Chalmers University of Technology

OOP Projekt

Requirements and Analysis Document (RAD)

Rent A Car - Stulb

Albert W, Hannes T, Jamal M, Johan S, Josef N

24/10-2021
v.2.0

1. Introduction

The project aims towards making renting cars easier and better by providing one platform all over the world. A platform where users can both rent out their own private car, rent someone's private car or for companies to reach out to more and new customers through an application easy to use for all. In today's society owning your own car has become less and less convenient where different circular car owning options have become more popular. This has created a need for an app where you can easily rent a car for a shorter amount of time like a day or a longer time if needed. By enabling private persons and companies to list their cars we create a competitive market where the users are made sure to get the best prices and lowest fares. Part of the purpose is to make car-sharing better and more convenient so that less people will be in need of buying a car so that we can lower the amount of lighthouse gas being emitted into the atmosphere.

There are a lot of beneficiaries of Rent a Car with foremost people who are in need of renting a car but also car rental firms and people that own one or multiple cars which are not used all the time. It will facilitate for both domestic and foreign travellers as the app could be used worldwide and therefore gathering cars to rent in one platform. Another one of the most important beneficiaries is the climate, which through reduced new production of cars will avoid the cost of more emissions into the atmosphere.

1.1 Definitions, acronyms, and abbreviations

The following is a list of words used throughout this document and their explanations:

GUI ”Graphical User Interface” also ”User Interface”, the part of the program that the user sees and interacts with.

User Story short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

API ”Application Programming Interface”, an interface or communication proto-col between a client and a server intended to simplify the building of client-side software.

Hi-Fi ”High fidelity”, a sketch with much detail.

UML ”Unified Modeling Language”, A visual language for mapping and repre-senting systems. Can be used for modelling och class diagrams, sequence diagrams and more. In this project used for Domain and design model.

DoD "Definition of Done", a list of requirements that have to be met for a User Story to be considered done.

2. Requirements

2.1 User Stories

This subsection lists the user stories that define the application’s requirements, in terms of the end-user’s wanted functionality.

2.1.1 UserStory1: Epic - Done

Story Name: User Story - Epic

Description:

(Epic) As a User, I need to be able to see the cars are available to rent so that I can rent a car to do what I need a car to travel.

Confirmation:

The user can see a list of available cars, choose one and then rent it.

2.1.2 UserStory2: Done

Story Name: Search

Description:

As a User, I need a search feature so that I can search and iterate between different types of cars.

Confirmation:
- The user can Search for specific types of cars.
- The user can Search for different car brands.

2.1.3 UserStory3: Done
Story Name: Account Page

Description:
As a car rental firm, We need a page so that we can overlook our account and our ads so that we know what we are renting out.

Confirmation:
- The user can scroll through his ads.
- The user has one page for his account where he can see and change his information.

2.1.4 UserStory4: Done
Story Name: Car Owner

Description:
As a user, I want to be able to put my car up for rent so that I can make money when I don't use my car.

Confirmation:
- The user can advertise his car
- The user can add images to the ad
- The user can set a price for the rental
- The user can post their contact information
- The user can add an item

2.1.5 UserStory5: Done

Story Name: Customer Order

Description:
As a Customer, I need to place an order so that I know if I can rent a car.

Confirmation:
- The user can save his order and come back to it later.
- The user can change his order before I pay for it.
- The user can see a running total of the cost of what he has chosen so far.

2.1.6 UserStory6: Done
Story Name: Personal Information

Description:
As a user of the app, I need to be able to create an account and have my personal information saved so that I do not need to re-enter all information.

Confirmation:
- The user can create an account.
- The user can save personal information, including cards etc.

2.1.7 UserStory7: Done
Story Name: Car Details

Description:
As a user, I want to be able to read and write details about the car so that I can make an informed choice about the car I will rent.

Confirmation:
- The user can open a detailed view.

- The user can read a description.
- The user can see an image of the car.
- The user can get personal information about the renter.

2.1.8 UserStory8: Done
Story Name: First Time User

Description:
As a first time user I want to be able to search for cars and navigate / look through the app without the need to register an account.

Confirmation:
- The user does not need to be logged in or register just to scroll through the app.

2.1.9 UserStory9: Done
Story Name: Frequent User

Description:
As a frequent user, I want to be able to stay logged in so that I don't need to log in every time that I use the app.

