primary Actor** | Admin |
| **Preconditions** | The admin is logged in to the app. The device is connected to internet. The status of the booking is “Fixed / Completed” |
| **Postconditions** | The costs of parts/items that were needed to service/fix the vehicle are allocated |
| **Extends use cases** | **Allocate cost** |
| **Main Success Scenario** | 1. The admin selects Allocate parts<br>2. The app displays the form to fill the parts, extra work required and the cost.<br>3. The admin enters the name of a part of a vehicle (or selects a predefined item) or a task that mechanic performs to fix/service a vehicle, and a cost associated to the item or task<br>4. Repeat step 3 for all parts/items<br>5. The app validates the data entered by the user<br>6. The admin confirms the allocations<br>7. The app saves the costs<br>8. User case ends |
| **Extension** | **Step 7. No internet connection**<br>7.1 The app displays “No internet connection” message<br>7.2 The use case resumes at main success scenario step 7<br>**Step 5. Invalid data**<br>5.1 The app displays “Data invalid” message<br>5.2 The use case resumes at main success scenario step 3<br>**Step 6. The admin cancels the allocation**<br>6.1 The app displays “The allocation was cancelled” message |

| | |
|---|---|
| | 6.2 The use case resumes at main success scenario step 8 |
| **Priority** | High |

| | |
|---|---|
| **Id** | 10 |
| **Title** | **Print Invoice** |
| **Description** | This use case allows the admin to print an invoice when the service/repair is complete. The invoice prints all the costs included |
| **Primary Actor** | Admin |
| **Preconditions** | The admin is logged in to the app. The device is connected to internet. There is a printer connected to the same network as the device. The status of the booking is “Fixed / Completed” |
| **Postconditions** | The admin prints an invoice. |
| **Main Success Scenario** | 1. The admin selects Print Invoice<br>2. The app displays options to select a booking (date, range of dates, id of booking, email)<br>3. The admin enters a date, ranges of dates, id or email to search the booking<br>4. The app retrieves the bookings with status “Fixed / Completed”<br>5. The app displays the booking/s retrieved<br>6. The admin select a booking<br>7. The admin select print invoice<br>8. The app generate a pdf and allow user to select a printer if this one is detected in the net<br>9. The admin select a printer and submit the printing<br>10. The admin confirms the printing<br>11. The app sends the data (invoice) to the printer<br>12. The printer prints the invoice/bills for the customer/user<br>13. Use case ends |
| **Extension** | **Step 4, 8 and 11. No internet connection**<br>4.1 The app displays “No internet connection” message<br>4.2 The use case resumes at main success scenario step 4,8 or 11 respectively<br>**Step 4. No bookings “Fixed / Completed” were retrieved**<br>4.1 The app displays “No Booking / “Fixed / Completed” were selected” message<br>4.2 The use case resumes at main success scenario step 2<br>**Step 10. The admin cancels printing**<br>10.1 The app displays “Print cancelled” message |

| | |
|---|---|
| | 10.2 The use case resumes at main success scenario step 13 |
| **Priority** | High |

These descriptions could be slightly different when we create the wireframes because we can group different functionalities in a screen.

## Wireframes

A wireframe is used to show the structure of the screens of an application, its content (button, panel, etc) and how they interact each other. In other words, it is a graphical representation of an application with the most essential elements and the content of a screen. It is useful to work with the users of the app and receive feedback from them. The users, when see a wireframe, they put more focus on the functionality and user experience than the aesthetic.

How did I construct my wireframe design?

First, I think someone who works in a garage should work with a screen which is very easy to use, user friendly and predictable.

Second, the interface must be consistent, i.e., the distribution of the information and content must be similar among the screens and we must use a familiar navigational process. I used the android apps as examples to follow my designs.

Third, I use the desktop application Balsamiq Mockups (version 3.5.17) to wireframing the screens and I took a look of some examples of wireframes on the web.

Fourth, I designed the graphical interfaces using the information gathering from the supervision/guided classes and Ger's garage brief, use case diagram and its description.

Lastly, I created the most important elements in the screen which always are and are really important from my point of view. The question to describe this, would be What does the user expect to see on certain areas of the screen? Then I filled the space with details, as for example button to come back to another screen. I also defined the interactions among the screens.

Why did you choose this wireframe design?

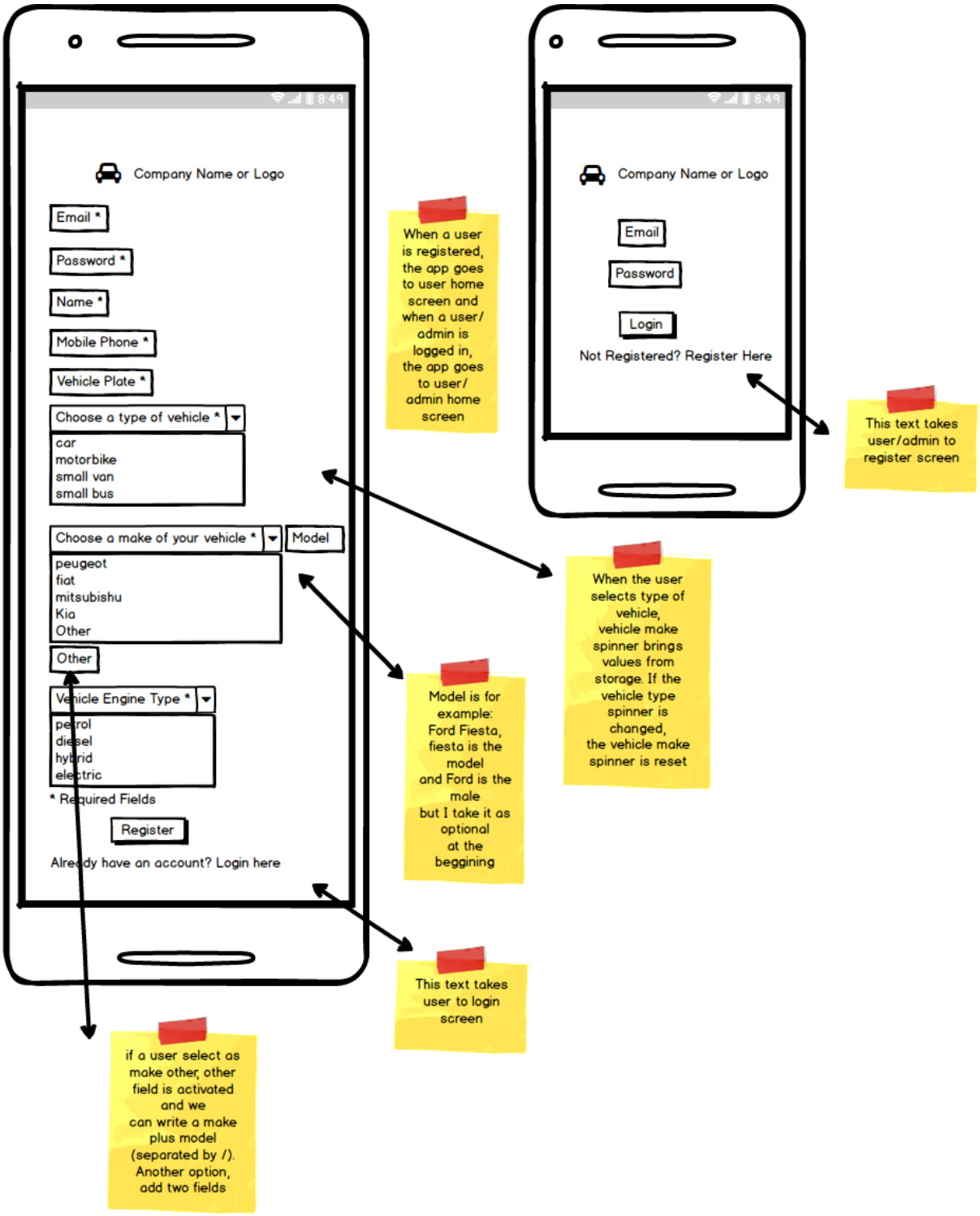
I choose wireframe because:

1. this tool helps us to identify new functionalities which could have forget when we do the use case diagram or requirements gathering,
2. this tool helps us to identify new fields and its data types on the screen before creating the class diagram and the structure of the database,
3. We can draw the structure of the screens and its content quickly, and where we are coding, we already have everything defined in our previous steps such as the elements that we will require, the layout for different kind of screens
4. We work on the interactions and layout needs in the early stage of the development process
5. This tool helps us to get detailed and validate the requirements of the software

There is a map in which we can see all wireframes and how they are connected. This one is saved in the map folder.

## Register and Login Wireframes

**Register and Login screens for user/new user and admin (new user is an abstraction of user)**

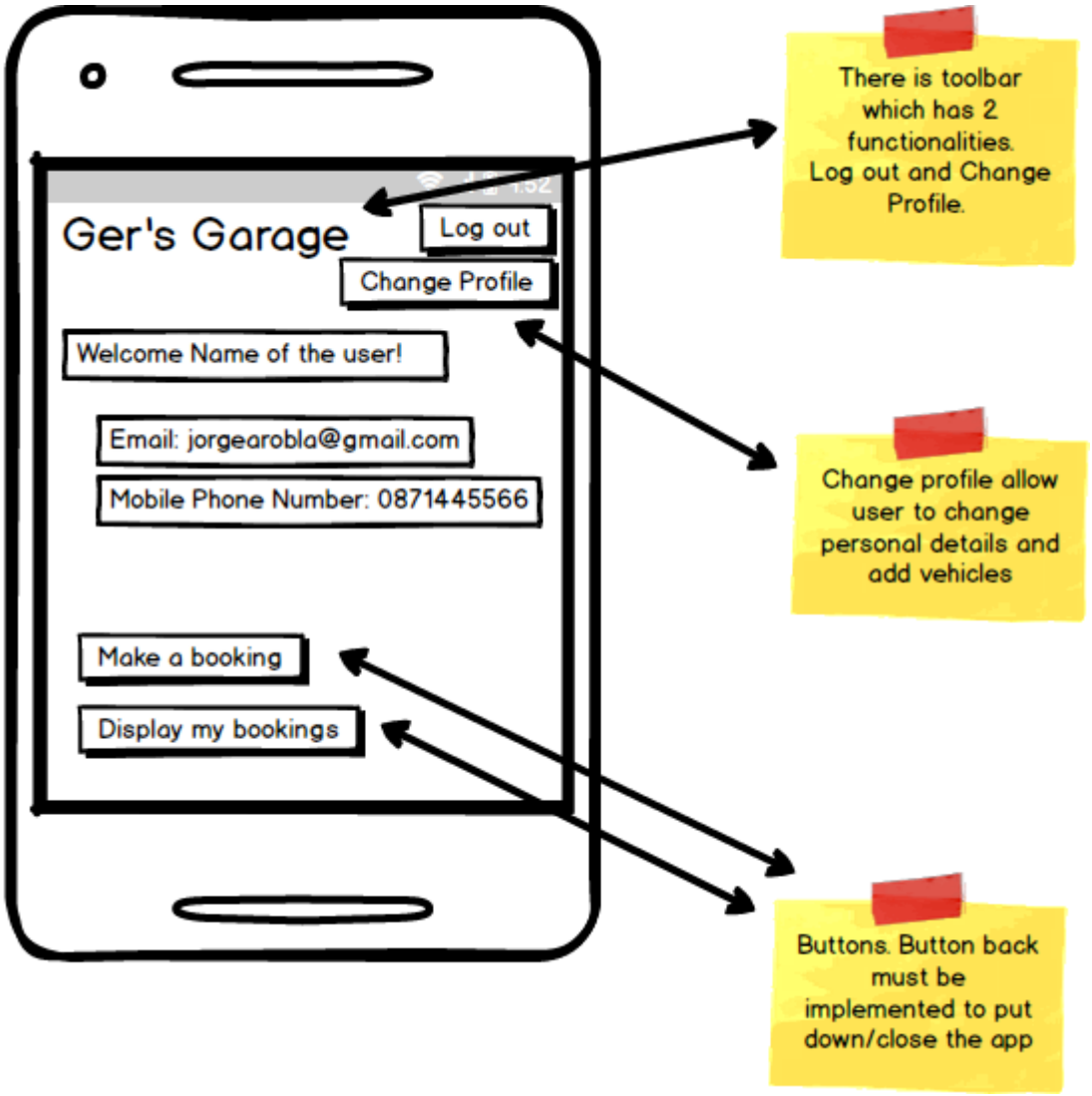



I must add Forgot Username or password button: this functionality allows a user to recover the account if the user does not remember the details to log in to the app. (priority high)

Register and Login screens must have a progress bar to avoid user to continue using application after user clicks on the register/login button.

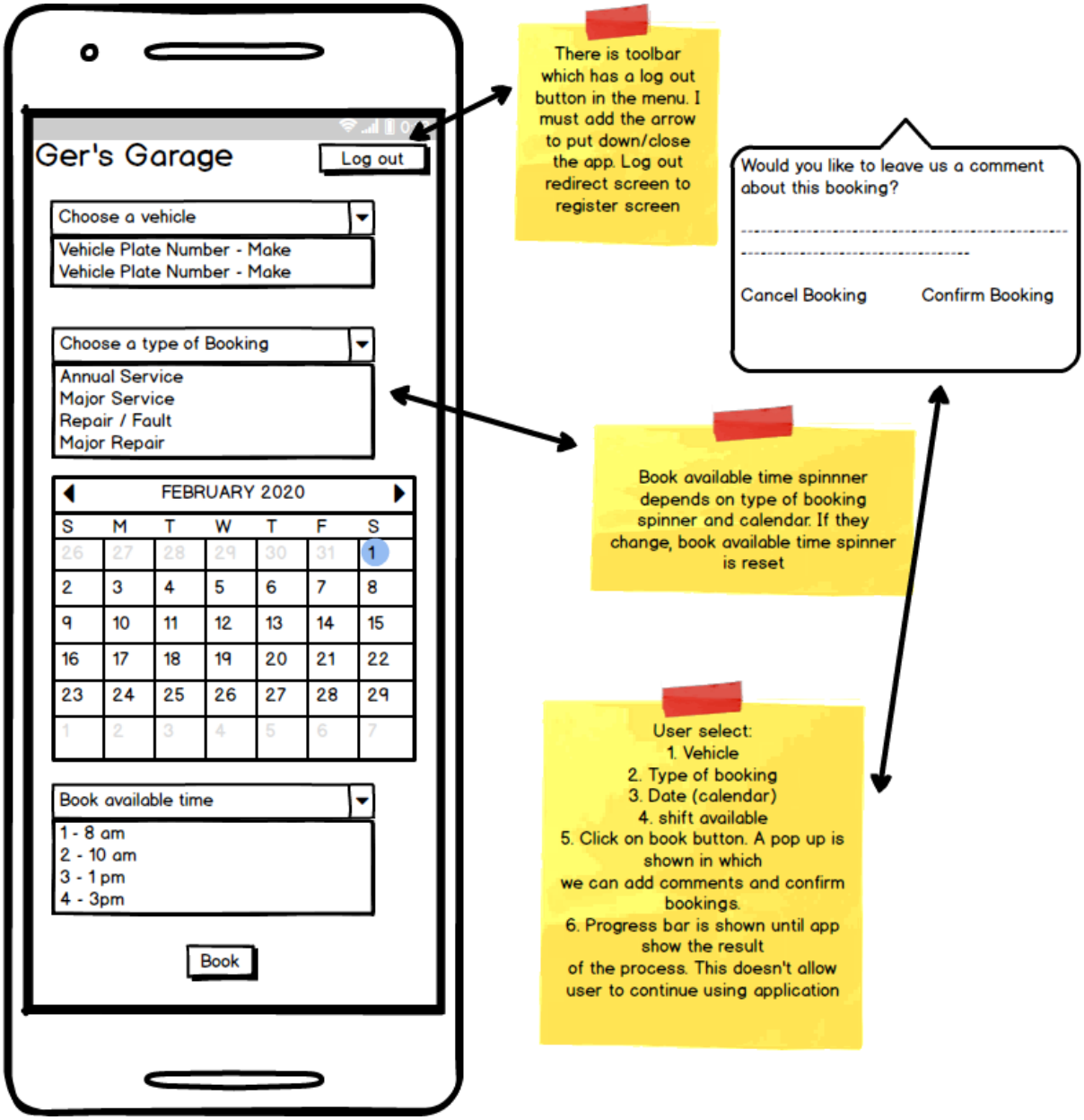
## Wireframes for user

**Home user screen**




New functionality: Change profile: it is a new functionality which allow a user to change the personal details and add/modify a vehicle. (priority medium)

**Make a booking screen**



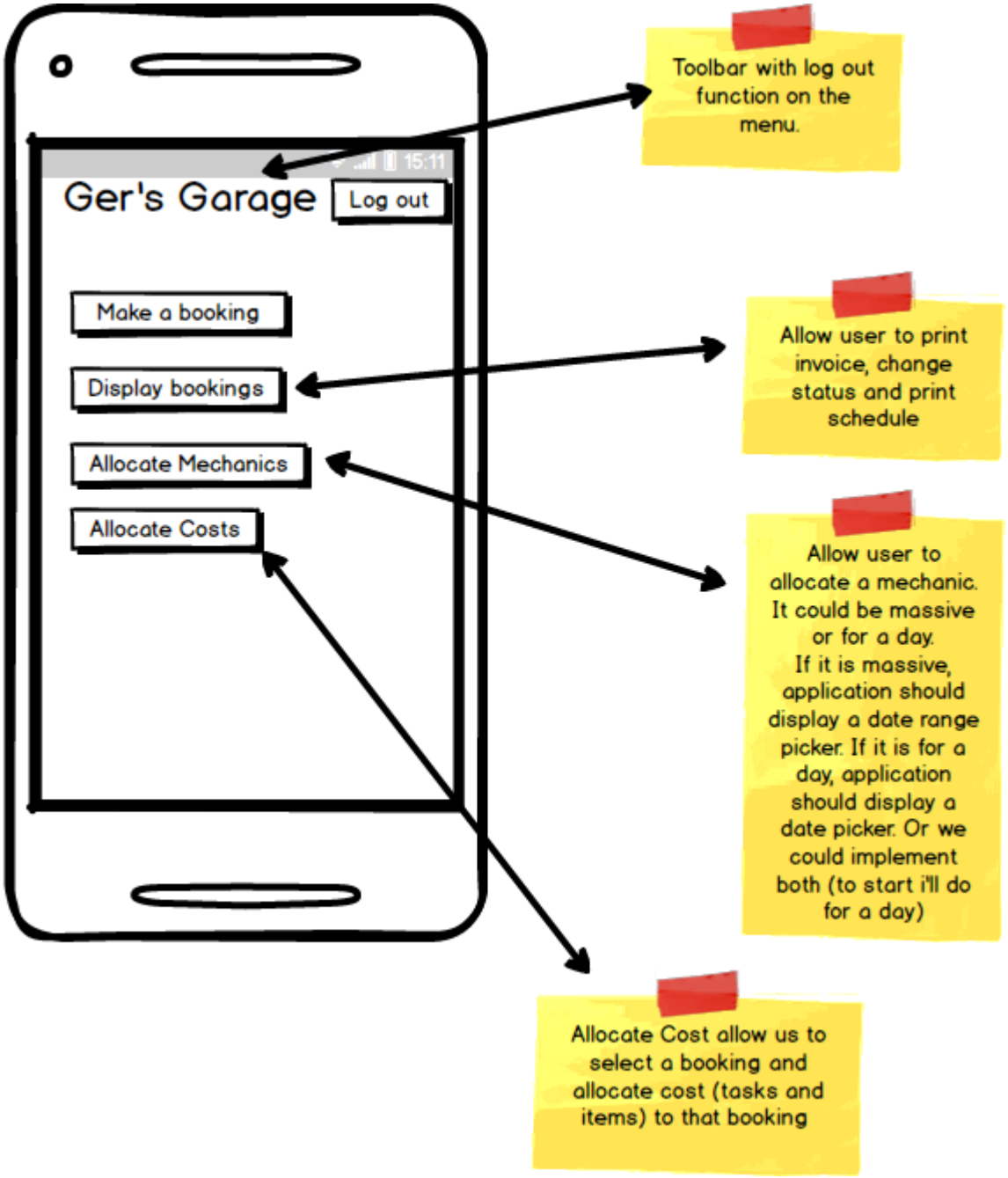

**Display booking screen** must show important information for user: For example

1. Number booking
2. Date of booking and shift (time)
3. Type of booking
4. Vehicle (number plate and make)

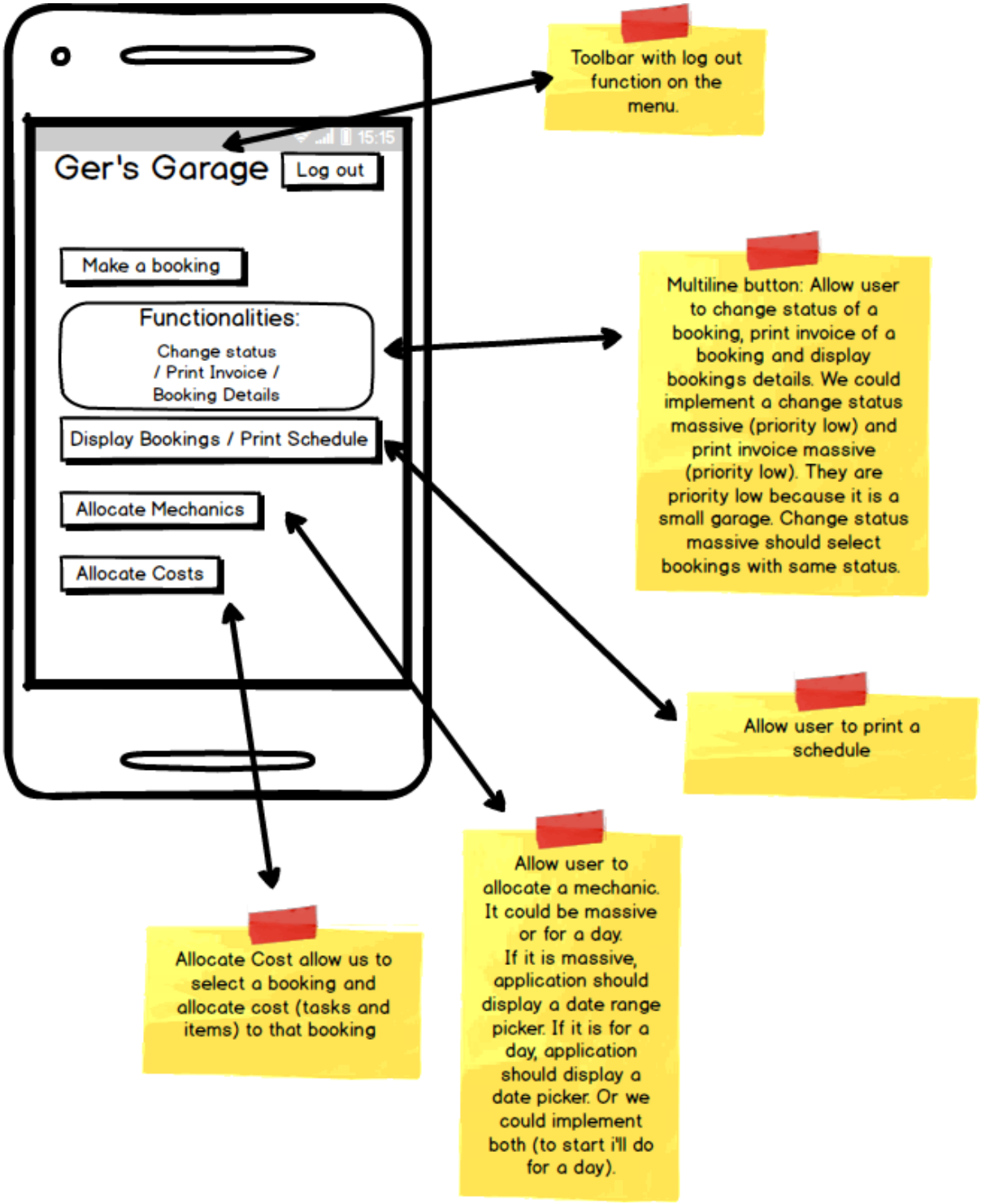
This screen is accessed from user home screen.

## Wireframes for admin

**Admin Home Screen (Option 1)**




**Admin Home Screen (Option 2)**



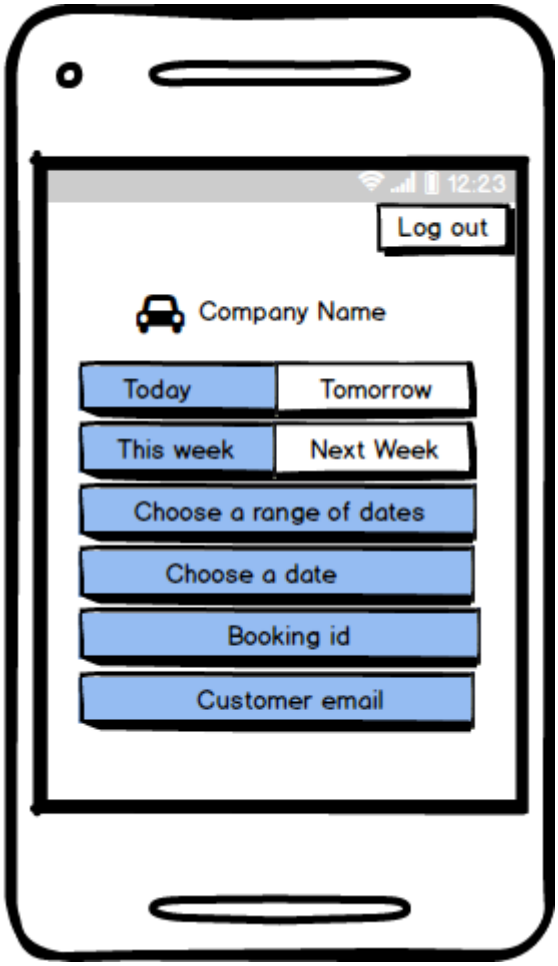

New Functionality: Make a booking: The admin could make a booking in order to extend another booking because they need more time to fix/service the vehicle, or just because they want to make a booking for someone. (priority low)

Four important things here:

1. If user wants to allocate Mechanic, booking must be in "in service" or "booked" status
2. If user wants to allocate cost, booking must be in “fixed” status
3. If user wants to print an invoice, booking must be in “fixed” status

4. We can implement a method to change status of a booking (from collected to fixed or unrepairable to fixed) in case user forget to print invoice or allocate cost or car was put in unrepairable status by mistake

From **Display bookings and Allocate cost buttons** (admin home screen option 1) **or Change Status / Print invoice / Booking details, Display bookings / Print schedule and Allocate cost buttons** (admin home screen option 2) go to the following screen called **Display Booking screen**:

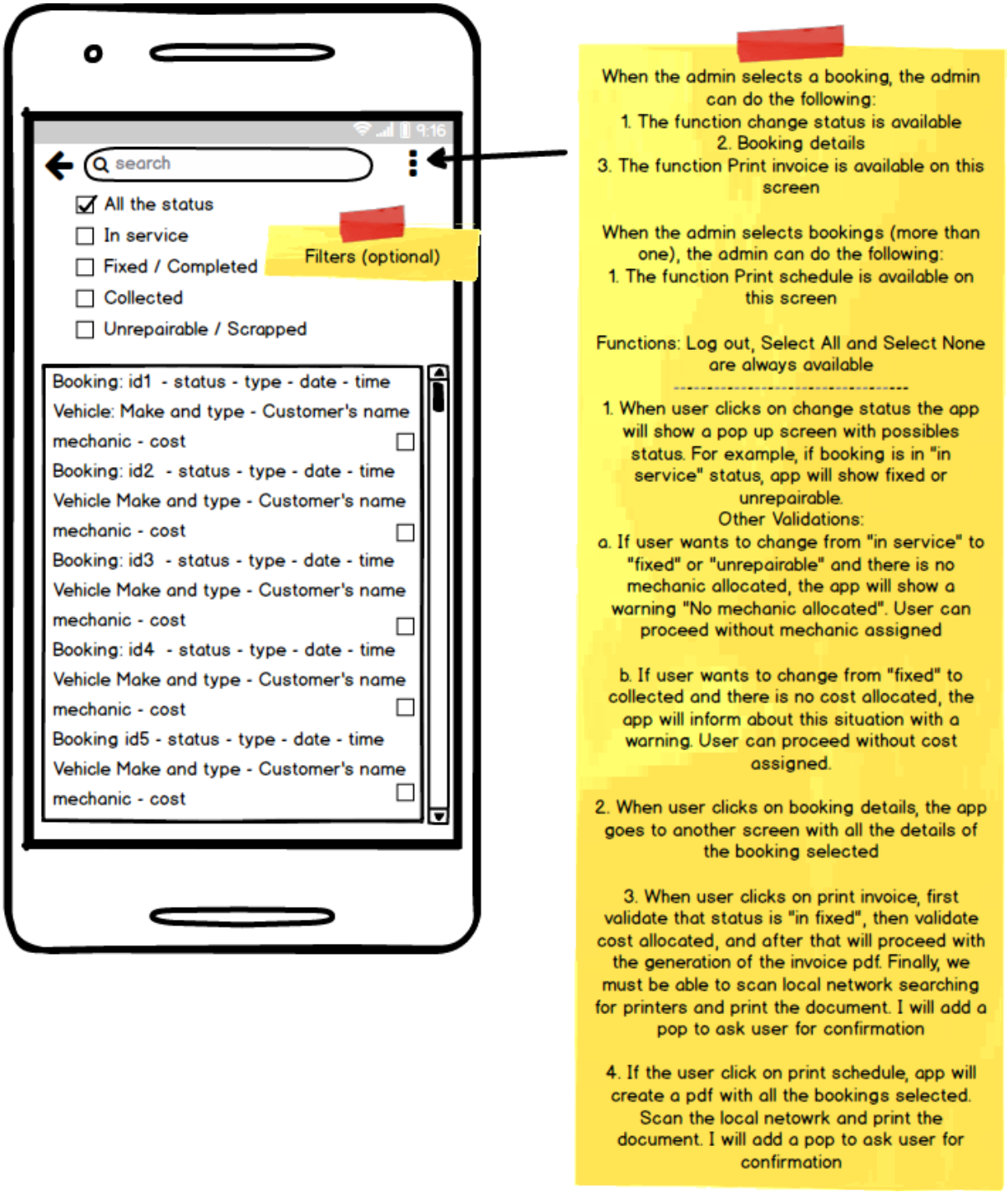



In this screen must be added two input fields to enter an id of a booking and/or an email to search bookings or pop ups in which we can type ids or emails. We must add button to go back to previous screen.

**List of Bookings screen**

I'm going to show screens for admin home functionalities (option 1 and 2).

**Option 1**: This screen has all the functionalities together (**Print Invoice, change status, Bookings details and print schedule)** when we click on Display Bookings button.




New functionality:

1. Change status of the booking: The user must be allowed to change the status of a booking. We have 5 status:

   1) Booked – this is the default status when a booking is made

   2) In Service – when the vehicle arrives at the garage

   3) Fixed / Completed – when the vehicle is ready for collection

   4) Collected – When the customer has taken the vehicle away and paid their bill

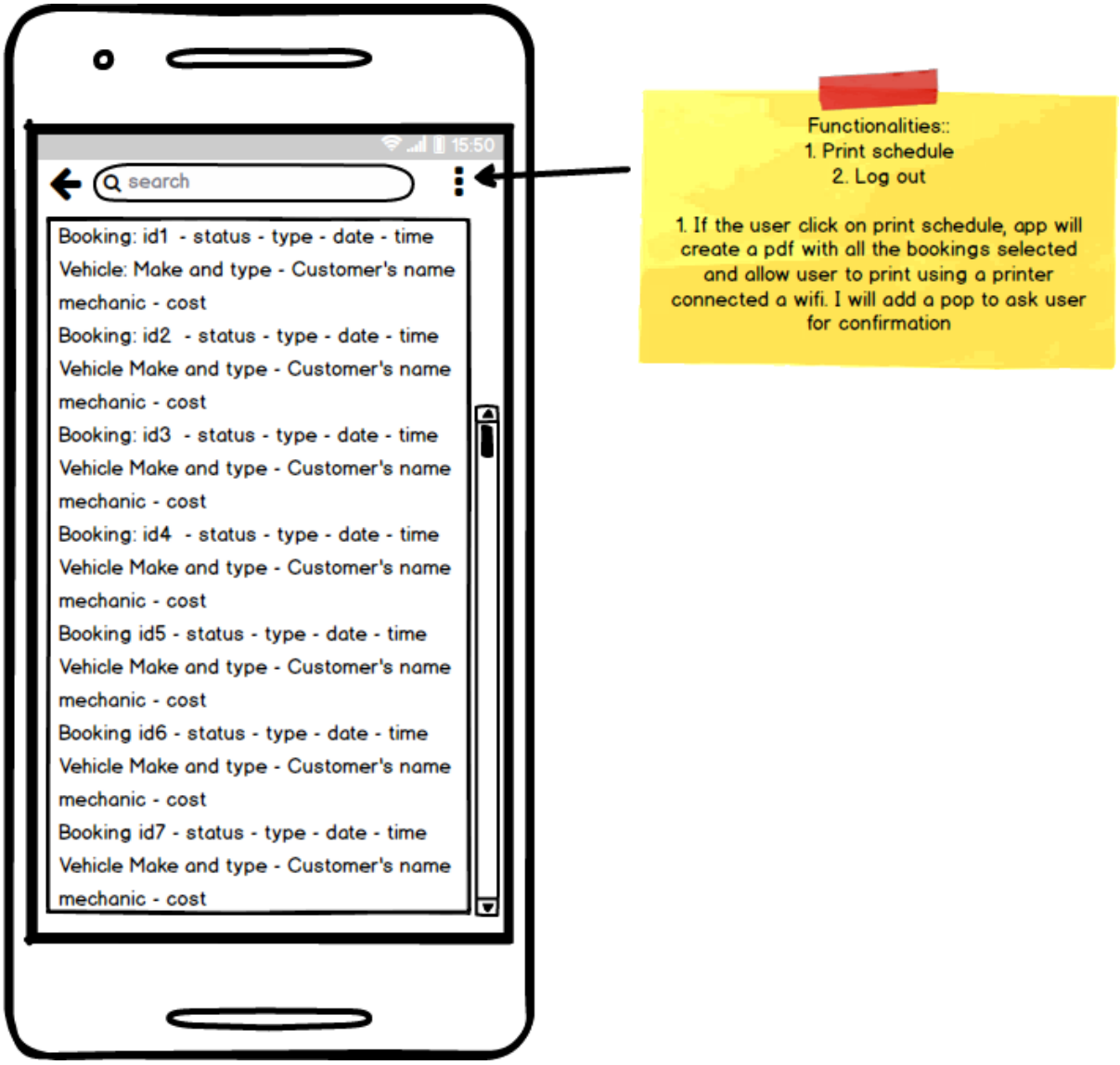
5) Unrepairable / Scrapped – when the fault cannot be fixed; in this case the car has either been taken away by the customer or has been sent for scrap.

(priority high)

This screen can be reused for allocate mechanic because they are quite similar.

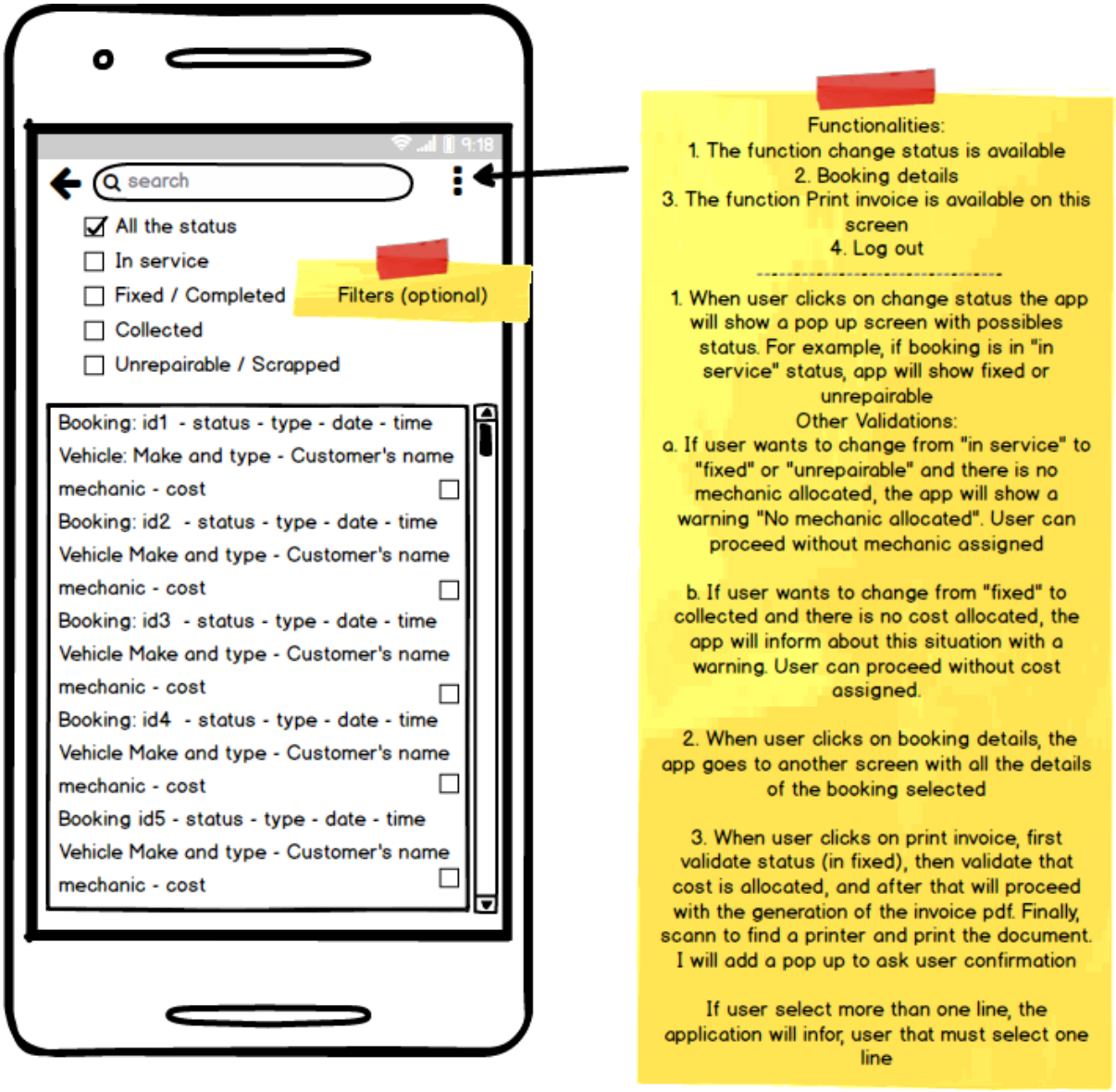
**Option 2:**

**Display Bookings / Print schedule button take us to the screen below:**




In this screen we don't have checkboxes, because every element which is selected from storage will be used to generate pdf. We could add checkboxes (optional).

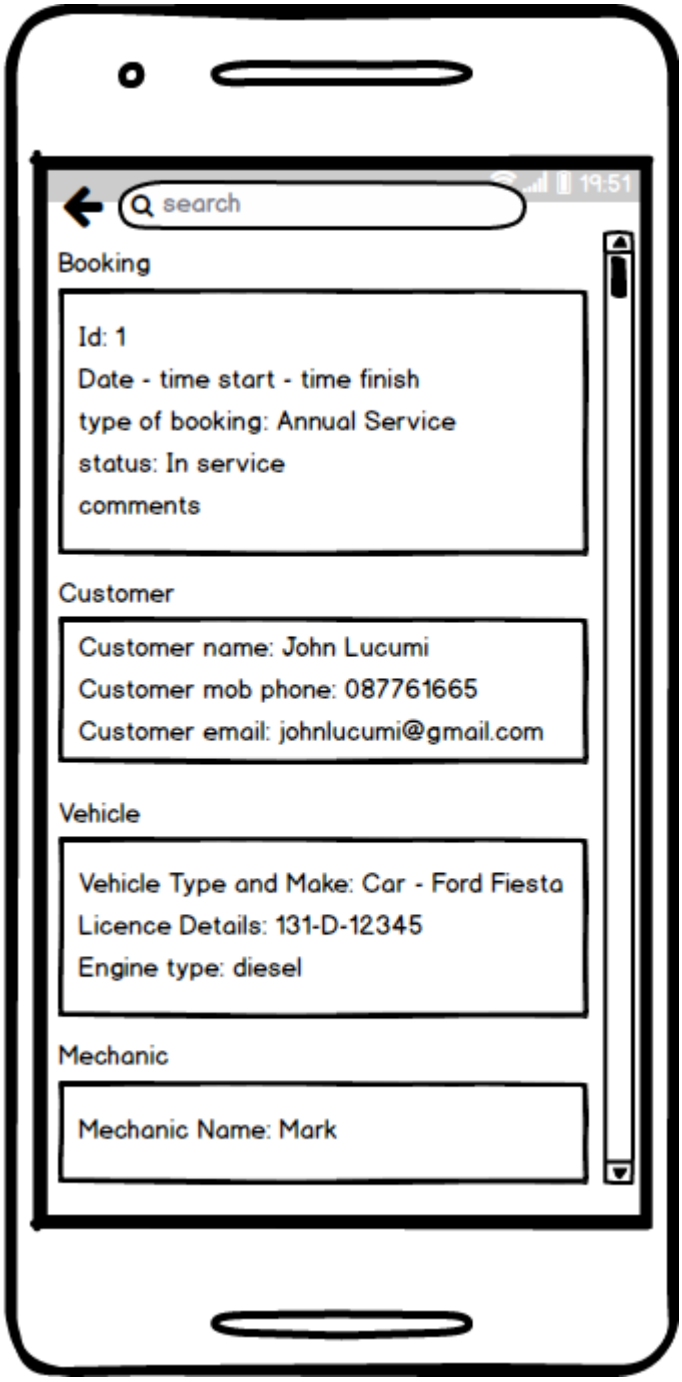
**Print invoice / change status / booking details button take us to the following screen:**




The only difference between option 1 and 2 is that print schedule is developed in another screen separated from other operations.

**Booking details screen**

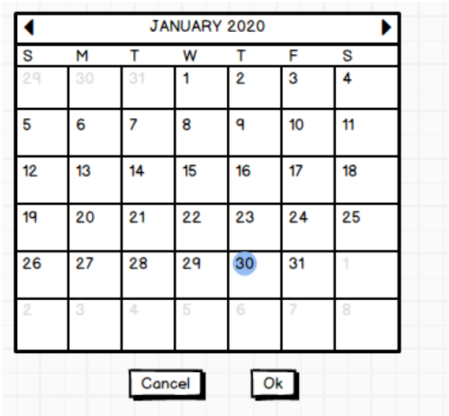
This one could be used for Admin home user option 1 and 2



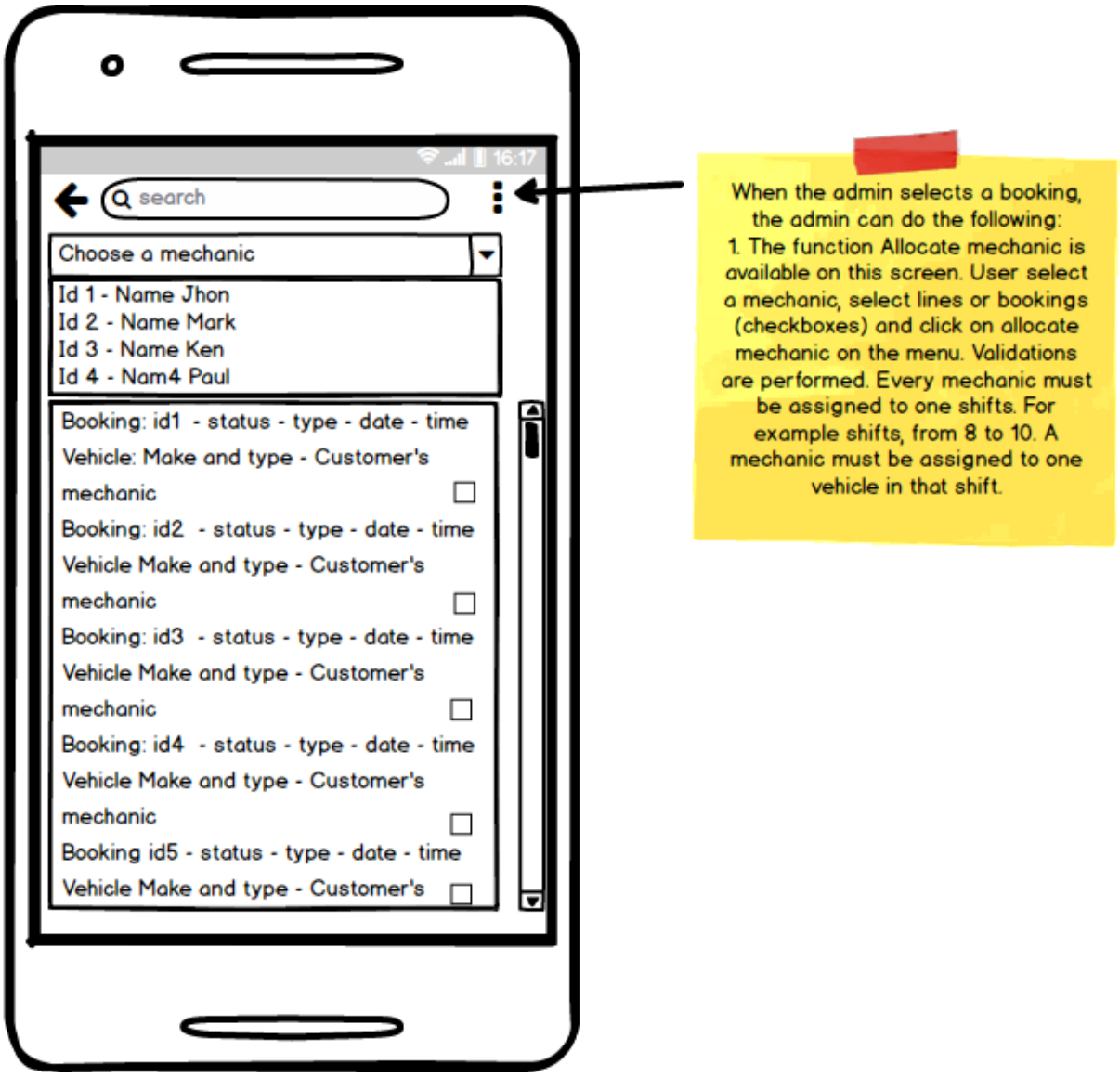

Information showed in Booking details is going to be used to generate the schedule pdf document.