Confirmation:
- The user stays logged in when they exit the app/switch views.

2.1.10 UserStory10 : Done
Story Name: Upload image from desired location (phone)

Description:
As a user, I want to be able to upload images of my car from my phone's own photo library so that I can show pictures of my car to rent it out.

Confirmation:
- The user is able to upload an image of choice from their

phone library to the ad.

2.1.11 UserStory11 : Done for prototype .
Story Name: Show featured cars nearby

Description:
As a user, I want to easily see cars that are available in my local area so that I might rent a car close to where I am located.

Confirmation:
- The user is able to see featured cars listed near the Users location.

2.2 Definition of Done

The Following is the DoD used in the project:

1. GUI Implemented.
2. The code is compilable.
3. All tests are successful.
4. All changes have been committed and pushed to the master branch and the master branch is running without error.

2.3 User interface

In the beginning of the project a preliminary sketch of the user interface was created to help with the visualization of the program (See figure 1,2,3).

The application is built with three major views: home, browse and profile. The bottom menu makes it easy for the user to switch between the three different views at any time. Navigate to the home view at the bottom left, the browse and search view in the middle and finally the profile view at the bottom right.

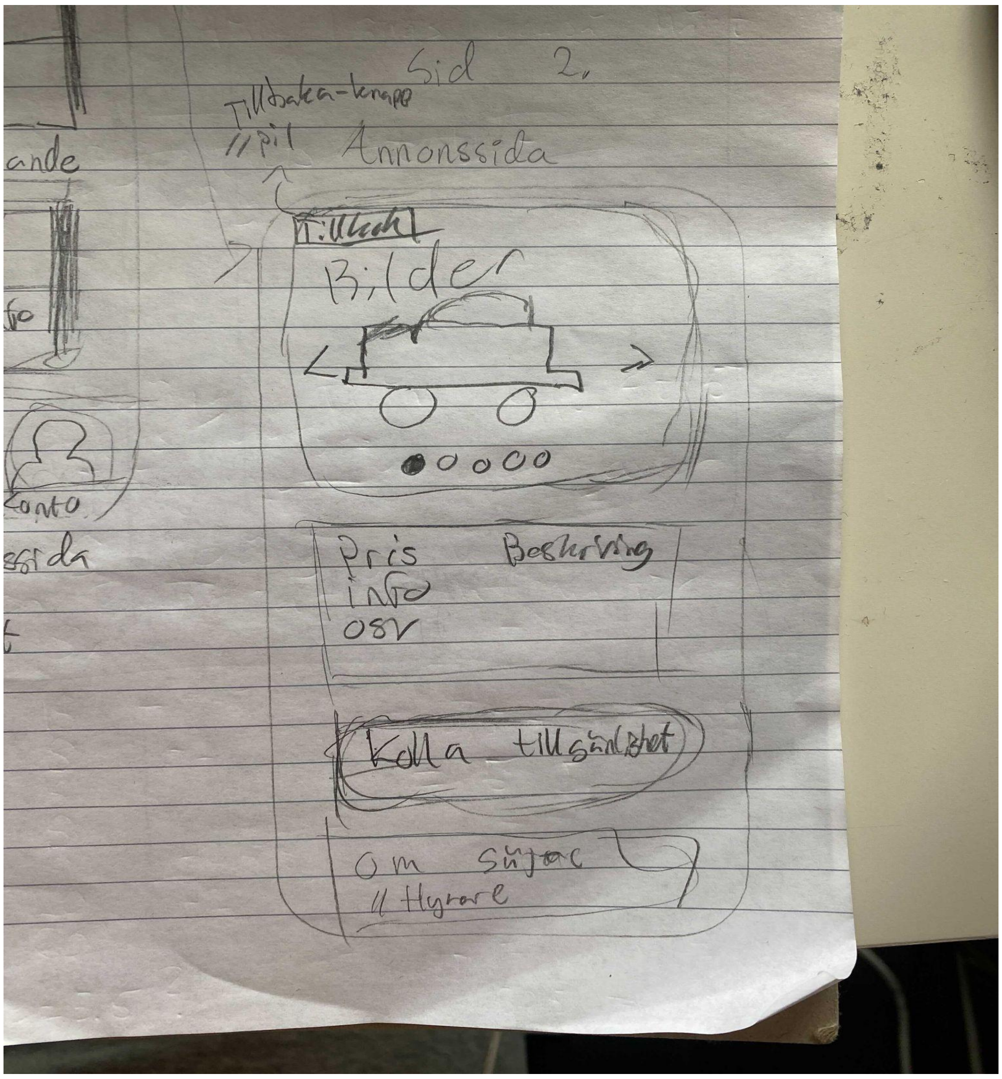


Figure 1: A Preliminary sketch of the interface

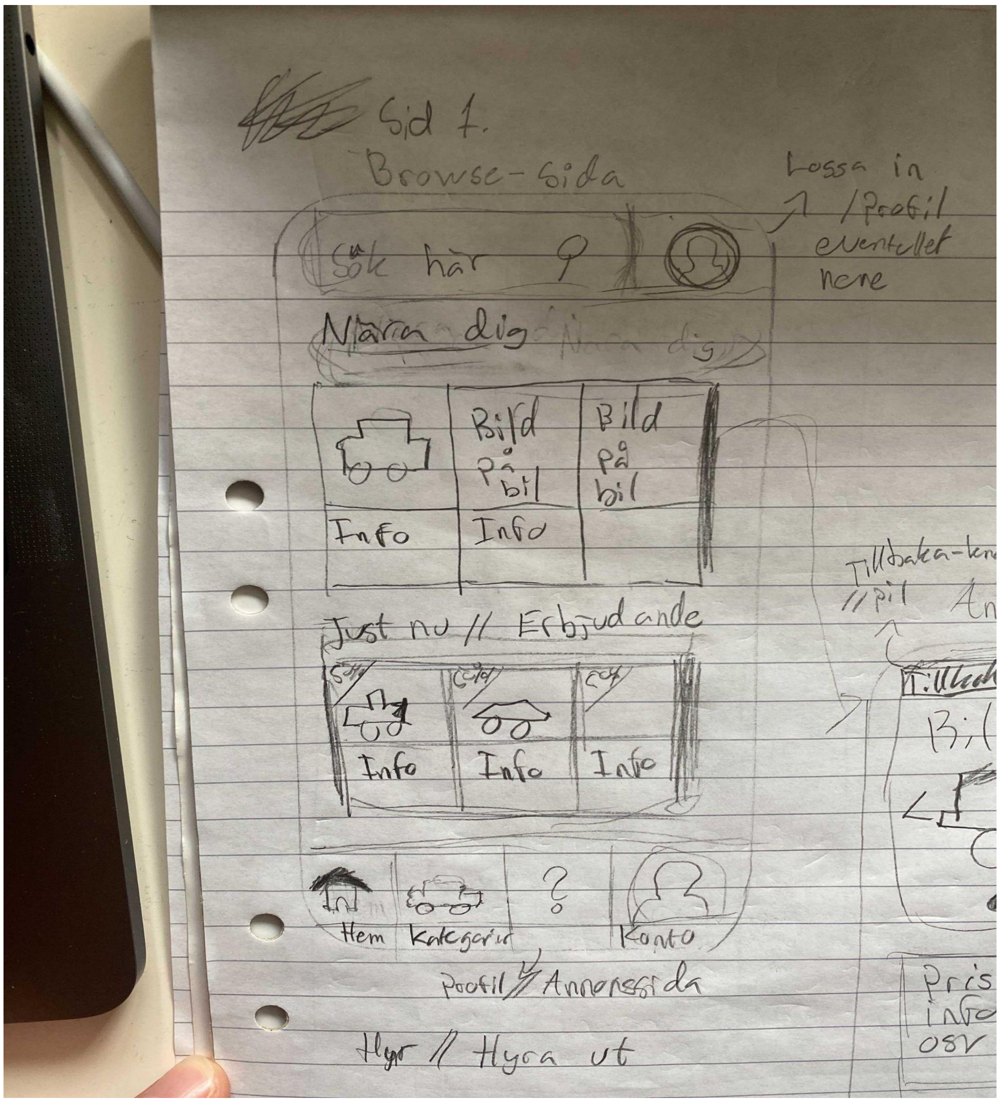


Figure 2: A Preliminary sketch of the interface

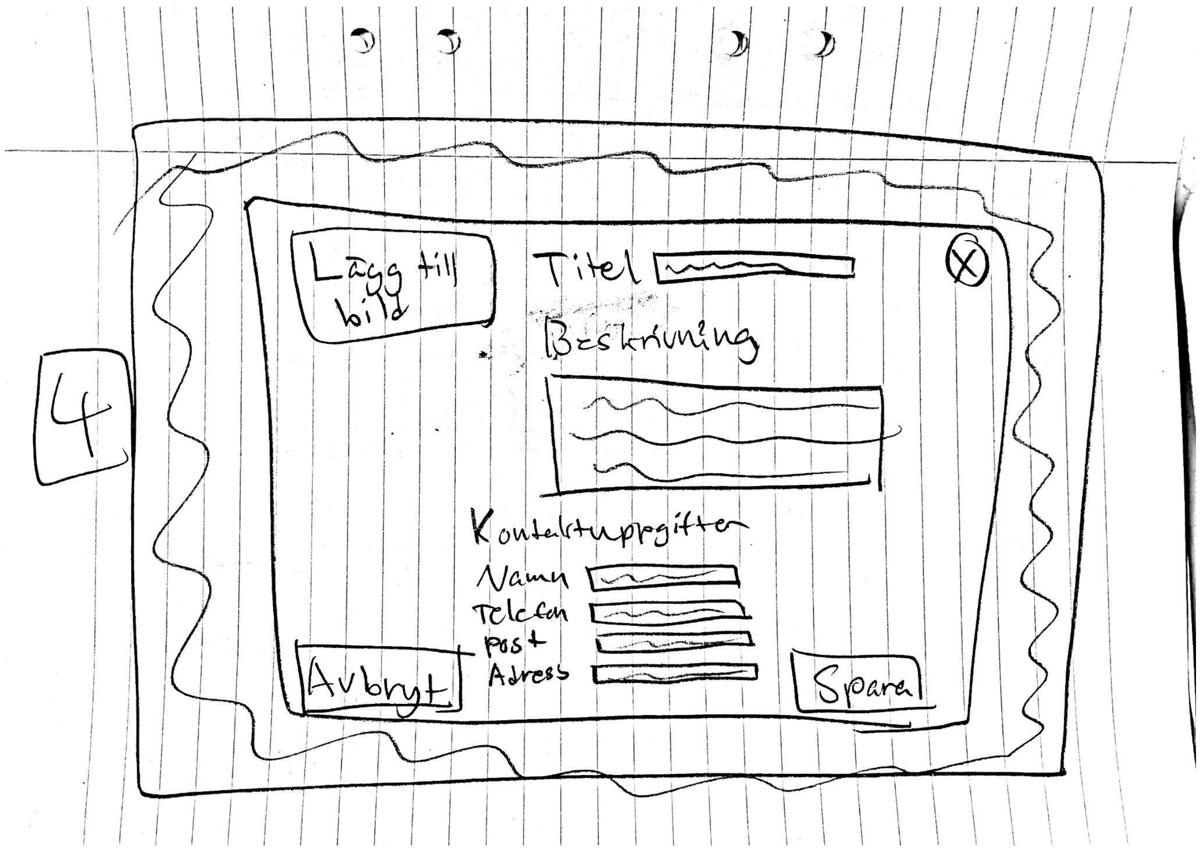


Figure 3: A Preliminary sketch of the interface

Later the interface was made more defined through the creation of a “High fidelity” (detailed) mock-up sketch, created in ”Figma” (See figure 4,5,6).