**Allocate Mechanic screen (allocate mechanic per day)**

When user clicks on Allocate Mechanic Button on admin home screen (option 1 and 2), a pop up appears asking for a date (Date Picker). The date picker is shown below.



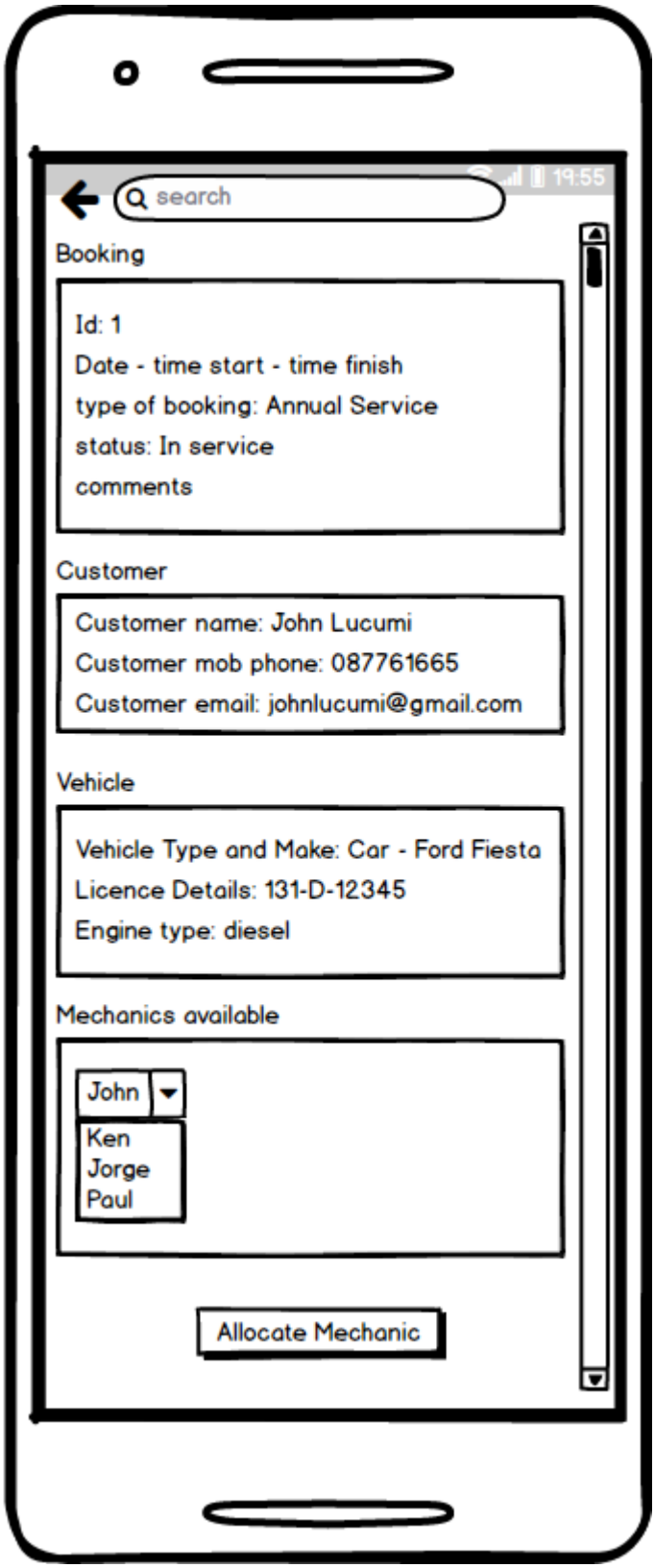

The following screen allow us allocate mechanics. This screen shows bookings with status “In service” or “booked” and is shown below:




We could add a pop up asking for confirmation.

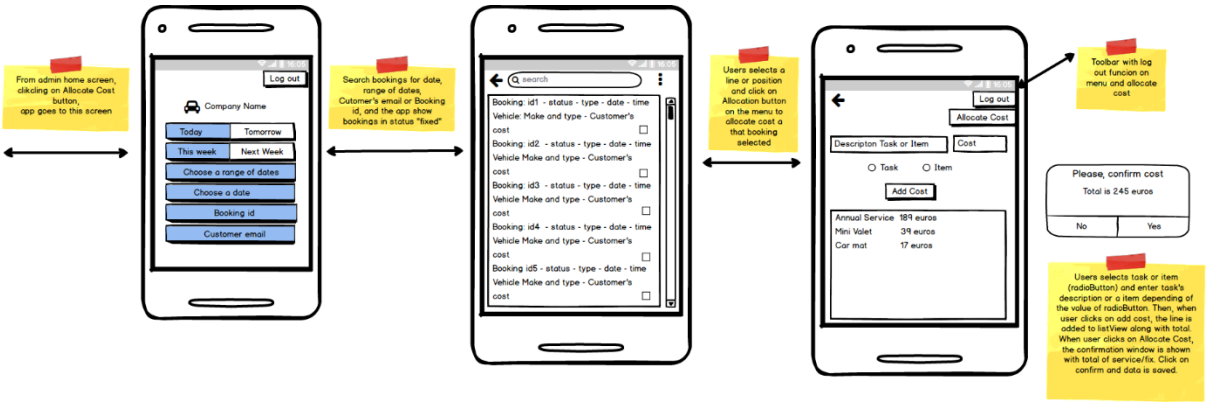
**Allocate mechanic 2 screen**

This one could be used for Admin home user option 1 and 2



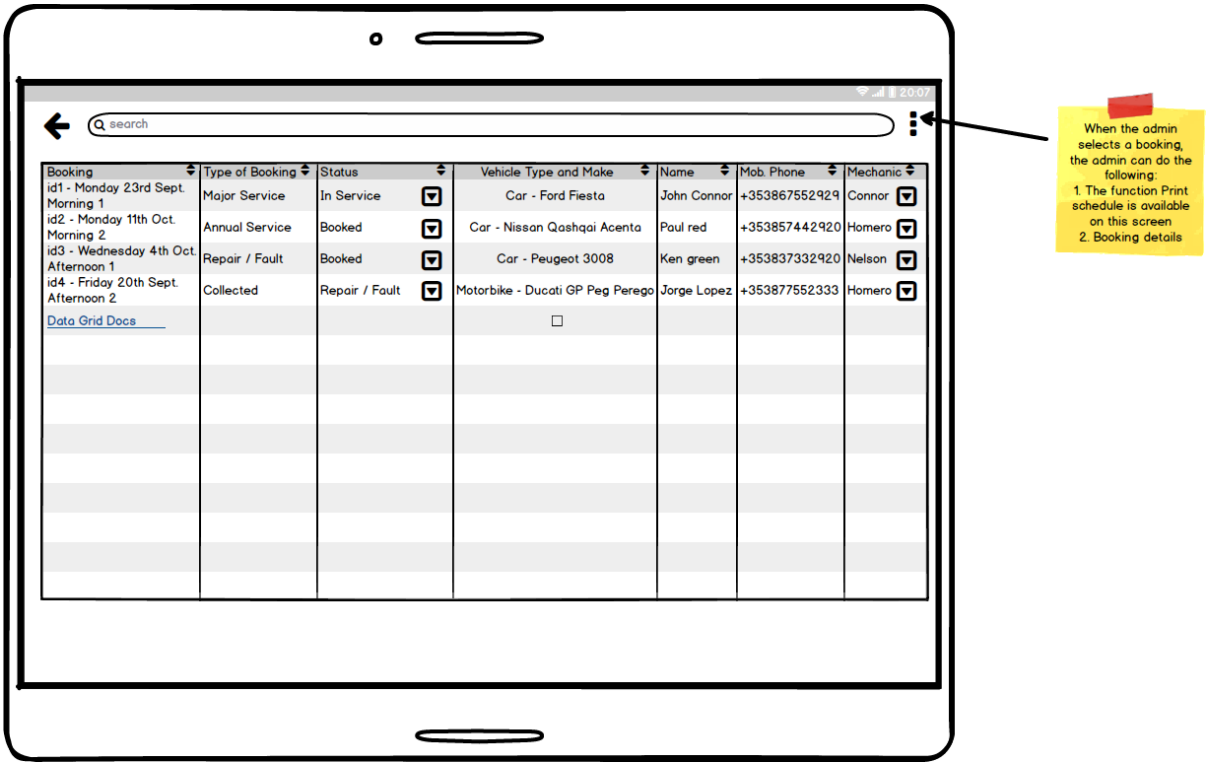

**Allocate cost screen**

This screen is used when user clicks on Allocate Cost button in the admin home screen (option 1 and 2). Before Allocate Cost, screen is shown, the application must filter bookings by fixed status. The sequence of screens from user clicks on Allocate costs button in admin home screen is the following:




The euro sign (€) must be added to the screen Allocate Cost (last one).

Another design to allocate a mechanic, change bookings status, print schedule and to see booking details could be the following (this design could be used for devices which have a screen bigger than smartphones):




The number of bookings allowed everyday are:

- Maximum: 16 (We have four mechanics and four shifts. Every mechanic can take at most four shifts, so 4 mechanics multiply by 4 shifts is 16)
- Minimum: 8 (If the booking is a Major Repair then this would count double, so 4 mechanics multiply by 2 shifts is 8)

There are 4 shifts:

- Morning 1: from 8 to 10
- Morning 2: from 10 to 12
- Afternoon 1: from 13 to 15
- Afternoon 2: from 15 to 17

Four Bookings can be made for each shift (maximum)

Two Bookings can be made for each shift (minimum)

To conclude, I'm going to create a list with the functionalities defined in this section.

1. Register **(High)**
2. Login (include forgot username or password) and Logout **(High)**
3. Make a booking for users (customers) **(High)**
4. Display bookings for users (customers) **(High)**
5. Change Profile **(Medium)**
6. Change Status per booking **(High)** or Change status massive **(Low and optional)**
7. Display Bookings for admin (owner of the garage) with filters: date, range of dates, booking id and customer's email **(High)**
8. Print Schedule **(High)**, print Invoice per booking **(High)** or print invoice massive **(Low)**
9. Allocate Mechanic per day **(High)** or Allocate Mechanic massive or more than 1 day **(Low)**
10. Make a booking for admin **(Low)**
11. Allocate Cost (include Allocate Parts) **(High)**
12. Display Booking details with all the information **(High)**

## Functional Design

### Domain Model

"Conceptual model of a system which describes the various entities involved in that system and their relationships" (BigLeap Solutions, 2020)

This model is very similar to class diagram and is not part of UML specification. There are no standard notations and this diagram have borrowed notations from the class diagram. We focus on entities from problem space, not solution space. In other words, we model entities related with the business space.

**How did I model this?**

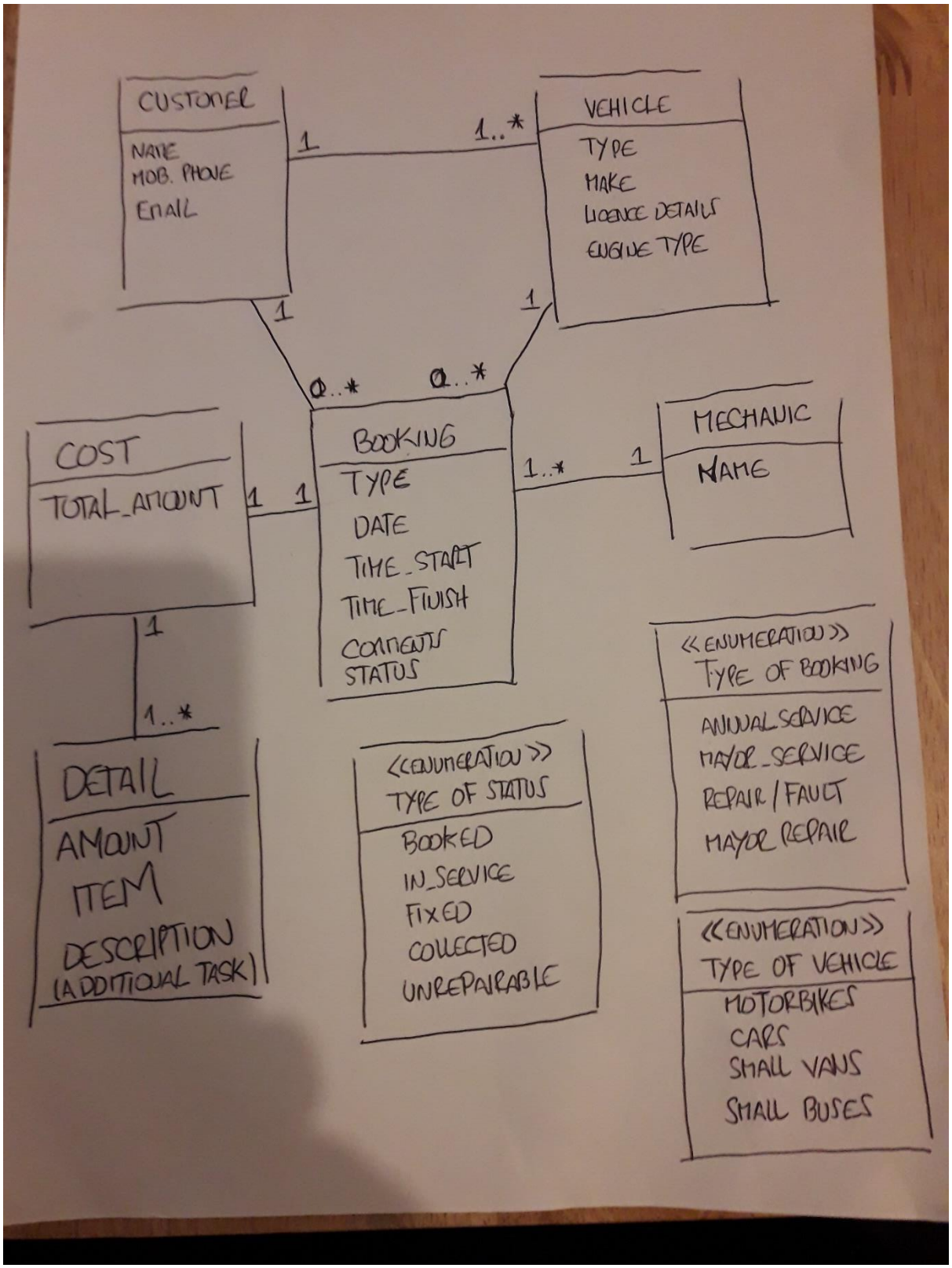
There are three steps in the following order:

1. Identify entities names (I underlined all the nouns in the brief which I thought will be good candidates for entity names)
2. Identify attributes of each entity
3. Identify associations between entities and put the multiplicities (degree of relationship between two different entities o classes).

**Why did you choose this domain model?**

Because it is used to have a good understanding of the business domain and tell us what is problem that we are trying to solve. With this model, we can create the sequence diagram using the entities designed.

I developed this model using a pen and a sheet of paper. Every class/entity has an identifier id.



## Sequence Diagram

UML Sequence diagrams are a type of interaction diagrams that show (with higher or lower details) how the objects of the system exchange message over time.

The sequence diagram shows the objects in the interaction and the sequence of messages among them.

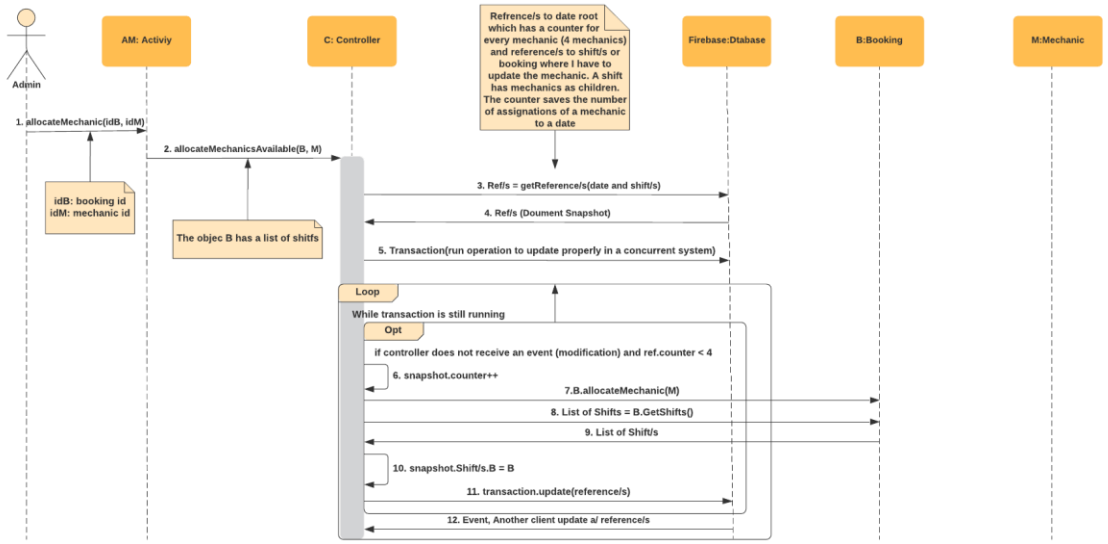
This diagram is time focus, but time is all about ordering, not duration because sequence diagrams shows the order of messages using a vertical line called lifeline. Lifeline represent an object or individual participant in the interaction.

**Why did I choose sequence diagram?**

Because this kind of diagrams help:

1. To think how to do what we are trying to solve because this diagrams are similar to algorithms.
2. Identify methods and where these methods should be, in other words, in what classes we should put these methods
3. How the methods must interact
4. Identify the responsibilities of each class.
   Example: we can identify:
   1. private attributes encapsulated of a class
   2. what objects another object must know
   3. what operations an object must include
5. To know the order of the calls among objects
6. To build the database design (JSON structure)

I created the sequence diagram for the main success scenario of **Allocate a mechanic to a vehicle** use case. The diagram is the following:



I assume:

1. the booking and mechanics were already got from the database along with its details
2. The admin just selected a booking and a mechanic

The previous sequence diagram shows the behaviour after the admin selected a mechanic, allocated the mechanic (clicking on Allocate Mechanic button) and confirms the allocation.

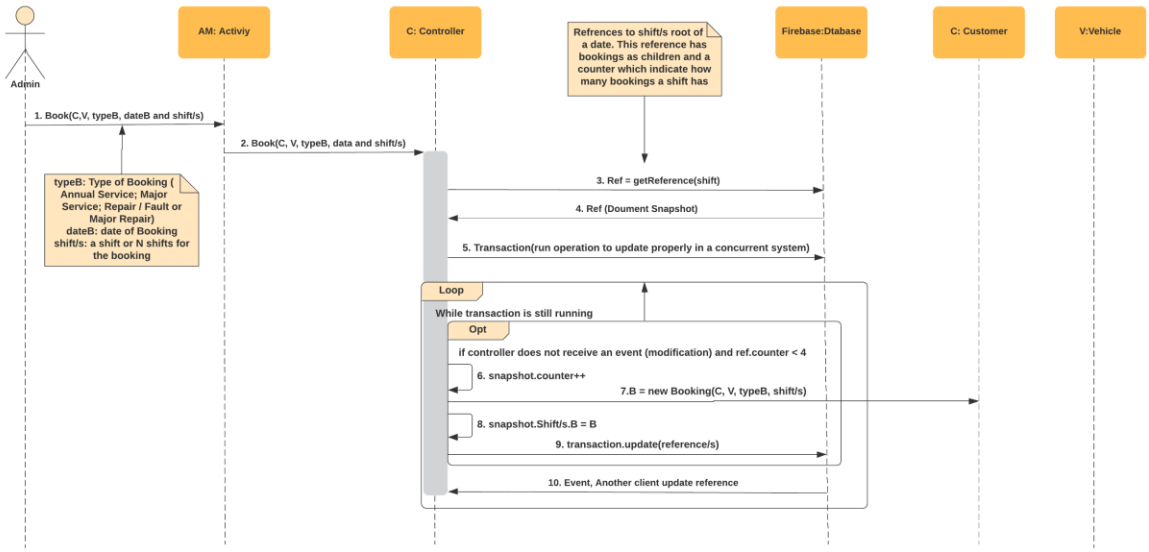
I'm going to explain some details about the previous diagram. The structure of the NoSQL cloud database firestore is very similar to json structure, in other words this database stores data as collections of documents. Therefore, to get information from firebase, we must have a reference to a tree or subtree to read/insert/update/delete elements on this structure.

The previous diagram uses a concept which is called transaction operation to lock parts of the database. It is not a real lock, but this ensure atomicity. The structure of the data (structure of the document in Firestore) will be explained further in this chapter, but I want to explain some things about the previous diagram:

1. I get a reference to a part of the database which has a mechanic counter for every mechanic. This tell me how many shifts in a date a mechanic has been assigned to (message number 3)
2. I get another reference to a part where I have the real booking with its details (it could be in different parts in the document at the same time because the denormalization is a better way of implementing this kinds of databases) (message number 3)
3. I use the transaction concept to ensure atomicity (message number 5). This concept is implemented how it was a loop. If the transaction operation detects there was a modification on tree/subtree referenced and the mechanic counter is less than 4, the operation retries again. It means, the operation is executed again from the beginning. If he shifts is equal to 4, it means that mechanic cannot be assigned to that booking. So, the transaction operations aborts and retry. Retry a number of times and abort throwing an error.
   The shifts and how many shifts can be assigned to a mechanic was already explained in the wireframes section.
4. The counter is incremented to indicate the mechanic has a new assignation, and the booking is updated with the new mechanic. Again, this booking could be in different places at the same time. So, we must update in different parts of the tree/subtree to have a consistent database (message numbers 6, 7, 8, 9 and 10)
5. The transaction updates the database with the counter + 1 and the booking with the new mechanic using the reference/s (message number 11)
6. The event of modification is represented in the message number 12

The diagram gives us an idea of how to do this method. This solution could be used to allow two admin allocate mechanics at the same time. We'll analyse this again; can be I find another solution during this process.

The second sequence diagram belongs to the Make a booking use case. The scenario represented by this diagram is the main success flow. The diagram is showed below.

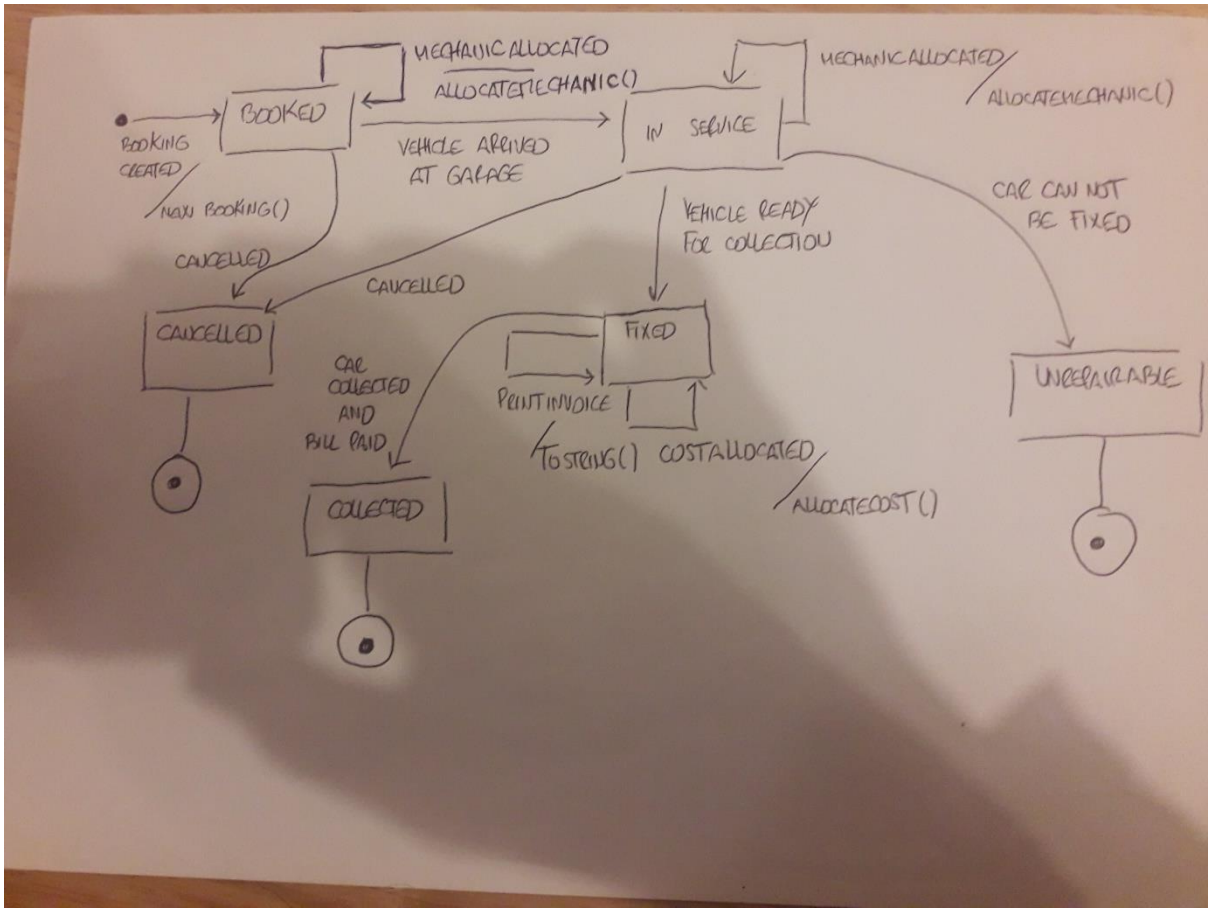



The booking details could be in different parts in the document at the same time because the denormalization is a better way of implementing this kind of databases. So, we must update the structure of the document wherever the booking is saved.

This diagram ensures each transaction is an indivisible and irreducible series of databases operations. To summarize we have four fields in the database (one for each shift) which contain a number with quantity of bookings in every shift. Therefore, if one of these numbers reach 4, there is no more space in that shift for a booking. Again, I used a transaction operation (Firestore feature) to ensure atomic operations.

## State Machine Diagram

I used a state machine diagram for the class booking because this class behave different depending of its state. Therefore, with this tool I can identify methods and I can identify when the app can use these methods, which depends on what state booking class/entity is in. The diagram is the following:




There is a new state which could be implemented to replace bookings. For example, a bookings is cancelled, we could book another customer for that day. We should implement the functionality cancel for customer users and admin. We can create a Cloud function in server (google platform) to delete cancel bookings or put them on another collection. I used transaction operations (secuence diagram) because If we want to cancel a booking, we will have two or more processes which could want to increment/decrement values in the database.

This diagram helps us to draw the process as a path. In the middle of the path we have validations and documents that I must check and create.

## Class Diagram

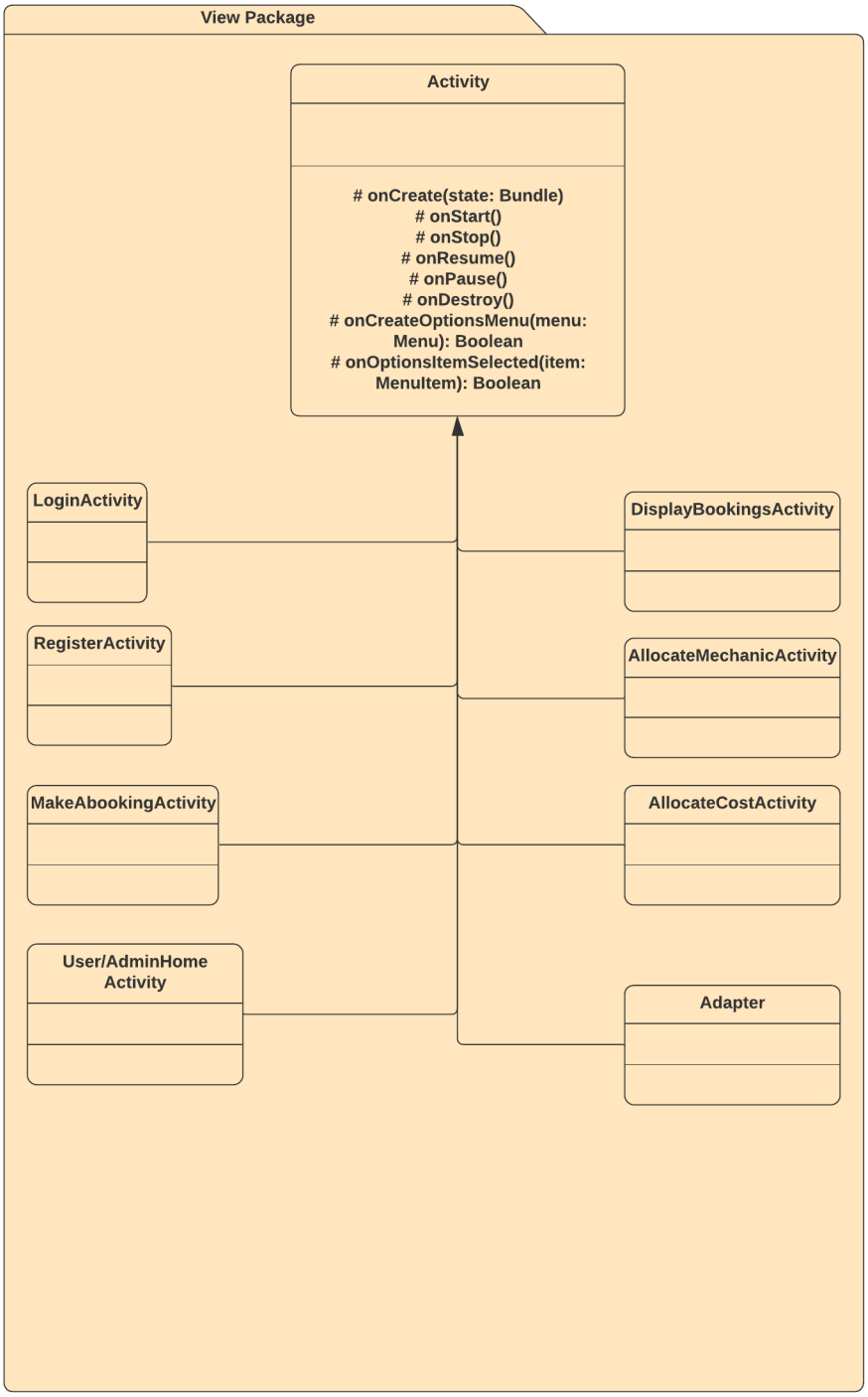
Class diagram is a type of static structure diagram which shows classes, their attributes and types (private, protected or public), relationship among classes (inheritance, aggregation), name of the methods, its formal parameters and the types, packages, etc. In other words, it shows the object-oriented structure of system.

Why did I choose class diagram?

I'm using Java programming language, which was designed following the object-oriented paradigm, so this diagram helps me to define the object schema of the software.

The structure is divided in 3 packages (View, Presenter and Model)

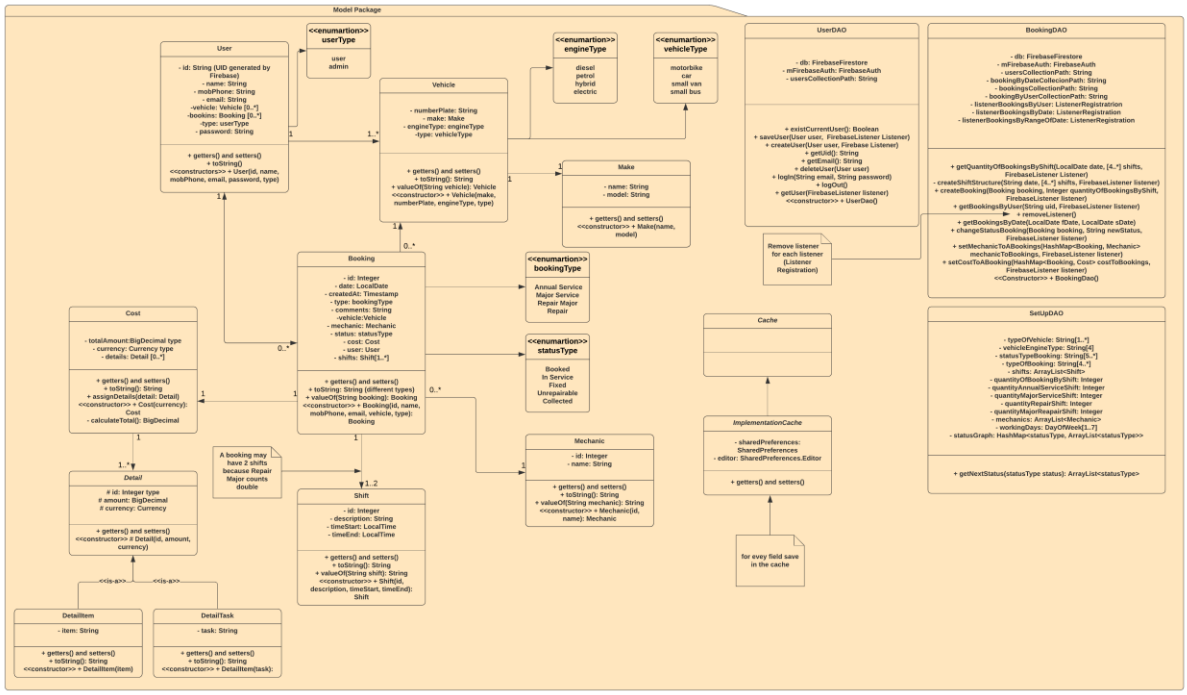
The structure of the view is the following:



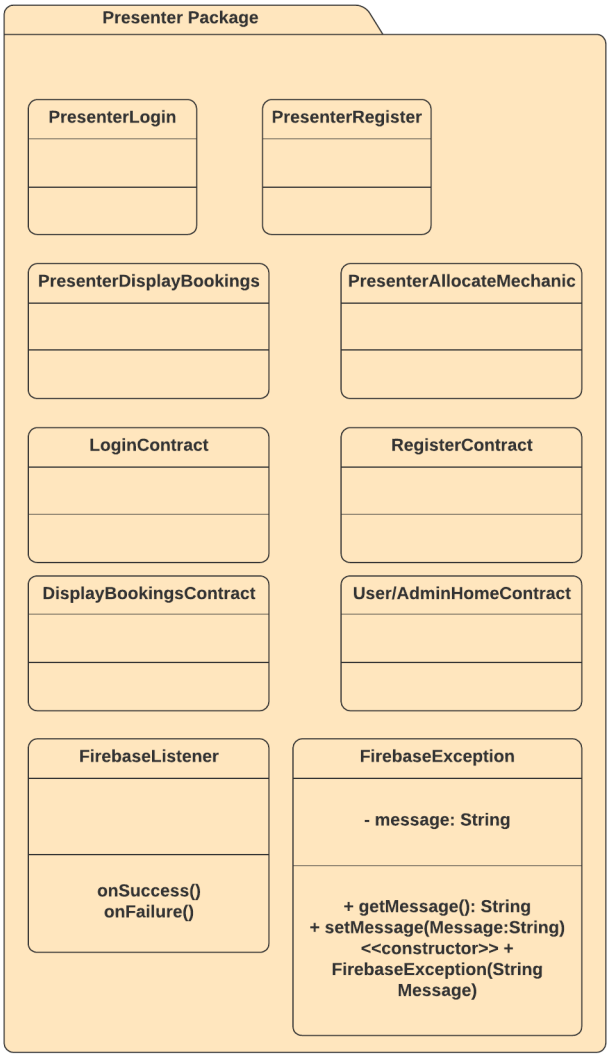

This structure shows that view package will contain classes related with the view. This package doesn't know anything about the model, view classes interact with the model through Presenter classes. Every class also implement a view interface which will define a kind of contract between view classes and presenter classes. Adapter is a class which extend ArrayAdapter class. This package will implement more views.

This package will be explained with more details further in chapter 4.

The structure of the Model is:



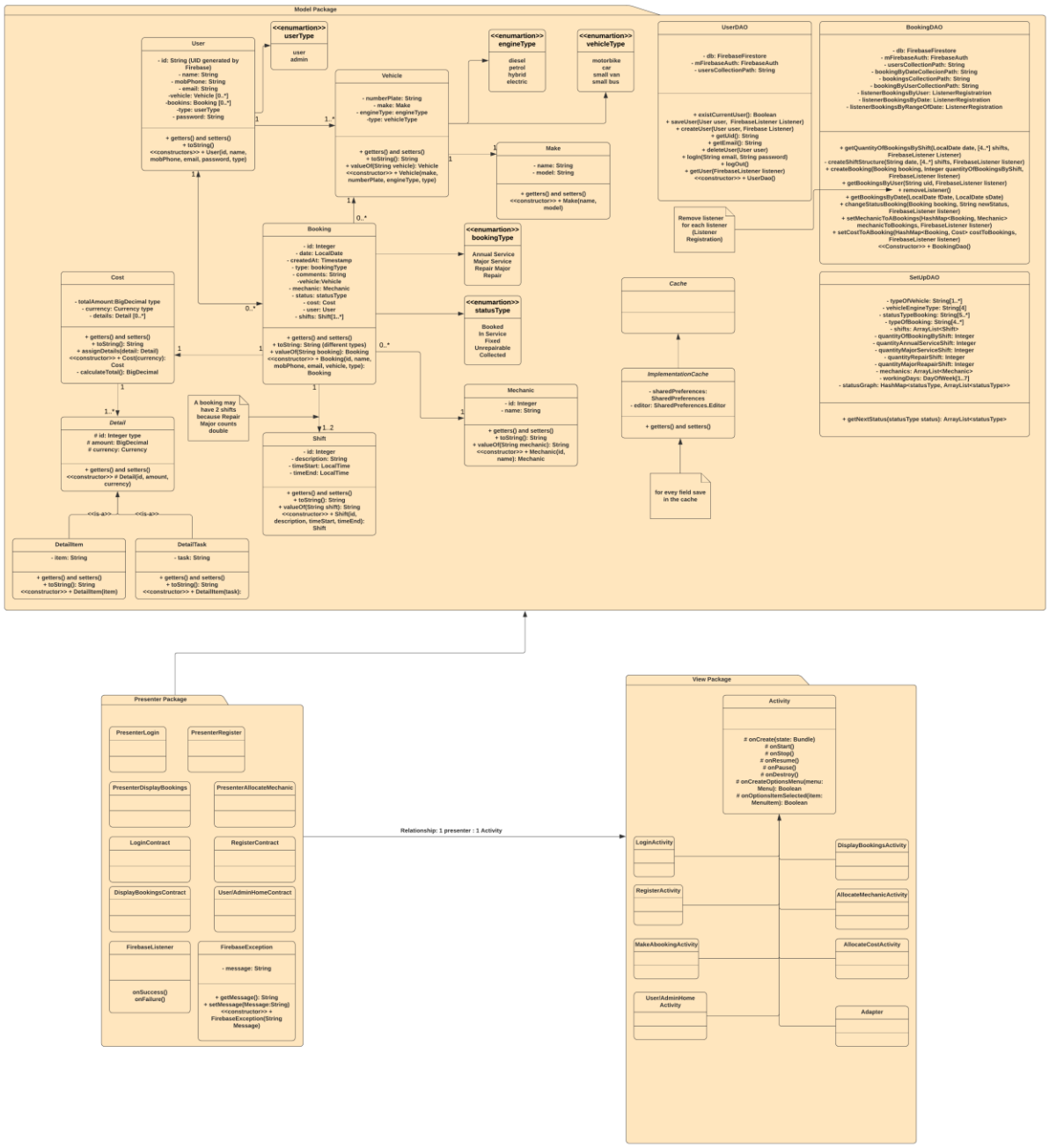

And Presenter,




This package contains all the presenter classes. This classes are the middleman in this diagram, they interact with the view and the model. In other words, they connect both. This package also has two other types of entities, which are the following:

1. FirebaseListener interface: This interface contains signatures which will be implemented by presenters to receive the results from database. It will be implemented like an event/catch concept
2. Firebase Exception class: Exception which will be thrown when database query return error.
3. Contract classes: These classes are a contract between Presenter and its pair View. Define methods which will be called from presenter to view and vice-versa, so they know each other. This concept also helps to replace both classes or one of them for another implementation, just implementing the contract.

Then, I show all the packages and how they interact,




## Database design (Firestore)

The data storage is organized as a tree because I'm going to use a No-SQL document-based cloud database called Firestore, which is organized in collections and documents. Each collection can have many documents, which could be different. Each document can have subcollections and fields. Fields could be of different types (number, boolean, map, string, map, array, timestamp, etc).

I created database structure of my application. You can find it into the folder database.

I tried to use Hackolade (version 3.5.5) to make my database design installing a firestore plugin, but the document generated by this application is not easy to read and doesn't show a full hierarchy of the structure.

Considerations:

1. Information about different entities of the program can be repeated in different parts of the database because if we do this, we don't need to access twice or use two references to retrieve data. For example, in my database users' collections contain users and their list of booking. The same list is saved in another part of the three to save reading and access to database. Therefore, denormalization is good practice.

   For example:

   User with UID equal to O5WexYCpyHYzXUEXuxsH4Lh4LUq1 has the list of bookings in:

   /garage/userInformation/users/O5WexYCpyHYzXUEXuxsH4Lh4LUq1/ListOfBookings

   and the booking 145 which belongs to previous user is at

   /garage/userInformation/users/O5WexYCpyHYzXUEXuxsH4Lh4LUq1/ListOfBookings/145

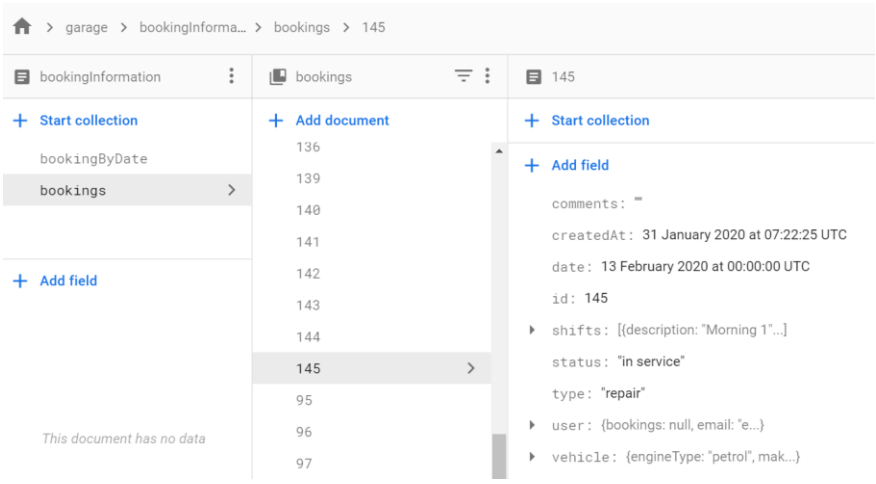
   And is also saved in

   /garage/bookingInformation/bookings/145.

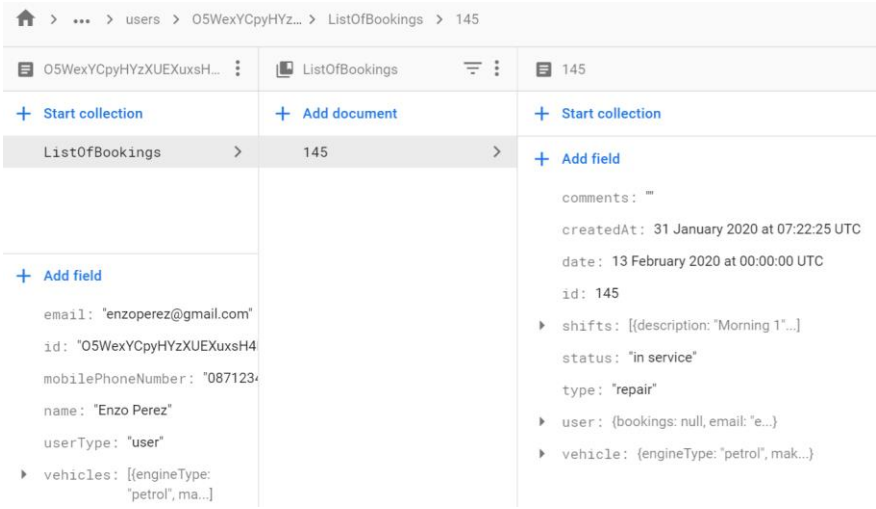
   Therefore, we have this information repeated twice, one below user and below bookings.

   Example: Images below show this.

   Booking number 145 at bookings collection.




   Booking number 145 at ListOfBookings collection below user information

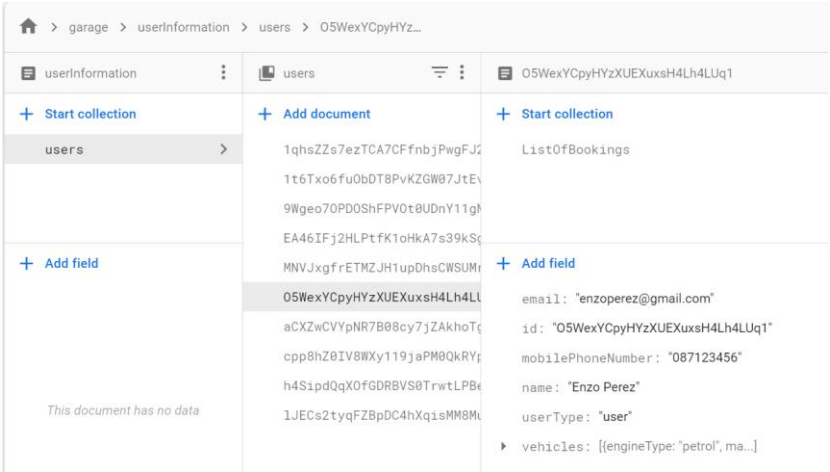



2. This reference

   /garage/userInformation/users/aCXZwCVYpNR7B08cy7jZAkhoTgv2 contains personal details about a user. Password is not saved here due to security reasons. Password is saved with the email because I'm using another service provided by google which is called Firebase Authentication. This service is used to log in/log out/register a user. Email field is also saved in my database (Firestore).

   Additionally, I use UID to distinguish a user from another user, in other words, it is the name of the document which has almost all the information about a user. From my point of view, UID is just an identifier for an account, and it does not have any encoded information. If somebody want to crack an account, they will need credential to access to information. Therefore, we can't get any personal data if we know this value.

   Example: In the Image bellow, we can see a list of UIDs which represent a user in Firestore.




   The only thing hackers and crackers can do is try to access to my database tree using these UIDs but they should know the path, I mean the full path. Therefore, I must customize my security rules correctly to avoid hackes/cracker can damage or steal information using this string.