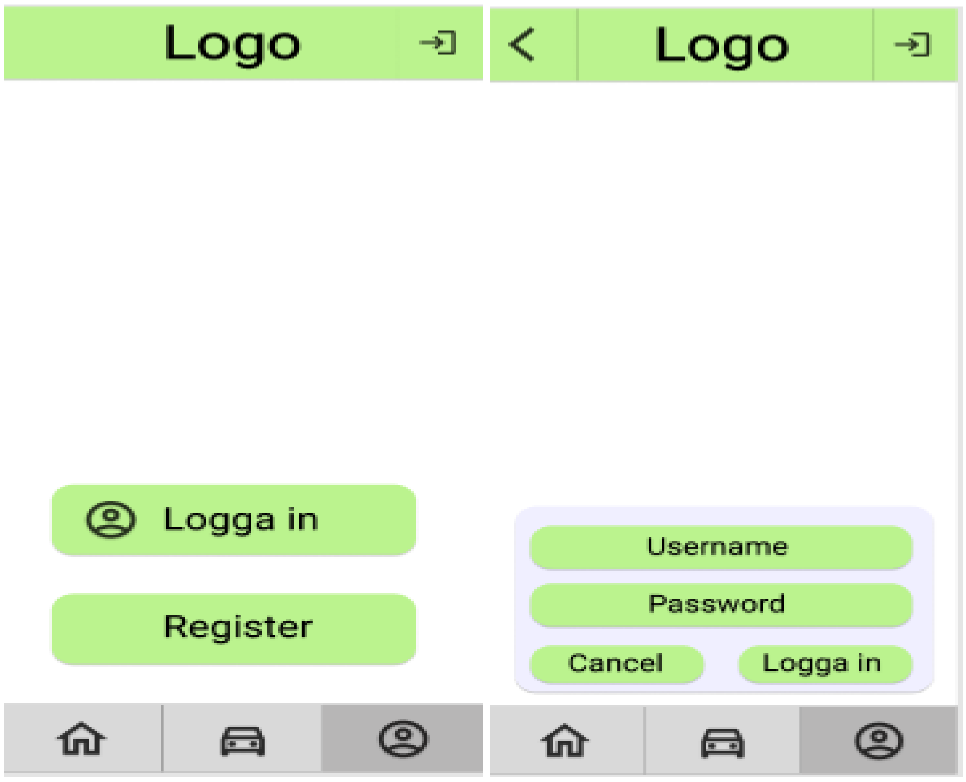
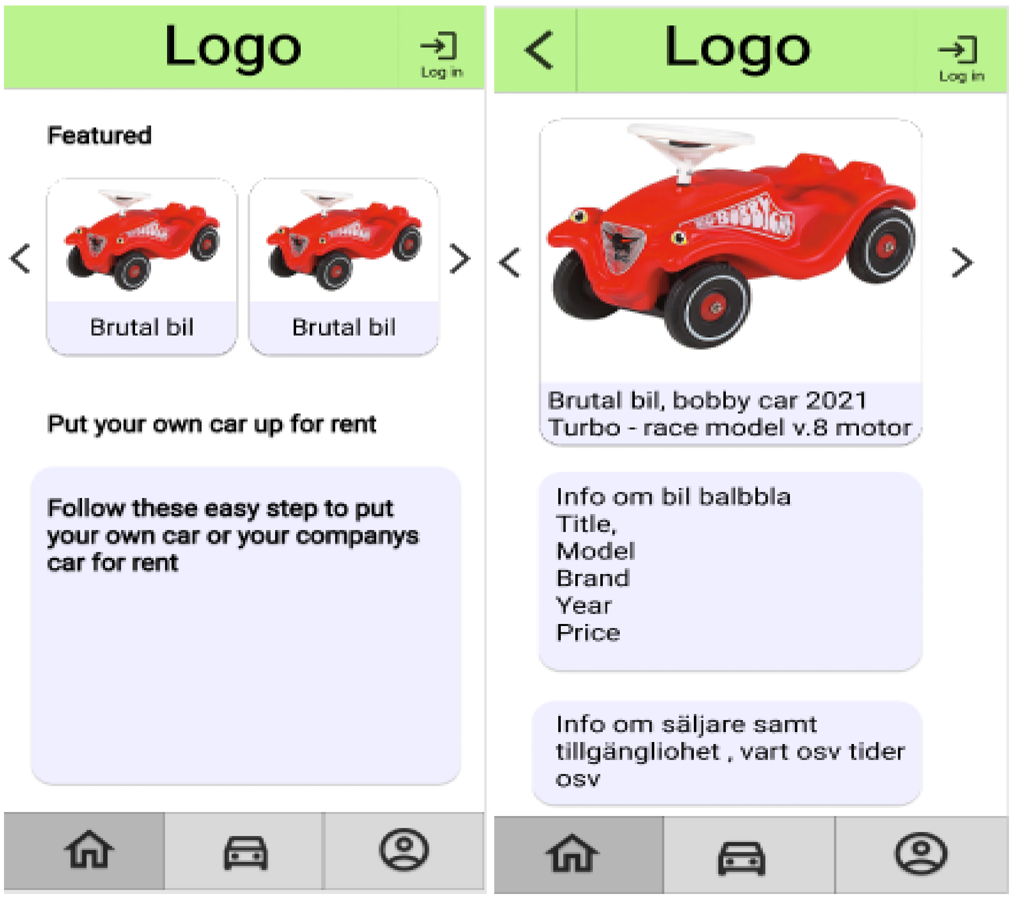