3. Parameters of customization are also saved in the database to make the application more customizable.
4. Indexes to fields of the documents are created automatically by firebase which makes queries more efficient. So, I don’t need to create indexes.

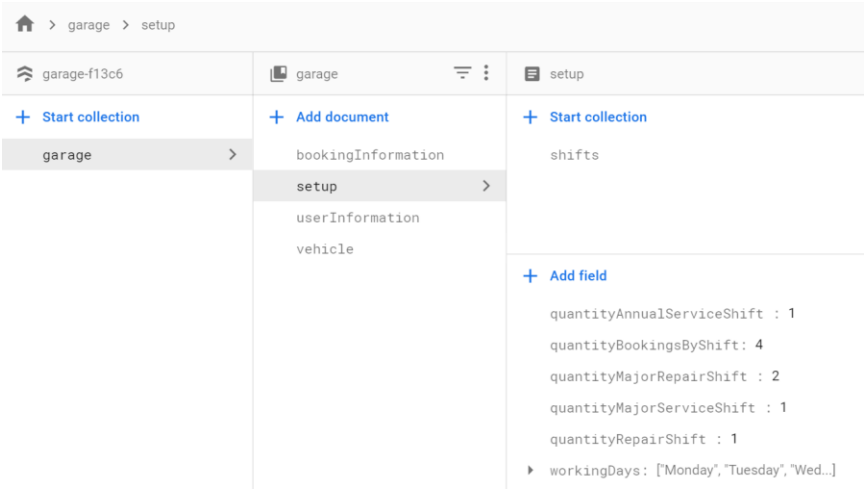
# Chapter 4 - Implementation of the system

First, I'm going to describe the different technologies I used to develop this application:

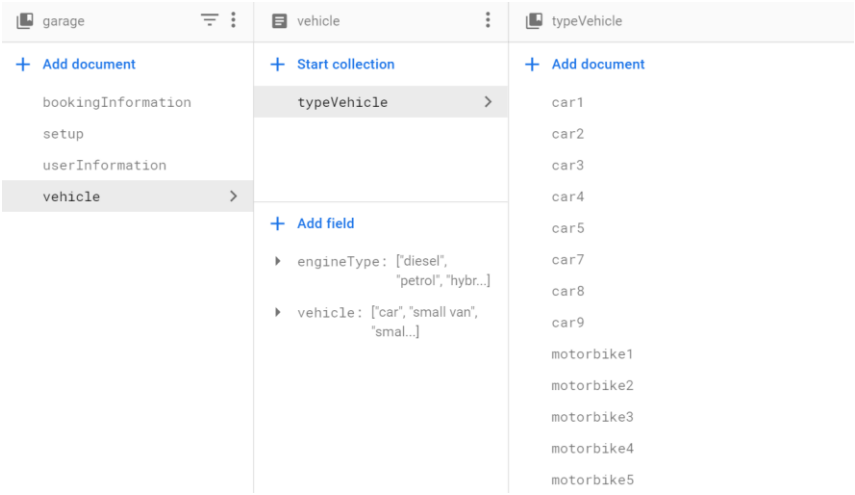
1. Firebase Authentication: it is a service provided by firebase to work with user authentication (version 19.2.0).
2. Android Studio: it is the standard integrated development environment to develop native applications for android (version 3.5.3)
3. Firestore: it is No-SQL cloud database provided by Firebase (version 27.3.1).
4. Andorid 9 OS (PIE) – API Level 28 - Framework SDK 28: I'm developing my mobile application for this platform using level 28 of API.
5. Windows OS: It is the operative system in which I'm developing my application. It is the host OS for Android Studio IDE.

Before starting the project, I must add the dependencies for every module that I want to use. In this case I used Firebase for Cloud Firestore and Firebase Authentication. So I added this dependencies to the file build.gradle in the folder app. I also used other libraries which I'm going to explain when further in this documentation.

So far, I've developed some of the functionalities which were analysed in the previous chapter. In other word, I used the first part of this document how a guide to develop the product. The first thing I did, after installation of the all product, was created the database in Firebase. I created a project called garage and install the dependencies in Android Studio to work with Firebase. I tested everything, when it worked, I added part of the data design of the database to Firestore that I considered necessary because how this database is document-based we can add documents and fields anytime. We do not need to create everything, how we do with databases which have an schema. Therefore, I added only information I needed to start with my development. Image below shows the setup parameters and




Customization of the application



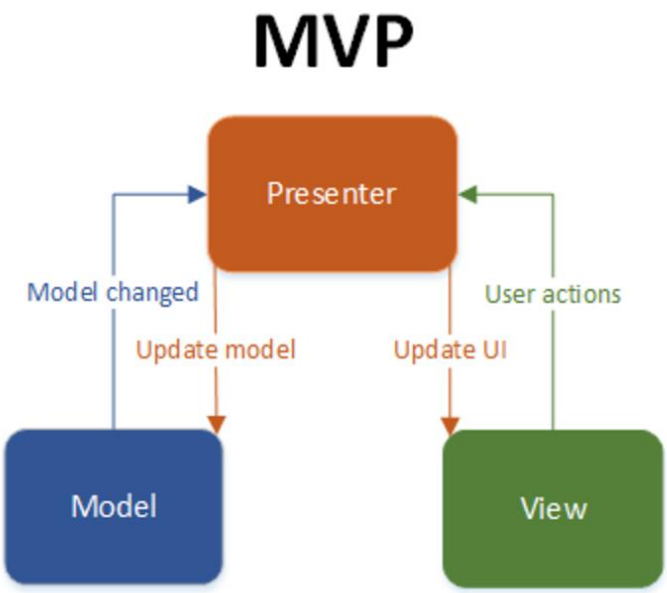

Customization of the vehicle

First image shows information to make the application customizable, for example: how many bookings a shift can have, workingDays and shift objects, etc.

Second image contain setup of the vehicle. For example, type of vehicle, type of vehicle makes, etc.

How I'm using the Architectural pattern design MVP (Model – View - Presenter), because I like when everything is organized and understandable, every screen/wireframe must be developed following this architecture. This architecture was presented in the previous chapter, when the diagram class was presented. This pattern design directs the development and testing.

How this pattern works?



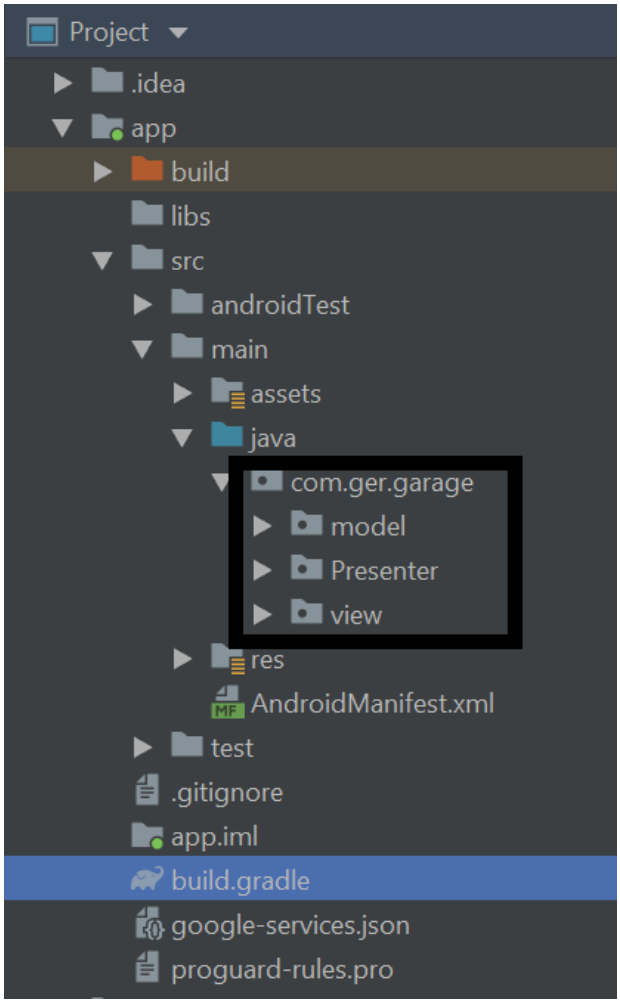

Elements in MVP:

**Model:** This package contains everything related with database management, caching data. This module also has small classes which are used to put data from database (Booking, User, etc.)

**Presenter:** It is the middleman between model and view. This module allows model and view connect and pass information from view to model and vice-versa. This module is also the brain of the application.

**View:** This package contains what users see. Screens and catch events, listeners, spinners, buttons, etc. This module does not know anything about the model. It knows a booking how a string of characters.

I created three packages to represent this structure




How Firebase works with asynchronous results, the logic flow can be awake in model because application receive an update notification from database or in the view because user trigger an event. So, it could be seen like a circle, if the user does something on the screen, it tells presenter. This one does something with this, could be give an answer immediately or talk with the database/cache. The model could query database or cache and give an answer to the presenter. Then, presenter will inform this to screen. On the other hand, if the model receives a notification of change, this one will inform presenter and presenter talk immediately with the screen.

This pattern design is going to direct everything. With this, I want to say, Development and Testing.

Development because when I must develop a screen, I must think in 4/5 classes. View Class (activity), Presenter class, Contract class, model classes and cache/Dao classes. Contract class is a kind of contract between view and presenter, they must implement this class to know each other. This allow us to change the view or the presenter without a big impact on the program, just we need to implement the interface. I'll give some examples about this contract during this chapter.

Testing because this pattern allows to create classes loosely coupled. So, we can test these classes separated. This will be explained in the next chapter.

With Register and Login functionalities, I created the activities classes and I did everything inside, in other words, connection to database and logic was inside of Register/Login Activity. When they worked, I did a refactoring of register. I created the Registerpresenter and the userDao and the contract Interface for view inside or RegisterPresenter. Login was not modified, because I didn't have time.

With the rest of the functionalities, the architectural pattern was applied from beginning. I created a contract (with two interfaces one for presenter and another for the activity/view), then presenter class, view class and re-used classes/methods from model. I would have liked to have more time to refactor everything and deliver a program totally MVP oriented.

I left an image with the diagram class generated by Android Studio. This one is in class diagram folder.

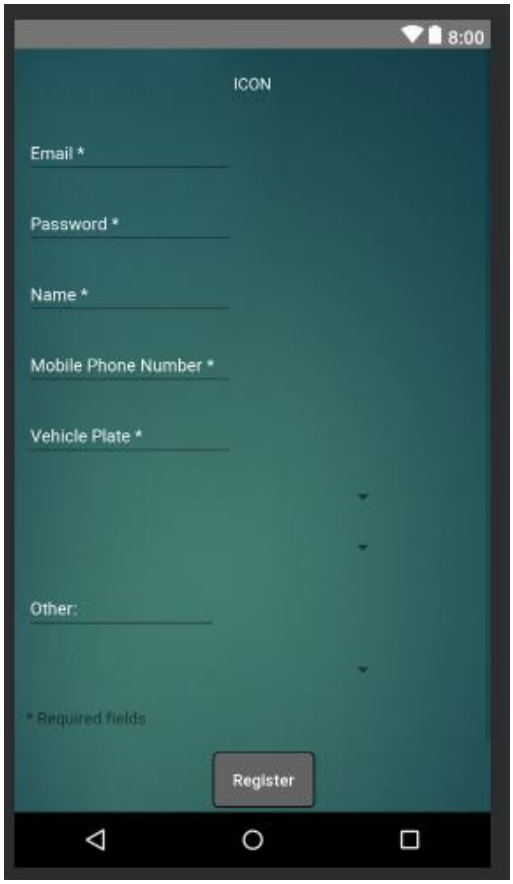
## How was the process? And What do every functionality need to be finished?

From this point I'm going to explain every functionality/screen used in this development.

First, I created register and login screens using the wireframes and the class user using the class diagram.

### Register Functionality / RegisterActivity Class

This is the screen for register:




It is a scrollView layout as parent and a linear layout to organize the different elements inside of the screen. More elements used: editText, spinners, button, textView. The following images are global variables defined in RegisterActivity class for the elements in the screen.

```
// Screen Elements
private EditText email, password;
private EditText mobilePhoneNumber;
private EditText name;
private EditText other;
private EditText vehiclePlate;
```

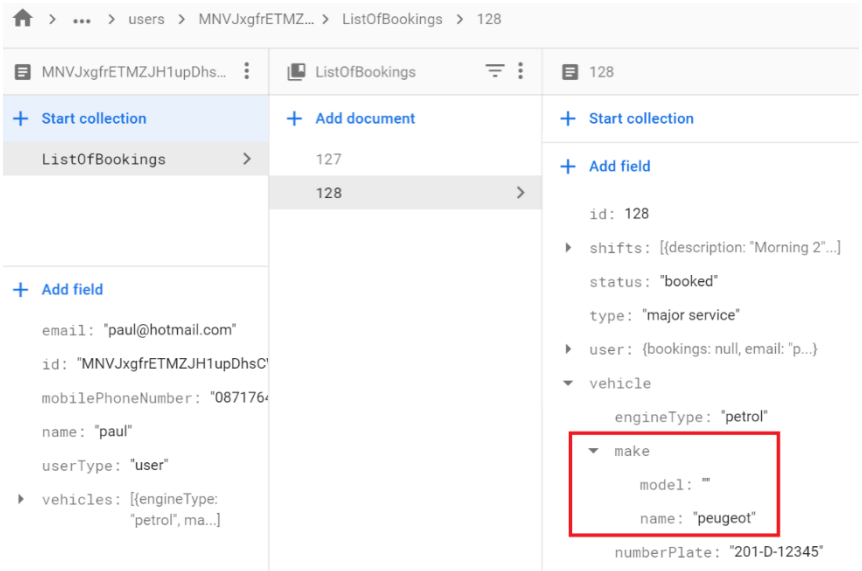
```
// More elements
private Button btnRegister;
private TextView haveAccount;
private Spinner typeOfVehicles;
private Spinner makeOfVehicles;
private Spinner engineTypeOfVehicles;
```

RegisterActivity class responsabilities:

1. interact with users, catch events and implement handlers for different events triggered by elements (button, editText, etc) on the screen.
2. This class also fills spinners using an ArrayAdapter<T>. When the information changes they are updated properly using notifyDataSaveChanged method, which belongs to ArrayAdapter <T> class.
3. Collect information to register a user.
4. Validations:
    a. Check every field is not empty
    b. Check plate number calling a method of presenter class
    c. I didn't add validation for email, password and mobile phone number. Firebase authentication validates if password length is greater or equal than 6. I must add validations for email and mobile number, and reinforce password validation
    d. I must add an icon
    e. I must expand fields every field a little bit, they are so small. If the text is long, this one is hidden behind screen
    f. I also must add a flag when I call to user/admin home screen with intent class because if I click the button twice two home screens are open. One of the possible solutions could work is FLAG_ACTIVITY_CLEAR_TOP which reuse activities if these one are in the activity stack. I could add a progress bar to show user that has to wait for the result.

```
// go to adminHomeScreen (owner of the garage) or userHomeScreen (customer)
public void goToHomePage(String userType) {

    if (userType.equals(admin))
        startActivity(new Intent( packageContext: RegisterActivity.this, AdminHomeActivity.class));
    else
        startActivity(new Intent( packageContext: RegisterActivity.this, UserHomeActivity.class));
}
```

    g. I must add another field, which represent the model of the vehicle. I have two field in the database to register the make and the name of the model (see image below).



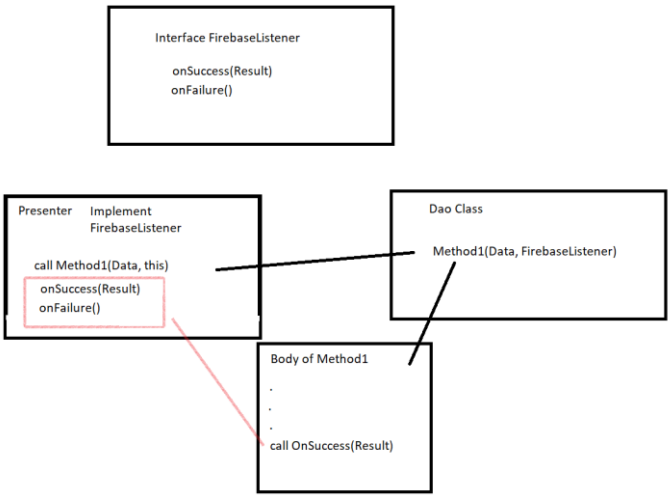

RegisterPresenter class:

1. Check if user is logged in using the UserDao class
2. Get information from screen and pass it to createUser method (UserDao class) to register a user. Previously this class create a user object
3. Check if the user is admin or normal user to tell View/Activity what screen the user should be redirected
4. Receive information from UserDao or Firebase authentication with the results of the registration and inform to the screen about the result. Call a showMessage() method of the RegisterActivity class
5. Validate number plate with the following regular expression

```
private final String regexPlateNumer = "^([0-9]{2})[12]?-
(C|CE|CN|CW|D|DL|G|KE|KK|KY|L|LD|LH|LM|LS|MH|MN|MO|OY|RN|SO|T|W|WH|WW|WX|LK|
TN|TS|WD)-([0-9]{1,6})$";
```

   I must add to this validation that the first 2 digit, which represent a year, are less or equal than 20 and the third digit must 1 or 2.

6. I get information about vehicle (type, make engine type) from SetUpDao and pass this to the RegisterActivity class
7. Implement a method called detach to clean memory before RegisterActivity be destroyed
8. This class has an interface View, which is implemented by RegisterActivity. So, RegisterPresenter knows RegisterActitvity class but not vice-versa. This must be changed, a class separated must be created with two interfaces, one for Registerpresenter and another for RegisterActitivty.
9. I implemented handlers to receive results from Firebase authentication/Firestore. I don't agree with this because presenter classes just must use classes from java library. This classes **must not work** with android/firebase libraries. It has to be a pure java class. Therefore, what we can do is, create an interface called FirebaseListener with methods. Then presenters' classes implement this new class and they give a body to the methods. Lastly, when we call methods of Dao classes, we pass presenter how actual parameters and the result of the database query is sent to the presenter classes using the methods created in FirebaseListener. Image below explained this:



SetUpDao Class:

This class passes information to presenter about type of vehicle, make of a vehicle and engine type. I didn't have time to implement queries to the database (Firebase)

UserDao Class:

Private attributes:

```
private FirebaseFirestore db; // database
private FirebaseAuth mFirebaseAuth; // authentication
private final String usersCollectionpath = "garage/userInformation/users"; // collection of users
```

db: attribute allow us to query Firestore

mFirebaseAuth: allow us to manage user's authentication

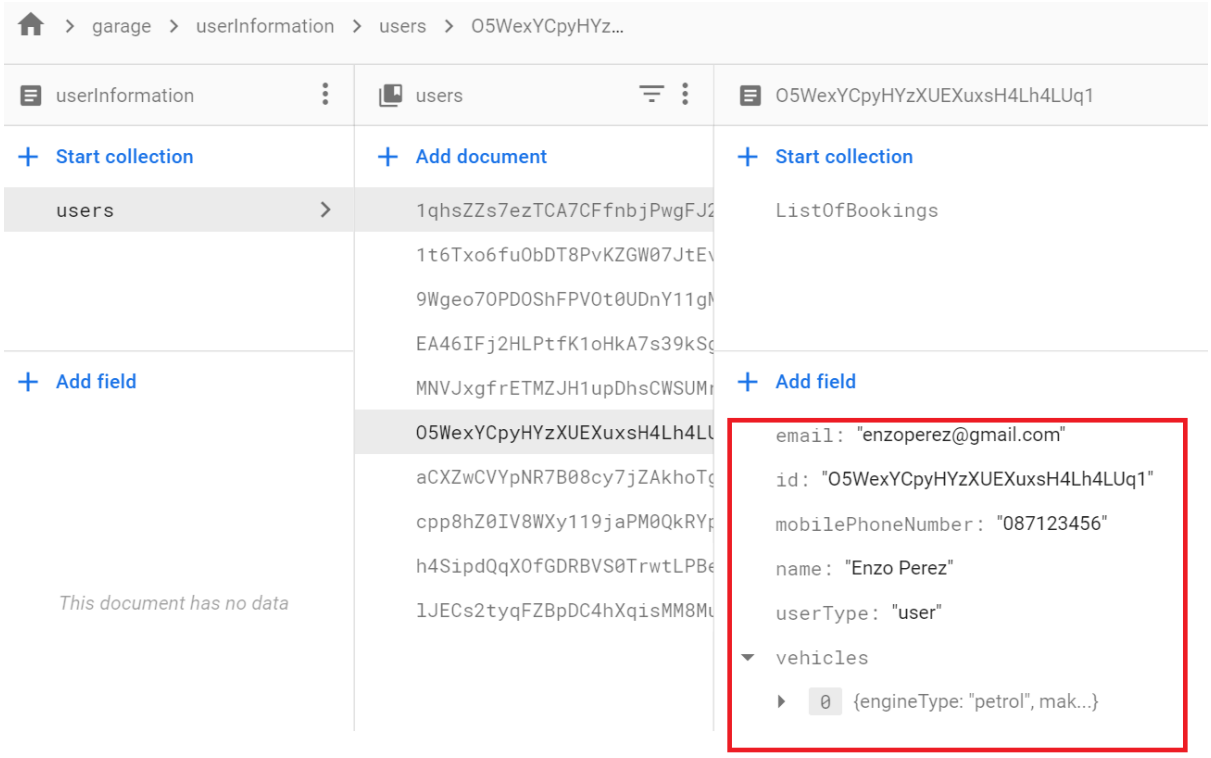
usersCollectionpath: is the Firestore link where the users have personal details

The image below shows how to set information about the user in Firestore using a Hashmap and a method called set provided by firebase library. This logic creates a document with UID (firebase authentication string) how id. The information saved is showed in green.

```
// Save user after login because login is made with firebase authentication
// which save username and password. Rest of information is saved in Firestore
// Anyway we can save more information about user in firebase Authentication (for example mobile phone number)
public void saveUser(User user, OnFailureListener failureListener, OnSuccessListener<Void> successListener) {

    Map<String, Object> docDataUser = new HashMap<>();

    docDataUser.put("id", user.getId());
    docDataUser.put("name", user.getName());
    docDataUser.put("mobilePhoneNumber", user.getMobilePhoneNumber());
    docDataUser.put("email", user.getEmail());
    docDataUser.put("userType", user.getUserType());
    docDataUser.put("vehicles", user.getVehicles());

    db.collection(usersCollectionpath).document(user.getId()).set(docDataUser)
            .addOnSuccessListener(successListener)
            .addOnFailureListener(failureListener);

}
```

Remember that information about user is saved in Authentication Firebase service (email and password) and in Firestore (id, name, email, mobile phone number, userType, vehicle).

Following image shows how the information is saved in Firestore:




The second image shows how to use firebase authentication to register a user.

```
// Register User
public void createUser(User user, OnCompleteListener<AuthResult> listener) {

    mFirebaseAuth.createUserWithEmailAndPassword(user.getEmail(), user.getPassword())
            .addOnCompleteListener(listener);

}
```

Second parameter is the listener, this formal parameter must be of type FirebaseListener and RegisterPresenter implement this interface to receive the result of the registration.

## Login Functionality / LoginActivity Class

This class was not refactorized, I'll explain how to the user is logged in and a listener which listen if the user is already logged in.

The image below signs in a user with an email and a password. The parameter task which is inside of listener bring the result of the sign In. The user is redirected depending is this one is admin or normal user. A message toast is shown in case of error.

```
mFirebaseAuth.signInWithEmailAndPassword(email, pass) // sign in with an email and password
        .addOnCompleteListener( activity: LoginActivity.this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) { // handler

                final String authenticationFailedMessage = "Authentication failed: ";

                if (task.isSuccessful()) { // result

                    Intent intentHome;
                    // redirect user to home
                    if (!mFirebaseAuth.getCurrentUser().getEmail().equals(setUpDao.getEmailAdmin()))
                        intentHome =  new Intent( packageContext: LoginActivity.this, UserHomeActivity.class);
                    else
                        intentHome = new Intent( packageContext: LoginActivity.this, AdminHomeActivity.class);
                    startActivity(intentHome);

                } else {
                    // shows error message
                    Toast.makeText( context: LoginActivity.this,  text: authenticationFailedMessage + task.getException().getMessage(), Toast.LENGTH_LONG).show

                }


            }
        });
```

The second image shows another listener which is activated if the user is logged in, this one is redirected to home screen.

```
// firebase authentication listener tell login screen
// if the user is logged in
mAuthStateListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        // get current user
        FirebaseUser currentUser = mFirebaseAuth.getCurrentUser();
        if (currentUser != null) { // if it is logged in, it is != null

            Toast.makeText( context: LoginActivity.this,  text: "You are logged in!", Toast.LENGTH_SHORT).show();
            Intent intentHome;
            if (!mFirebaseAuth.getCurrentUser().getEmail().equals(setUpDao.getEmailAdmin()))
                intentHome =  new Intent( packageContext: LoginActivity.this, UserHomeActivity.class);
            else
                intentHome = new Intent( packageContext: LoginActivity.this, AdminHomeActivity.class);
            startActivity(intentHome);
        } else { // show toast message
            Toast.makeText( context: LoginActivity.this,  text: "Please Login", Toast.LENGTH_SHORT).show();

        }

    }
};
```

I also must add a flag when I call to user/admin home screen with intent class because if I click the button twice two home screens are open. One of the possible solutions could work is FLAG_ACTIVITY_CLEAR_TOP which reuse activities if these one is in the activity stack. I could add a progress bar to show user that has to wait for the result.

## UserHomeActivity Class

This activity represents User Home Screen:

Information about user (name, email and mobile phone number) is asked to the presenter class. Presenter ask this to UserDao class using getUser(FirebaseListener) method. UserDao class use get() method of the DocumentReference class to get information from user. The result is sent to the class which call this method, because this class implements Interface FirebaseListener, and therefore give a body to onSuccess(User).

```
// get information of the user from Firestore
public void getUser(final FirebaseListener listener) {

    DocumentReference docRef = db.collection(usersCollectionpath).document(getUid());

    // Source can be CACHE, SERVER, or DEFAULT.
    Source source = Source.SERVER;

    docRef.get(source).addOnSuccessListener((OnSuccessListener) (documentSnapshot) -> {

            User user = documentSnapshot.toObject(User.class); // get the object user from database
            listener.onSuccess(user);

    });

}
```

In this screen, we have 6 classes/interfaces interacting in the cycle which are:

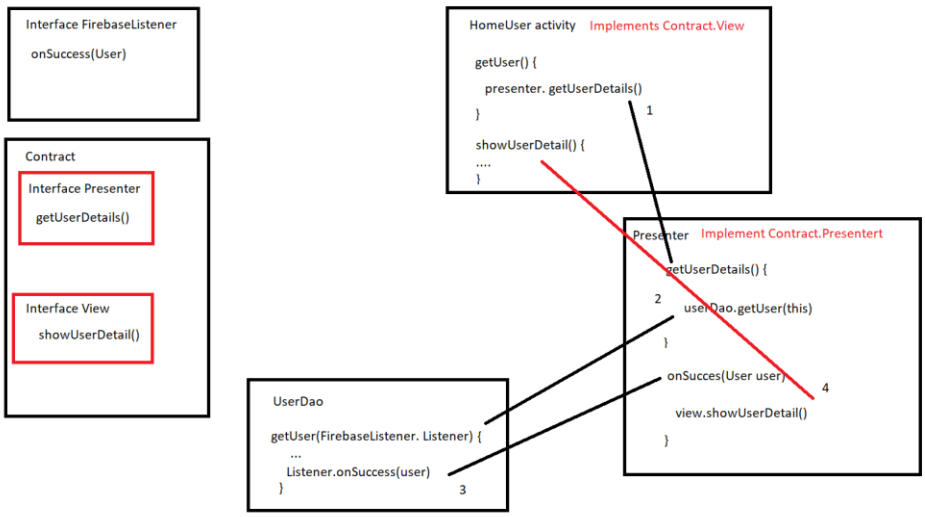
1. UserHomeActivity class
2. PresentHomeUser class
3. HomeUserContract

```java
HomeUserContract.java
package com.ger.garage.Presenter;

public class HomeUserContract {


    public interface Presenter {

        void detach();
        void logOut();
        void getUserDetails();

    }


    public interface View{
        void showUserDetail(String name, String email, String mobilePhone);
    }

}
```

View and Presenter interfaces are implemented by UserHomeActivity class and PresenterHomeUser class respectively. After this, they know each other. View can call methods from Presenter interface and presenter can call methods from View interface.

4. UserDao class
5. Firebase Listener: This interface is implemented by the presenter and when presenter call UserDao's getUser(FirebaseListener) method , the instance of a presenter is passed as actual parameter to the method. The result is received by this instance.

The cycle is represented below with these five entities.




I also must add a flag when I call to others screen with intent class because if I click a button twice two instances of a screen are open. One of the possible solutions that could work is FLAG_ACTIVITY_CLEAR_TOP which reuse activities if these one is in the activity stack/task.

Change profile should be implemented in the same way calling a method from presenter with the information to modify. Then update database, tell it the presenter and presenter must inform this to the screen.

## Make a Booking functionality / MakeABookingActivity Class

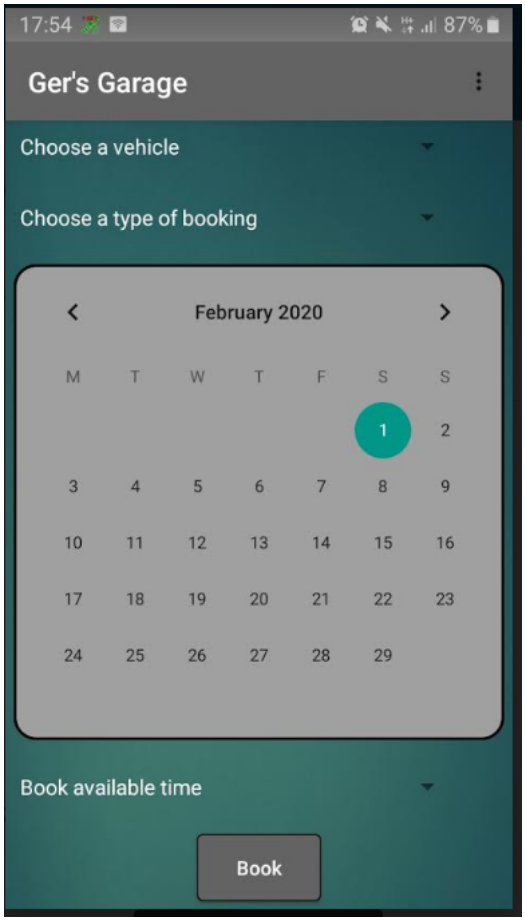
This functionality is implemented by the following group of class:

1. MakeABookingActivity class (view)
2. PresenterMakeABooking (presenter)
3. BookingDao (model)
4. MakeABookingContract (contract)

```
package com.ger.garage.Presenter;

import ...

public class MakeABookingContract {


    public interface Presenter {

        void detach();
        void logOut();
        void getVehicles();
        void getShifts(String typeOfBooking, LocalDate date);
        void getTypeOfBooking();
        boolean isWorkingDay(DayOfWeek day);
        void book(String vehicle, String typeOfBooking, LocalDate createdAt, String shift);
    }


    public interface View{

        void showVehicles(ArrayList<String> vehicles);
        void showTypeOfBooking(ArrayList<String> typeOfBookings);
        void showShiftsAvailable(ArrayList<String> shiftAvailable);
        void showErrorMessage(String errorMessage);
        void showSuccessMessage(String bookingId);
    }

}
```

5. FirebaseException (exception throwed by BookingDao when there is an error)
6. FirebaseListener (Listener to implement handlers to get result from database in presenter)

This is the screen which allow a user to make a booking




To get vehicles and Book Available time (spinners) we do same as for previous functionalities/screens call presenter, this one calls to UserDao and BookingDao classes respectively. These classes query Firestore database and implements the listener to wait for the result. When the result is got by the application, the Dao class calls presenter class. Lastly, presenter call the view to update the result on the screen.

Validations are made on screen:

1. If user touch Book available time spinner to choose a shift, the app validates both vehicle and type of booking spinner are set and date on calendar is not before than today.
2. It must be a working day (get working day from setup class – customizable parameter)
3. Same validations when user wants to book

On the other hand, if user select a shift, but after that modify type of booking or date in the calendar, the shift is blank, so users must be select a new shift.

Spinner works with arrayAdapter<T>.

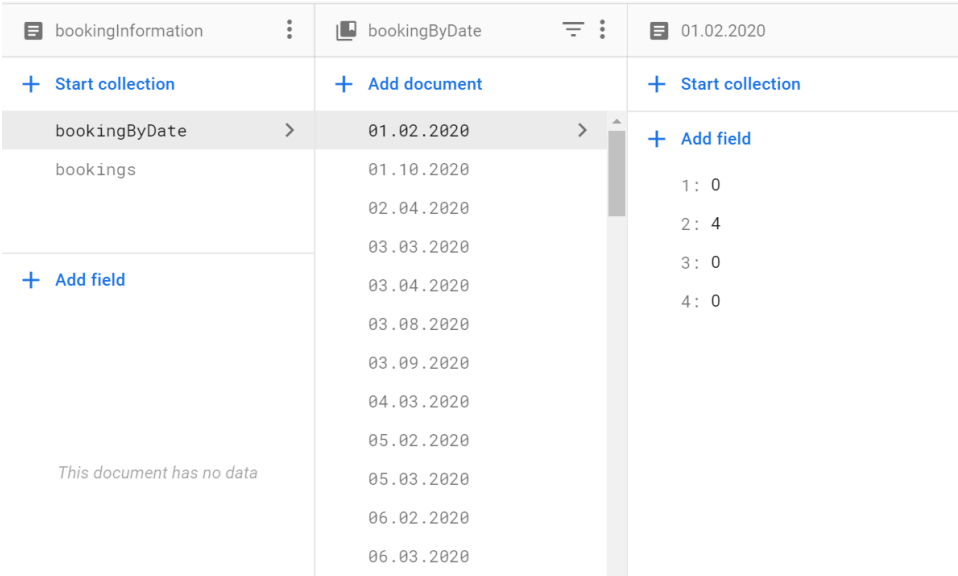
I must add a pop up (Dialog) to allow a user to add comments, these comments are added to the data that will be sent to the database to register the booking.

Book method in PresenterMakeABooking class, creates a booking with all the information collected in the screen. If user choose mayor repair (occupy 2 shifts), app creates a list with 2 shifts and assign this to the booking object. Then, createBooking method which is in BookingDao class is called.

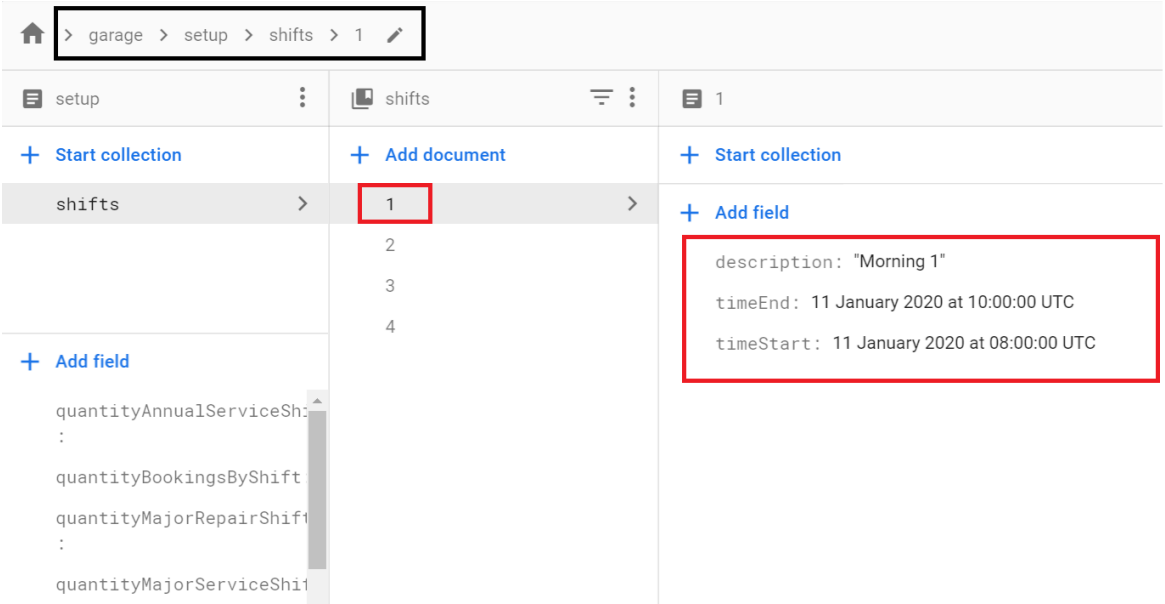
This method is crucial because this one use an operation provided by Firestore which allow execute atomic operations. The logic to book was developed and thought when I created the sequence diagram. This is very important if two users want to book at the same time, or the status cancelled (which was though and added to machine state machine diagram) because in this case the number of booking is incremented or decremented. This solution would be essential if we want to implement cancelled status.

How does createbooking method work?

There is a document saved in the database which has 4 fields, one for each shift (I have 4 shifts), and the id of the document is a date (04.03.2020). For example, image below, shows the document 01.02.2020 which has 4 fields: 1,2,3 and 4. Every number is the id of a shift. The shift 2 has already 4 bookings allocated. So, shift 1,2 and 4 are available. The link of these documents is garage/bookingInformation/bookings/bookingByDate.



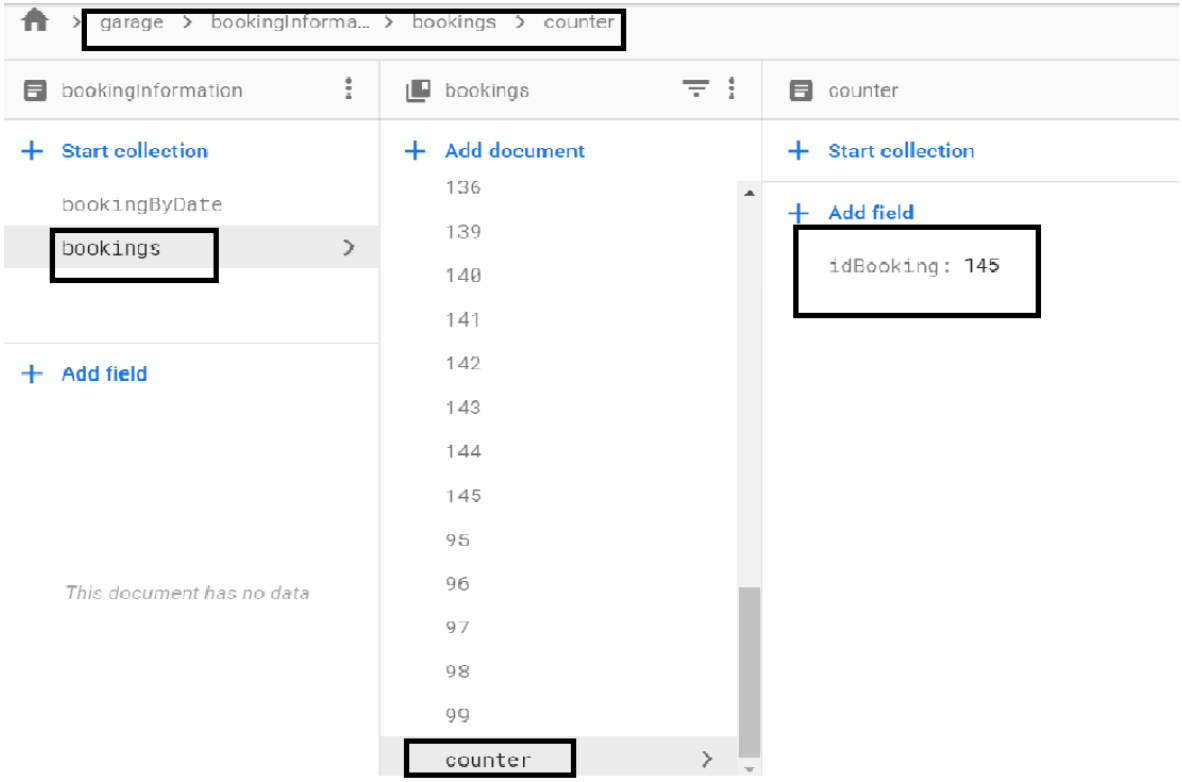

Shifts are saved in the link garage/setup/shifts (customizable parameters):




SetUpDao class is getting this information from memory (I created this values in SetUpDao class), but I want to read this information from Firestore and keep it in the memory. We could save these values, and all values customizables created in SetUpDao class, in a cache and then this information could be used by different classes.

```
// create booking with all the data
public void createBooking(final Booking booking, final Integer quantityTotalBookingByShift, final FirebaseListener listener) {

    // doc ref to get counter for new id
    final DocumentReference counterDocRef = db.collection(bookingsCollectionPath).document(counterDocument);

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.YYYY");

    final String dateAux = formatter.format(booking.getDate());

    // check again if there are space for the booking
    final DocumentReference dateRef = db.collection(bookingByDateCollectionPath).document(dateAux);

    // run atomic operation called transaction
    db.runTransaction(new Transaction.Function<Integer>() {


        @Nullable
        @Override
        public Integer apply(@NonNull Transaction transaction) throws FirebaseFirestoreException {

            final String idBookingCounter = "idBooking";
            final String listOfBookings = "ListOfBookings";

            // Read
            DocumentSnapshot counterSnapshot = transaction.get(counterDocRef);

            // get counter last id
            Integer counter = Integer.parseInt(String.valueOf(counterSnapshot.getLong(idBookingCounter)));
```

First part of this algorithm defines and declare a reference to a counter in the database (see image below)




This counter is used for the id of the booking. It is the next id available.

Then I define and declare another link to a garage/bookingInformation/bookings/bookingByDate which has dates and the quantity of bookings. So, source code gets the reference to the data of the booking to know if there are shifts available. Remember, when user select the shift on the screen, it had already been verified, but we need to do a second check just in case another user took last slot for that date/shift.

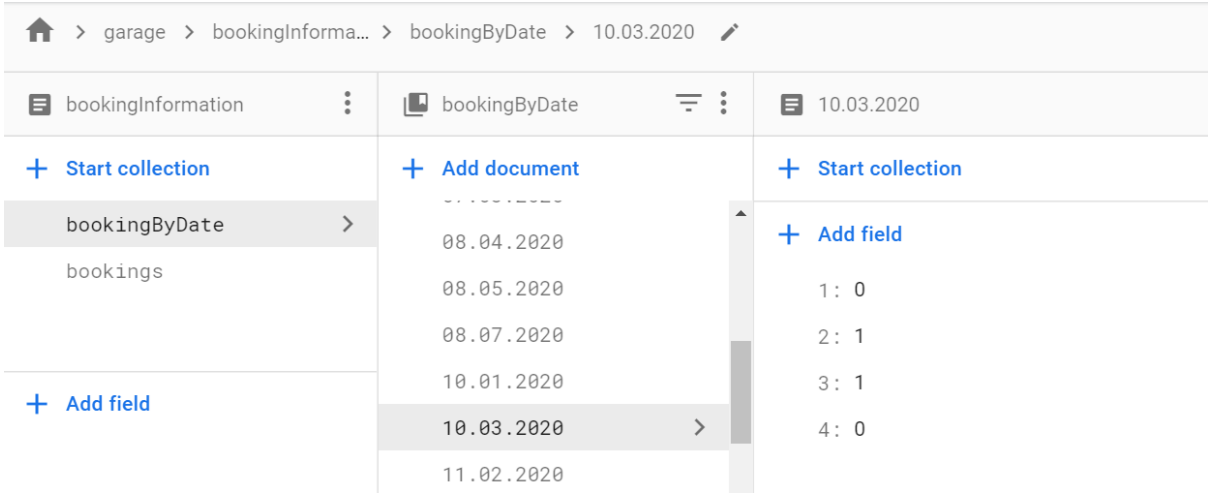
After, the transaction operation, which ensure atomicity, is executed. First, get the counter to increment it and get the new id. Second, the application checks if the shift/s is still available (mayor repair occupies two shifts). The image below shows how to check If the shift/s selected are available.

```
Integer shiftQuantity;

// check every shift that belongs to the booking has space
for (Shift shift: booking.getShifts()) {

    shiftQuantity = Integer.parseInt(String.valueOf(dateSnapshot.getLong(shift.getId().toString())));

    // current quantity booking is less than quantityTotalBookingByShift
    if (shiftQuantity == quantityTotalBookingByShift) { // abort retry operation

        throw new FirebaseFirestoreException(errorMsg1 + shift.toString() + errorMsg2,
        FirebaseFirestoreException.Code.ABORTED);

    }

}
```

Fourth, the application starts to write on the database (see image below).

```
// Write counter incremented
transaction.update(counterDocRef, idBookingCounter, counter);

//update quantity of the shift plus 1 for that day garage/bookingInformation/bookingByDate
for (Shift shift: booking.getShifts()) {

    shiftQuantity = Integer.parseInt(String.valueOf(dateSnapshot.getLong(shift.getId().toString()))) + 1;

    transaction.update(dateRef, shift.getId().toString(), shiftQuantity);

}

// write information about bookings in the links where bookings are
DocumentReference bookingsDocRef = db.collection(bookingsCollectionPath).document(booking.getId().toString());


Map<String, Object> bookingToSave = createDataBookingToSave(booking);

transaction.set(bookingsDocRef, bookingToSave);

// write booking inside of the user subtree
transaction.set(db.collection(bookingByUserCollectionPath).document(booking.getUser().getId()).collection(listOfBookings).document(booking.getId().toStri

return counter;
```

The source code tries to write the counter incremented on the database, then write the quantity of the shifts incremented by one. If the date of booking is 10.03.2020 and the user select shift 3, the application will try to set this value in garage/bookingInformation/bookings/bookingByDate/10.1.2020/3 (has value 1) to 2.




Lastly, the new booking is saved in garage/bookingInformation/bookings and in garage/userInformation/users/UID/ListOfBookings because we save the information twice, where all the booking is at, and below near user information.

There is one more important thing that I want to say about transaction operation. if there is a document which was change/update while the transaction was reading, this transaction is executed again.

If a client wants to book a shift but there is not more available slot for that shift because another client took this shift at the same time, the transaction operation to ensure consistent data, retries many times this set of statements and the book functionality takes more time to finish. An operation to stop the transaction does not exist. Other options to implement this can be:

1. Check before starting the transaction block if there is a slot available for that shift (probability is lower of this occurs, but it is not a good option)
2. Send a message to the user, these operations will take more time to finish (best solution)

3. After do a research on documentation online I realised that Realtime database (another firebase's database) implement a stop operation called abort

Anyway, from my point of view this solution is feasible and work if we want to add a new status such as Cancelled.

## Change Status and Allocate Mechanic functionalities

They were implemented in ListOfBookingsActivity and AllocateMechanicsActivity respectively. These functionalities were implemented following MVP pattern design.

Change Status and Allocate Mechanic used another atomic operation called batch which is only used when we just must write on the database. Change status and allocate mechanic update a booking at garage/bookingInformation/bookings and at garage/userInformation/users/UID/ListOfBookings with a new mechanic. There are two modifications and this update must be done all or no occur. The code below shows how we prepare both links (garage/bookingInformation/bookings/idBooking and garage/userInformation/users/UID/ListOfBookings/idBooking) to save mechanic using a hashMap/

```
// change status of a booking
public void changeStatus(Booking booking, final String newStatus, final FirebaseListener2 listener) {

    final String listOfBookings = "ListOfBookings";
    final String statusField = "status";

    // batch atomic operation
    WriteBatch batch = db.batch();

    // change status in the booking subtree
    DocumentReference bookingRef = db.collection(bookingsCollectionPath).document(booking.getId().toString());
    batch.update(bookingRef, statusField, newStatus);
    // change status in the user subtree
    DocumentReference userRef = db.collection(bookingByUserCollectionPath).document(booking.getUser().getId()).collection(listOfBookings).document
    batch.update(userRef, statusField, newStatus);

    batch.commit().addOnCompleteListener((task) -> {

            if (task.isSuccessful())

                listener.onSuccess(newStatus);

            else

                listener.onFailure( new FirebaseException(errorUpdateStatus + task.getException().getMessage()));

    });
```

How status works from user point of view?

We guess Booking number 2 is in status "in service", and user wants to change to next status. The application checks current status and shows what next status user can go. In this case, user could go to "Unrepairable" or "fixed". So, I must add a new piece of code which shows a warning/error if the booking does not have a mechanic assigned and user decide if he wants to proceed. In general, this is up to the customer of the product.

How is the list status stored?

I created a graph using two data structure (HashMap and ArrayList).

```
private HashMap<String, ArrayList<String>> statusGraph;
```

Implementation,

```
// I created a graph of status using one Hasmap and arrayLists
// My plan is define methods related two status in this class
// For example can allocate mechanic method or allocate cost method? could be implemented here
// and uses this structure. Another option define this issue in Booking Clsss
statusGraph = new HashMap<>();
statusGraph.put("booked", new ArrayList<String>(Arrays.asList("in service")));
statusGraph.put("in service", new ArrayList<String>(Arrays.asList("unrepairable", "fixed")));
statusGraph.put("unrepairable", new ArrayList<String>());
statusGraph.put("fixed", new ArrayList<String>(Arrays.asList("collected")));
statusGraph.put("collected", new ArrayList<String>());
```

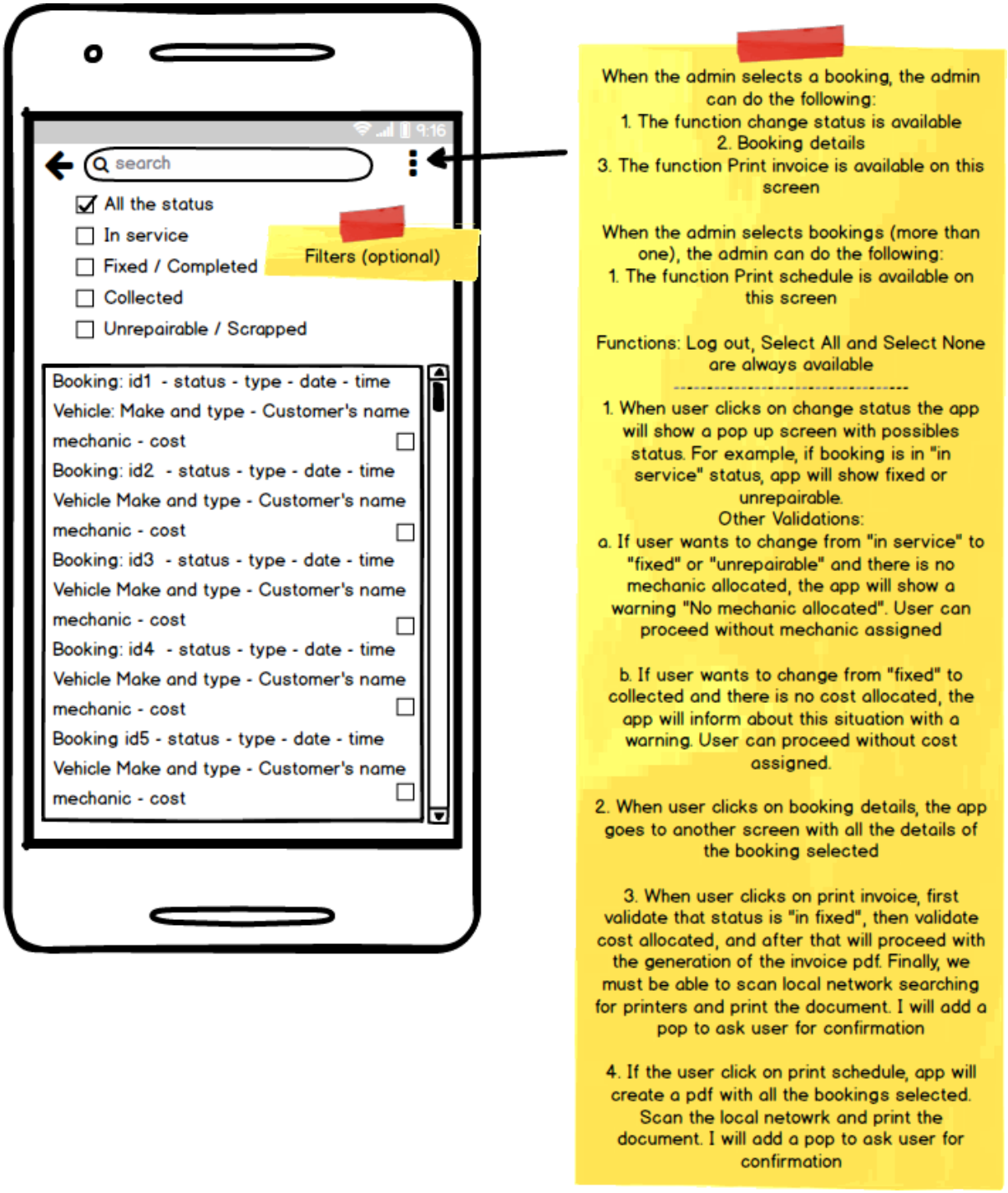
I could define method to define if a mechanic can be allocated or cost here. Everything related with status I'd like to implement in this class or create another class called status to manage this information. For example, another method could be canUserAllocateMechanic(Booking) which validate if this booking can receive a mechanic.

Another option could be in Booking class defining allocateMechanic() and allocateCost() methods but I'd like to implement this near database, just in case we can save all this information on Firestore which is a good idea.

AllocateMechanic functionality brings information of bookings from database with status "in service" or "booked" and allow a mechanic be assigned to a booking. Mechanics are created in SetUpDao classes, but I want to pass this information to Firestore.

Everything related with status was thought and developed in previous chapters (use case descriptions, wireframes and state machine diagram).

The following screen, which was developed in wireframes section in the previous chapter, is called ListOfBookingsActivity in our implementation.




In this screen I'm using a ListViewAdapter which provide a binding between data (booking and checkboxes) to the view/screen. I must check this source code because there is an error which is related when I refresh screen. When user change status, the MVP flow starts, it goes to presenter, presenter call a method changeStatus in BookingDao with new status, booking and FirebaseListener object. Then the listener is waiting for the result of the asynchronous function. When the result arrives, it goes back to the screen. The screen is refreshed but I always create a new ListViewAdapter, so the screen behaves strange because of this. This does not happen with Print Schedule functionality. So, I need to check if it is not null, don't create an arrayListView. Anyway, I'd like to do another method which receives update notification.

Select All and select none are not implemented, they should be implemented calling a notifyDataSetChanged() from ListViewAdapter and changed checkboxes arrayList with all the position checked. We can define a attribute/flag inside of ListViewAdapter class, and set it when we want to select all/none, and then called notifyDataSetChanged to update all the positions.With ArrayAdapter, we save memory recycling items, this method only creates the items we need to show on the screen.

Another important issue is, when I want to sort the result from database, first ordering must be on the same field.

You can combine `where()` filters with `orderBy()` and `limit()`. In the following example, the queries define a population threshold, sort by population in ascending order, and return only the first few results that exceed the threshold:

Web | Swift | Objective-C | Java Android | Kotlin Android | Java | Python | More

```
citiesRef.whereGreaterThan("population", 100000).orderBy("population").limit(2);
```

DocSnippets.java

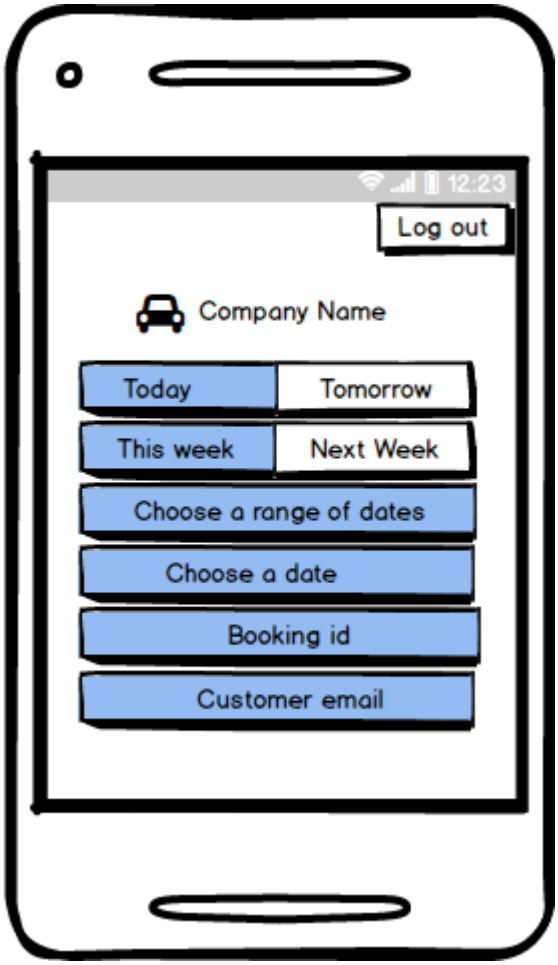
However, if you have a filter with a range comparison (`<`, `<=`, `>`, `>=`), your first ordering must be on the same field:

So, I'm implementing Comparato<Booking> to sort lists on ListOfBookingsActivity and AllocateMechanicsActivity.

ListOfBookingsActivity has four functionalities, print schedule, print invoice, change status and booking details. When I was doing this screen and its logic, I realised the source code was so complex. So, I decided to create another solution which was explained in the wireframes section. That solution (option 2) decouple functions, putting Print Schedule which is massive in another screen. I also think this solution is more user friendly.

## AdminDisplayBookingsActivity Class

The following screen was implemented in my app and is called AdminDisplayBookingsActivity




This class uses buttons, methods from the classes shown below in the image, a Cache and CacheImplementation classes and the Sky CalendarView library ([https://github.com/psinetron/slycalendarview](https://github.com/psinetron/slycalendarview)) to select a range of dates. I must put a validation to not allow user to select more than 3 weeks.

```
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.temporal.TemporalAdjusters;
import java.util.Calendar;
```

Lastly, CacheImplementation class, which implement Cache interface, use SharedPreferences to keep value in a form of key-value pair. Then, this object persist between classes to pass information and keep classes decouple without thread between them. It helps to test an application.

Choose a date (implement a date picker), id Booking and email have very similar functionality. User clicks on the button, save value in cache and call next screen. Next screen lifts the values from cache, call presenter and this one call BookingDao/UserDao class. This classes query database using get method from Firestore Library. We only must filter for one field, so we don't need to create an index. The Dao class informs presenter and so on until screen.

## Listeners on ListOfBookingsActivity, AllocateMechanicsActivity and DisplayBookingsActivity Class

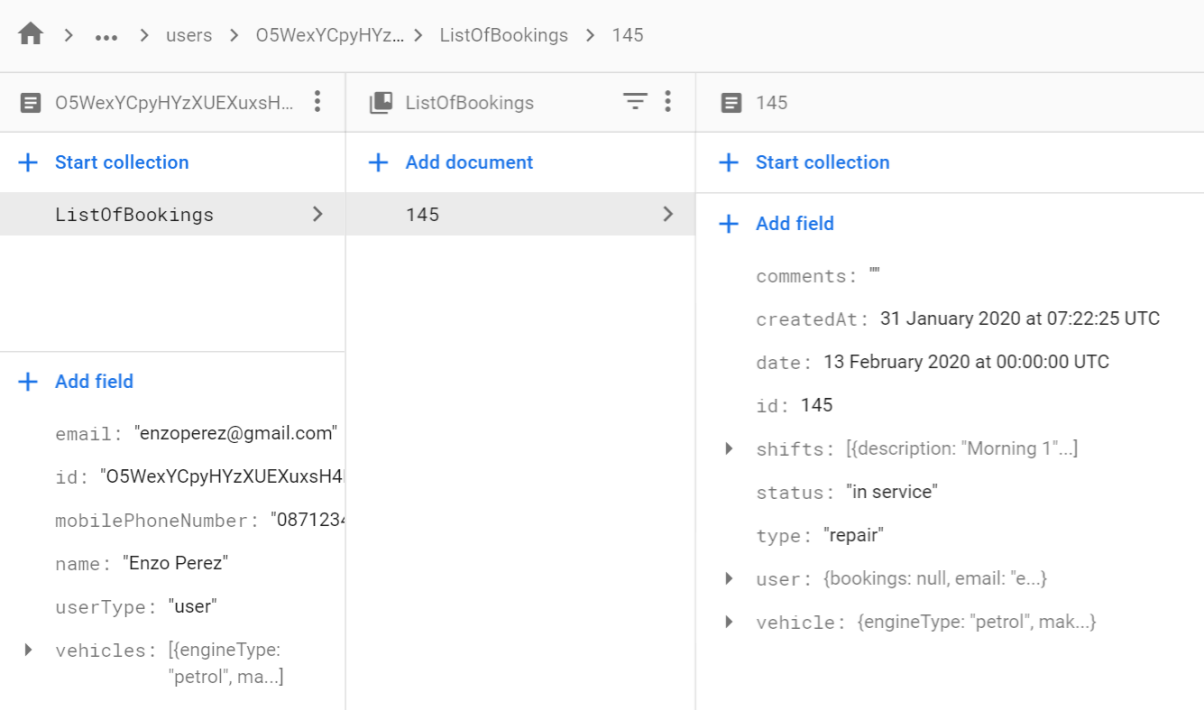
DisplayBookingsActivity shows a listView with bookings for the user. I must add the shift to every booking to give to user more details about the booking.

ListOfBookingsActivity and AllocateMechanicsActivity already were explained previously.

Three listeners were implemented in BookingDao which are called if the content of the reference changes. For example, we point to a document or set of documents in the database, if it changes, the register listener is called wit the new snapshot. Then BookingDao informs presenter and so on.

```java
// listener which list if there is a change in the subtree user
listenerBookingsByUser = colRef.addSnapshotListener(new EventListener<QuerySnapshot>() {
    @Override
    public void onEvent(@Nullable QuerySnapshot queryDocumentSnapshots, @Nullable FirebaseFirestoreException e) {

        if (e != null) {
            return;
        }

        ArrayList<Booking> bookings = new ArrayList<>();

        for (QueryDocumentSnapshot document: queryDocumentSnapshots) {


            bookings.add(buildBooking(document));

        }

        listener.onSuccess(bookings);

    }
});
```

For example: Enzo Perez user has a booking (145) in “in service”



So, the following screen shows current status.

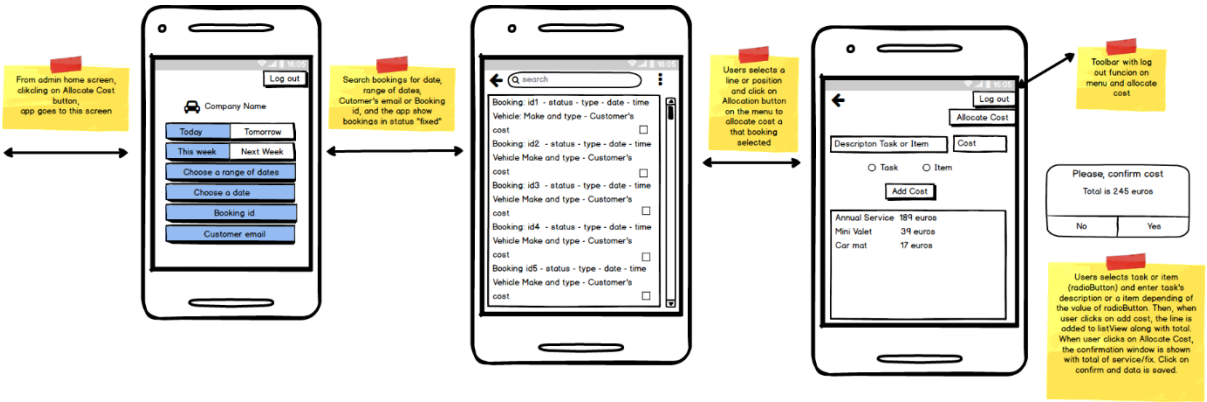
If we change it to “Fixed”, the screen is refreshed automatically without any action. Image below shows the screen updated.

Two more listener were implemented in BookingDao, but I didn't have time to test them properly.

I also wanted to add a search function on the toolbar to search/filter for different bookings. For example, in ListOfBookingsActivity.

## Allocate Cost Functionality

The followings image shows the sequence of screens from Admin Home Screen to allocate cost a booking.



To implement first screen, we can reuse the screen already implemented which is called **AdminDisplayBookingsActivity screen.** The only thing that we have to do is saved in the cache that we want to allocate cost. The rest of the methods do not change.

Second screen is a ListView with an ArrayAdapter. We must implement a listener to have the most updated information on the screen. We also add a button on the menu to allocate Cost when we select a booking. If we select more than one booking, the app shows an error informing to the user.

When user select a booking and click on Allocate Cost on the menu, the program takes us to the third screen. The data about booking can sent to the third screen using the cache. In the third screen, we must add two editTexts, one radio group with two radio buttons (item and task), one button called Add Cost, one button called Save/Allocate Cost on the menu and one ListView.

When user clicks on item raddioButton, he/she is able to add an item with a cost. Then user click on add Cost to add this line to the list of costs. When user click on Task raddioButton, user is able so add a task (it can be general task, for example Annual Service and the cost). Once that user allocates all the cost, he/she must go to the menu to save cost. Then, view call presenter (they know each other because there is a contract), presenter call a method in BookingDao and this one saves data in the database using Bath operation to update consistently. Then the listener is informed with the new data before the data is sent to the backend. This is called “latency compensation”. The new data is show on the screen. This must be implemented following MVP pattern.

## Generation PDF Print in ListOfBookingsActivity

The data about the booking is in a global variable in ListOfBookingsActivity class and we must display all the information about the booking to the administrator.

Sequence of steps to generate a pdf to print is the following:

1. I added permission to write in External Storage and define a provider to keep a file in our private folder
2. I added the following libraries in build.gradle:

```
ayBookingsActivity.java   AndroidManifest.xml   app
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.firebase:firebase-analytics:17.2.0'
    implementation 'com.google.firebase:firebase-firestore:21.3.1'
    implementation 'com.google.firebase:firebase-auth:19.2.0'
    implementation 'com.google.android.material:material:1.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
    implementation 'com.github.psinetron:slycalendarview:0.0.9'

    // Lib pdf toolkit
    implementation 'com.itextpdf:itextg:5.5.10'
    // Dexter is an Android library that simplifies the process of requesting permissions at runtime.
    implementation 'com.karumi:dexter:6.0.1'

}

apply plugin: 'com.google.gms.google-services'
```

3. Then we implement Dexter to ask user for permissions at runtime

4. If user allow app to write, we create the pdf using itextPdf library
5. We create a common class to create a directory, if it was not created
6. Lastly, we use PrintManager to access to printing features of Android

In the folder schedule I left a copy of the pdf generated.

## Print Invoice

The invoice functionality reused the algorithms used by Print Schedule. Information about booking is in a global variable in ListOfBookingsActivity class.

Lastly, I want to add that I set portrait orientation in Manifest for MakeABookingActivity, UserHomeActivity, LoginActivity and RegisterActivity classes. I didn't have time to set the other screens.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Ger's Garage"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".view.AllocateCostActivity"></activity>
    <activity android:name=".view.AllocateMechanicActivity" />
    <activity android:name=".view.ListOfBookingsActivity" />
    <activity android:name=".view.AdminDisplayBookingsActivity" />
    <activity android:name=".view.AdminHomeActivity" />
    <activity android:name=".view.DisplayBookingsActivity" />
    <activity
        android:name=".view.MakeABookingActivity"
        android:screenOrientation="portrait" />
    <activity
        android:name=".view.UserHomeActivity"
        android:screenOrientation="portrait" />
    <activity
        android:name=".view.LoginActivity"
        android:screenOrientation="portrait" />
    <activity
        android:name=".view.RegisterActivity"
        android:screenOrientation="portrait">
```

# Chapter 5: Testing and Evaluation

I'd have liked to use LeakyCanary to verify if the app is leaking and do performance tests but I did not have time. On the other hand, I also wanted to use Mockito to do unit test on classes because I wanted to take advantages of MVP implementation.

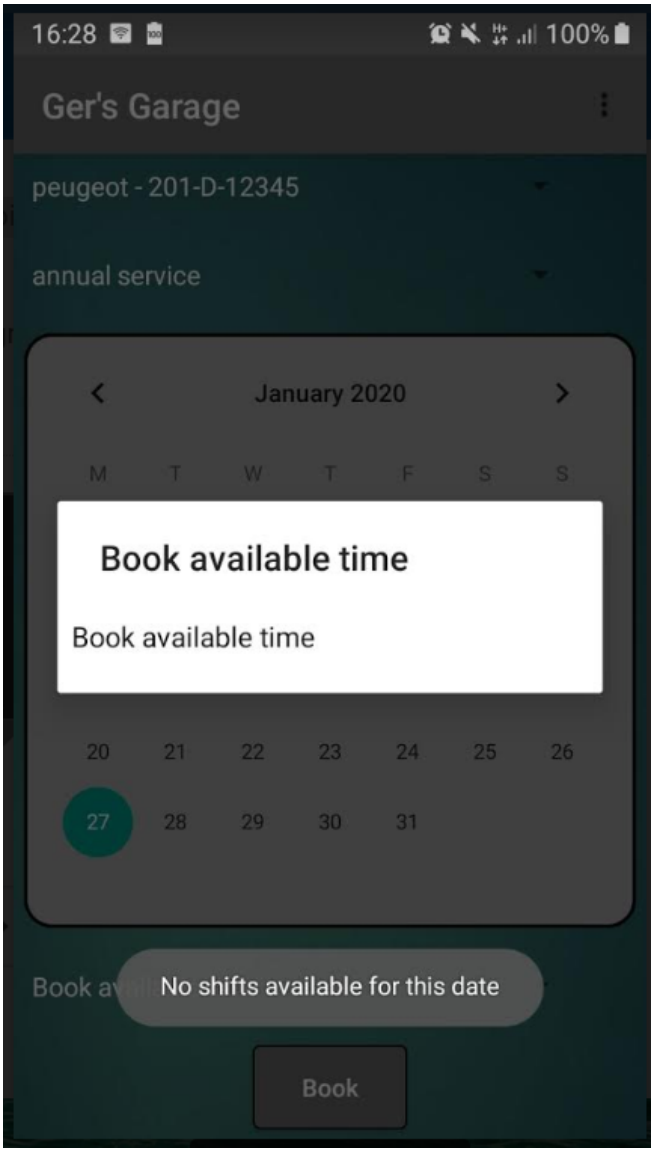
I put two files where I registered different test I've been doing on the application. I divided the testing phase in two parts. First part is for user screens and second one for admin screens. I put them in the folder testing. In Every part, I also tested every screen individually taking advantage of the MVP implementation. When I finished every screen, I did an integration testing of the current screen plus other screens.

There are more tests in the following link:

https://drive.google.com/open?id=13EGe6DXS6oih3ui2FyfAj9yrXShXR5_H

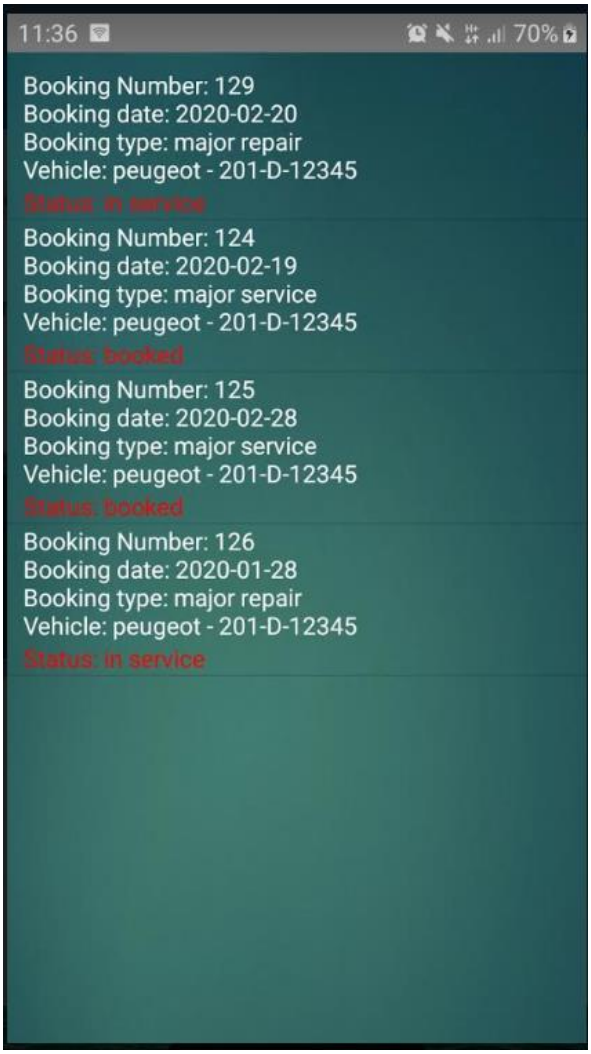
Some other tests:

1. Test 27/01/2020 at 4:30pm the application does not show available shifts because last one was at 3pm.

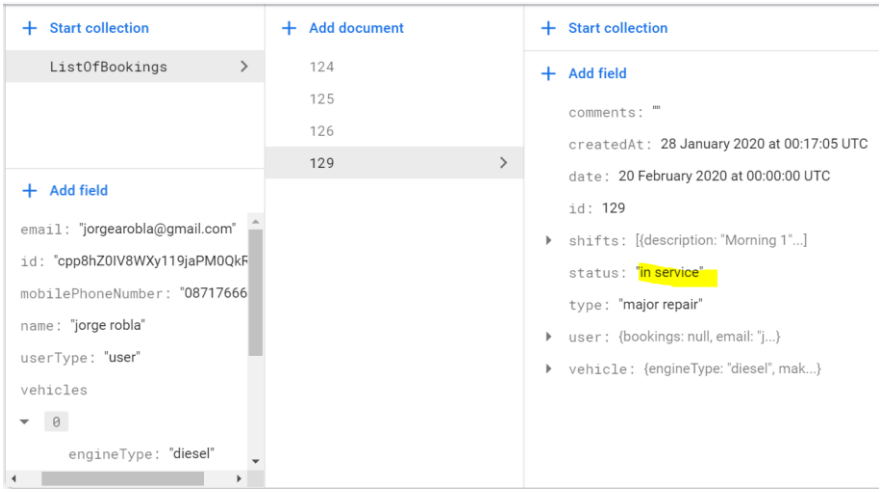



User can book at the same day depending on the time. If user wish to book at a time which is before that current one, the app shows an error.

2. Display booking for user shows idBooking number 129 with status “in service”

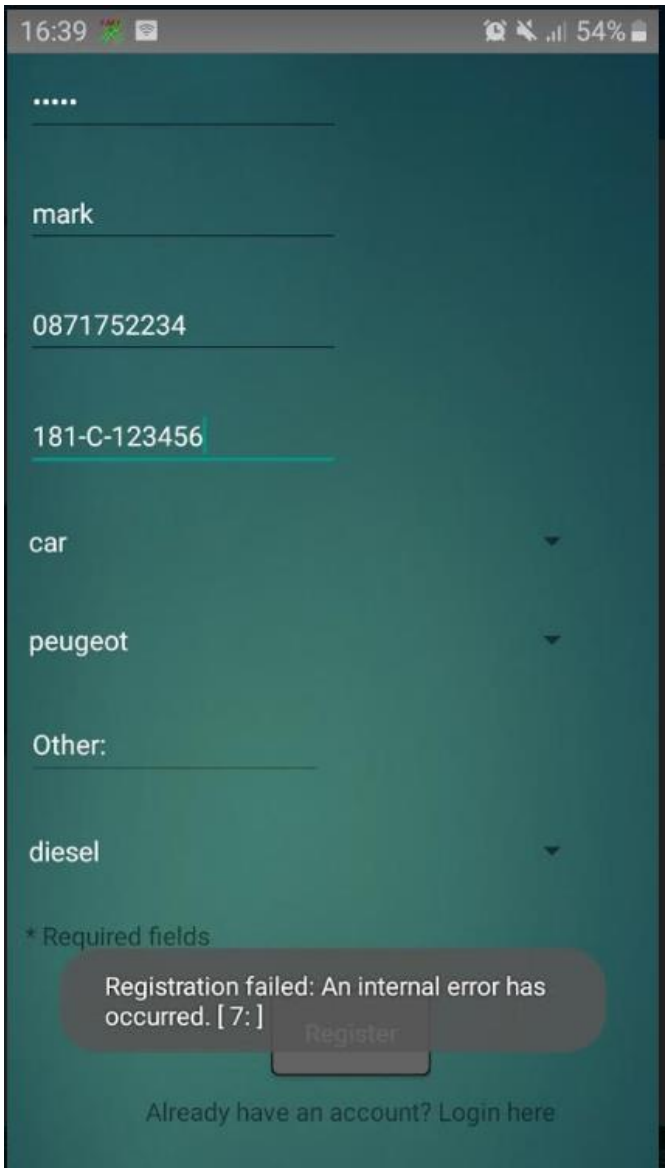



In the database we can see same status:

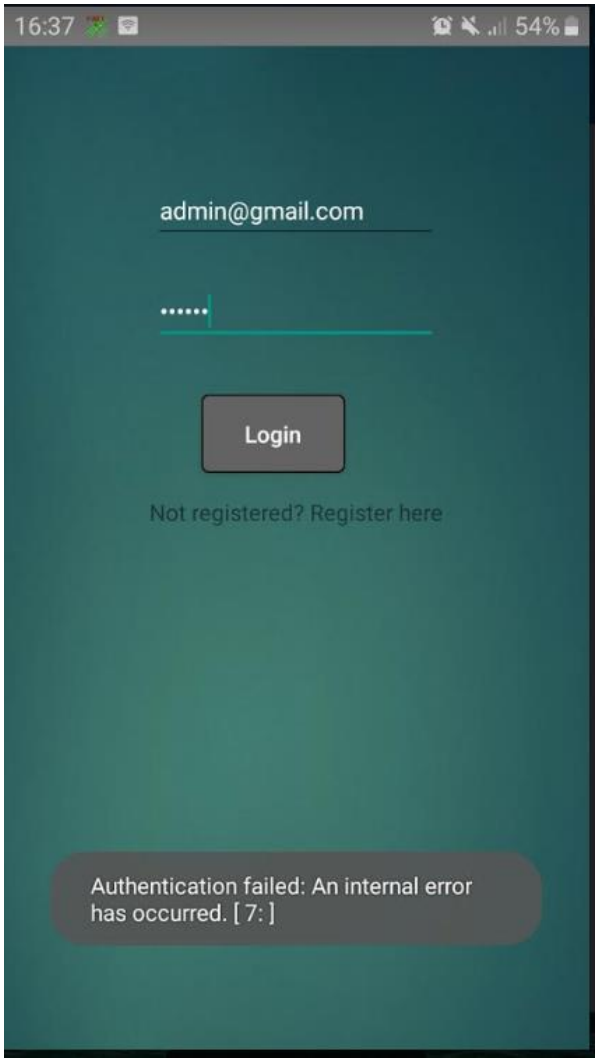


3. Test Login and register without internet connexion

Register Screen



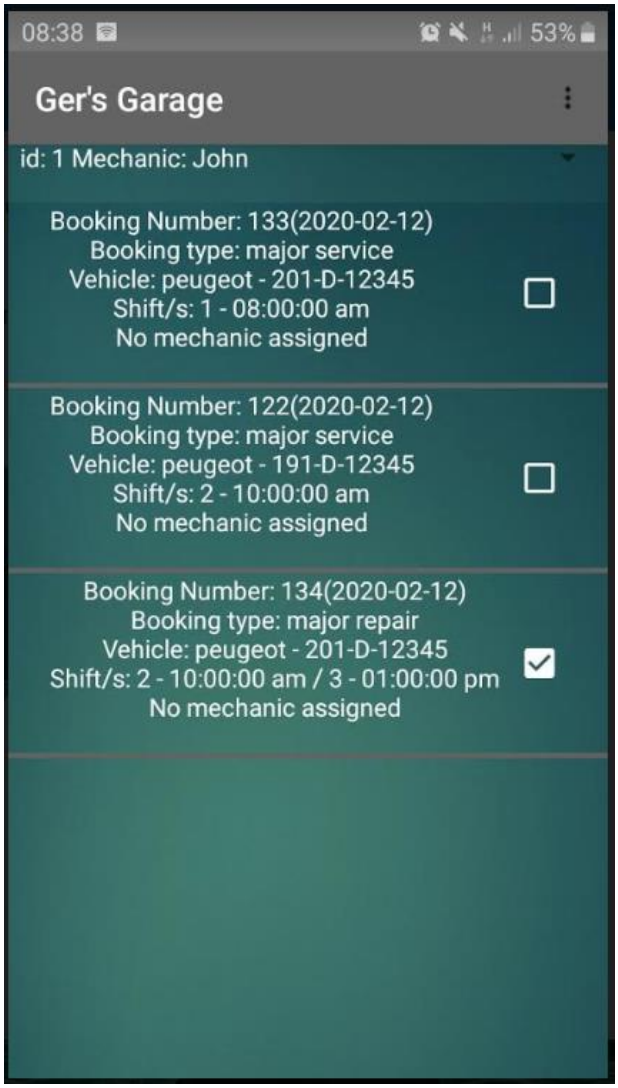

Login Screen

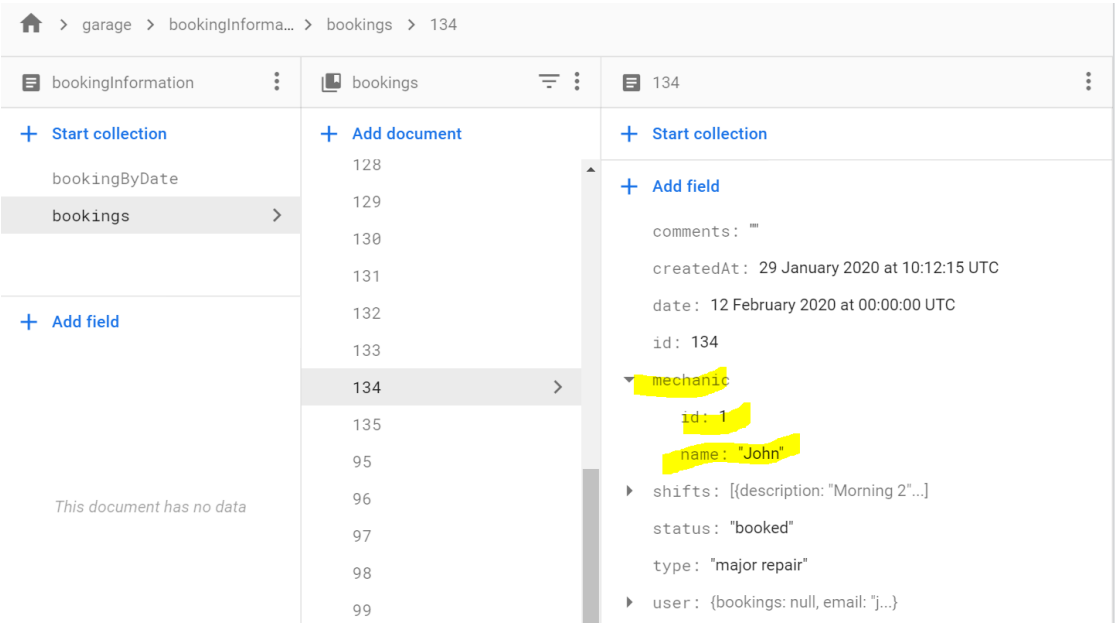



I must catch exception, which is thrown by Firebase Library and show another message to the user.

I didn't have time to check when the mobile application does not have internet for the rest of the screens.

4. Testing in allocate mechanic screen


08:38
53%
Ger's Garage
id: 1 Mechanic: John
Booking Number: 133(2020-02-12)
Booking type: major service
Vehicle: peugeot - 201-D-12345
Shift/s: 1 - 08:00:00 am
No mechanic assigned
Booking Number: 122(2020-02-12)
Booking type: major service
Vehicle: peugeot - 191-D-12345
Shift/s: 2 - 10:00:00 am
No mechanic assigned
Booking Number: 134(2020-02-12)
Booking type: major repair
Vehicle: peugeot - 201-D-12345
Shift/s: 2 - 10:00:00 am / 3 - 01:00:00 pm
No mechanic assigned


garage > bookingInforma... > bookings > 134
bookingInformation
Start collection
bookingByDate
bookings
Add field
This document has no data
bookings
Add document
128
129
130
131
132
133
134
135
95
96
97
98
99
134
Start collection
Add field
comments: ""
createdAt: 29 January 2020 at 10:12:15 UTC
date: 12 February 2020 at 00:00:00 UTC
id: 134
mechanic
id: 1
name: "John"
shifts: [{description: "Morning 2"...]
status: "booked"
type: "major repair"
user: {bookings: null, email: "j...}

# Conclusion

After I've been working for nearly two months, I can say the project was too long for the time I had. I spend between three and four weeks analysing the brief/functionalities and working in the design of the software. I liked the process I designed using status and we can generate documents in the middle of the process (schedule and invoice pdfs). I followed and respected the development cycle how I had planned at the beginning doing my best at the end when I did the testing. I just needed more time to finish the applications and to do more consistent tests.

My application is interesting because I used products such as Firestore, Firebase Authentication, android SDK which use modern concepts (for example listeners, database in cloud, etc).

To conclude I'd like to say I'm happy with my project because I learned a lot about android applications and Firestore and I'd like to continue working in mobile development.

Further work:

1. Ability to send an-email/SMS when the vehicle is serviced/fixed.
2. Ability to check the store if there are parts/items to fix/service the vehicle and allow a user to make a booking.
3. User manual to explain users how the system work
4. Extra functionalities. The admin could upload an excel document to allocate mechanic massive way
5. Ability to scrape a webpage to get items/parts of the vehicle
6. Add a new status called cancelled
7. Allow to user login with the phone number
8. Read plates to change status from booked to in-service using the Machine Learning (google's API). The app will show a screen when user can confirm the vehicle is in the garage
9. Define rules security (very important)

# Appendix A

Repository where source code is maintained:

https://github.com/Jorge36/Garage

# List of References


1. Medium. (2020). Android API Level, backward and forward compatibility. [online] Available at: https://android.jlelse.eu/android-api-level-backward-and-forward-compatibility-10e6d31cb848 [Accessed 1 Feb. 2020].
2. Black, M. (2020). Essential PH-1 is first non-Pixel phone to get Android Pie. [online] Tech Advisor. Available at: https://www.techadvisor.co.uk/news/google-android/android-pie-news-3670552/ [Accessed 1 Feb. 2020].
3. GeeksforGeeks. (2020). Top Programming Languages for Android App Development - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/ [Accessed 1 Feb. 2020].

4. MobiLoud Blog. (2020). Native, Web or Hybrid Apps? What's The Difference?. [online] Available at: https://www.mobiloud.com/blog/native-web-or-hybrid-apps/ [Accessed 1 Feb. 2020].
5. SearchMobileComputing. (2020). What is Android Studio? - Definition from WhatIs.com. [online] Available at: https://searchmobilecomputing.techtarget.com/definition/Android-Studio [Accessed 1 Feb. 2020].
6. Agiledata.org. (2020). Introduction to Object-Orientation and the UML. [online] Available at: http://www.agiledata.org/essays/objectOrientation101.html#DifferentCoreDiagrams [Accessed 1 Feb. 2020].
7. YouTube. (2020). UML Use Case Diagram Tutorial. [online] Available at: https://www.youtube.com/watch?v=zid-MVo7M-E [Accessed 1 Feb. 2020].
8. Fakhroutdinov, K. (2020). Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc. [online] Uml-diagrams.org. Available at: https://www.uml-diagrams.org/ [Accessed 1 Feb. 2020].
9. Pmi.org. (2020). Use Cases - what Every Project Manager Should Know. [online] Available at: https://www.pmi.org/learning/library/use-cases-project-manager-know-8262 [Accessed 1 Feb. 2020].
10. YouTube. (2020). L04 Use Case Description. [online] Available at: https://www.youtube.com/watch?v=i3dg99MWLZU [Accessed 1 Feb. 2020].
11. Careerfoundry.com. (2020). The Definitive Guide: How To Create Your First Wireframe. [online] Available at: https://careerfoundry.com/en/blog/ux-design/how-to-create-your-first-wireframe/ [Accessed 1 Feb. 2020].
12. Medium. (2020). Wireframe, Mockup, Prototype: What is What?. [online] Available at: https://uxplanet.org/wireframe-mockup-prototype-what-is-what-8cf2966e5a8b [Accessed 1 Feb. 2020].
13. Cyblance. (2020). Importance of Wireframing in analysis Phase and Different tools for Drawing Wireframes - Cyblance. [online] Available at: https://www.cyblance.com/custom-software/importance-of-wireframing-in-analysis-phase-and-different-tools-for-drawing-wireframes/ [Accessed 1 Feb. 2020].
14. Visual-paradigm.com. (2020). What is Sequence Diagram?. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ [Accessed 1 Feb. 2020].
15. YouTube. (2020). UML Use Case Diagram Tutorial. [online] Available at: https://www.youtube.com/watch?v=zid-MVo7M-E [Accessed 1 Feb. 2020].
16. Visual-paradigm.com. (2020). How to Draw a State Machine Diagram in UML?. [online] Available at: https://www.visual-paradigm.com/tutorials/how-to-draw-state-machine-diagram-in-uml/ [Accessed 1 Feb. 2020].
17. The Firebase Blog. (2020). Scheduling Cloud Functions for Firebase (cron). [online] Available at: https://firebase.googleblog.com/2019/04/schedule-cloud-functions-firebase-cron.html [Accessed 1 Feb. 2020].
18. Firebase. (2020). Schedule functions | Firebase. [online] Available at: https://firebase.google.com/docs/functions/schedule-functions [Accessed 1 Feb. 2020].
19. GitHub. (2020). psinetron/slycalendarview. [online] Available at: https://github.com/psinetron/slycalendarview [Accessed 1 Feb. 2020].
20. Lucidchart.com. (2020). [online] Available at: https://www.lucidchart.com [Accessed 1 Feb. 2020].

21. Sciencedirect.com. (2020). Case Description - an overview | ScienceDirect Topics. [online] Available at: https://www.sciencedirect.com/topics/computer-science/case-description [Accessed 1 Feb. 2020].
22. YouTube. (2020). Mobile App Wireframe sketch using Balsamiq | UX Design | Wireframe Tutorial | Wireframe sketch. [online] Available at: https://www.youtube.com/watch?v=ddsOn_gOLQc [Accessed 1 Feb. 2020].
23. BigLeap Solutions. (2020). Domain Model - Part A. [online] Available at: https://www.youtube.com/watch?v=M1e2XwSADDE&t=22s [Accessed 1 Feb. 2020].
24. LucidChart. (2020). How to Make a UML Sequence Diagram. [online] Available at: https://www.youtube.com/watch?v=pCK6prSq8aw [Accessed 1 Feb. 2020].
25. YouTube. (2020). Batch Import Images for Different Screen Resolutions - Android Studio Tutorial. [online] Available at: https://www.youtube.com/watch?v=bbFLDR6X2D8 [Accessed 1 Feb. 2020].
26. Describing Use-Case Relationships, with Sequence Diagrams. By JESUS M. ALMENDROS-JIMENEZ AND LUIS IRIBARNE. Departamento de Lenguajes y Computacion, Information Systems Group, University of Almeria, 04120 Almeria, Spain. The Author 2006. Published by Oxford University Press on behalf of The British Computer Society. All rights reserved. Advance Access published on September 20, 2006
27. Anon, (2020). [online] Available at: https://dev.to/iriskatastic/top-6-software-development-methodologies-9b [Accessed 1 Feb. 2020].
28. Agilenutshell.com. (2020). What is Agile?. [online] Available at: http://www.agilenutshell.com/ [Accessed 1 Feb. 2020].
29. Medium. (2020). Wireframe, Mockup, Prototype: What is What?. [online] Available at: https://uxplanet.org/wireframe-mockup-prototype-what-is-what-8cf2966e5a8b [Accessed 2 Feb. 2020].
30. Agilemodeling.com. (2020). UML 2 State Machine Diagrams: An Agile Introduction. [online] Available at: http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm [Accessed 2 Feb. 2020].
31. Getaround Tech. (2020). Use Android's FileProvider to get rid of the Storage Permission | Getaround Tech. [online] Available at: https://getaround.tech/android-fileprovider/ [Accessed 2 Feb. 2020].
32. Memorynotfound. (2020). Create PDF Document with iText in Java. [online] Available at: https://memorynotfound.com/create-pdf-document-itext-java/ [Accessed 2 Feb. 2020].
33. Firebase. (2020). Cloud Firestore | Firebase. [online] Available at: https://firebase.google.com/docs/firestore [Accessed 2 Feb. 2020].
34. Android Developers. (2020). Documentation | Android Developers. [online] Available at: https://developer.android.com/docs [Accessed 2 Feb. 2020].