Figure 4: A Hi-Fi mockup of the interface



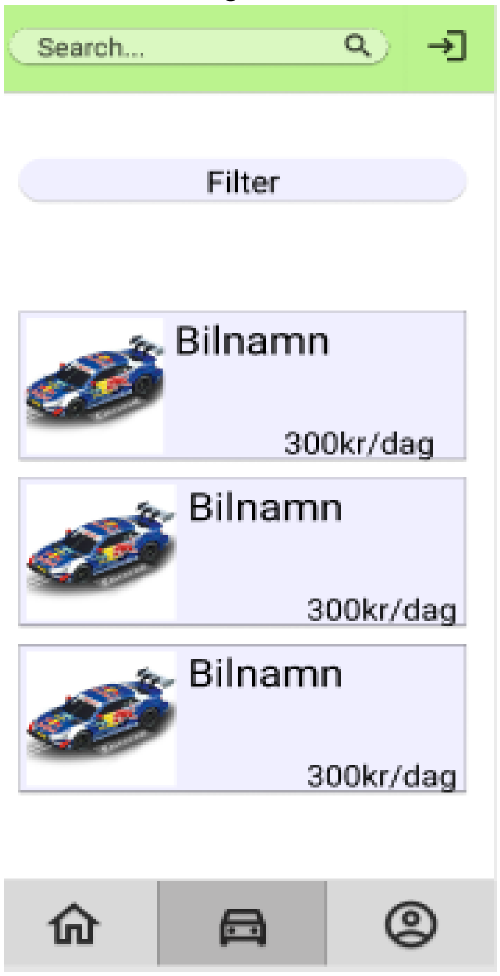
Follow these easy step to put your own car or your companys car for rent

Brutal bil, bobby car 2021 Turbo - race model v.8 motor

Info om bil balbbla
Title,
Model
Brand
Year
Price

Info om säljare samt tillgängliohet , vart osv tider osv

Figure 5: A Hi-Fi mockup of the interface



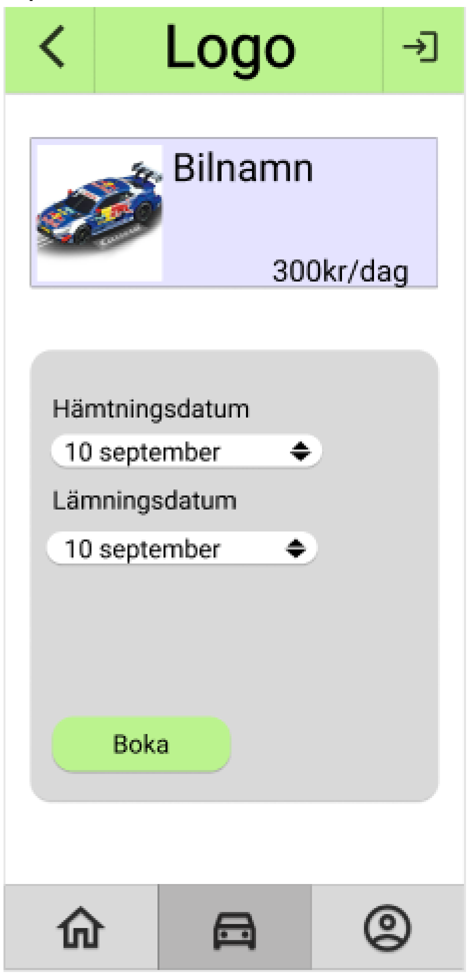


Figure 4: A Hi-Fi mockup of the interface

After getting a comprehensive overview of the application and the most important features that would be included, we started working on the design of the project in .xml files by using Android Studio to get the final product. (See figure 7,8).

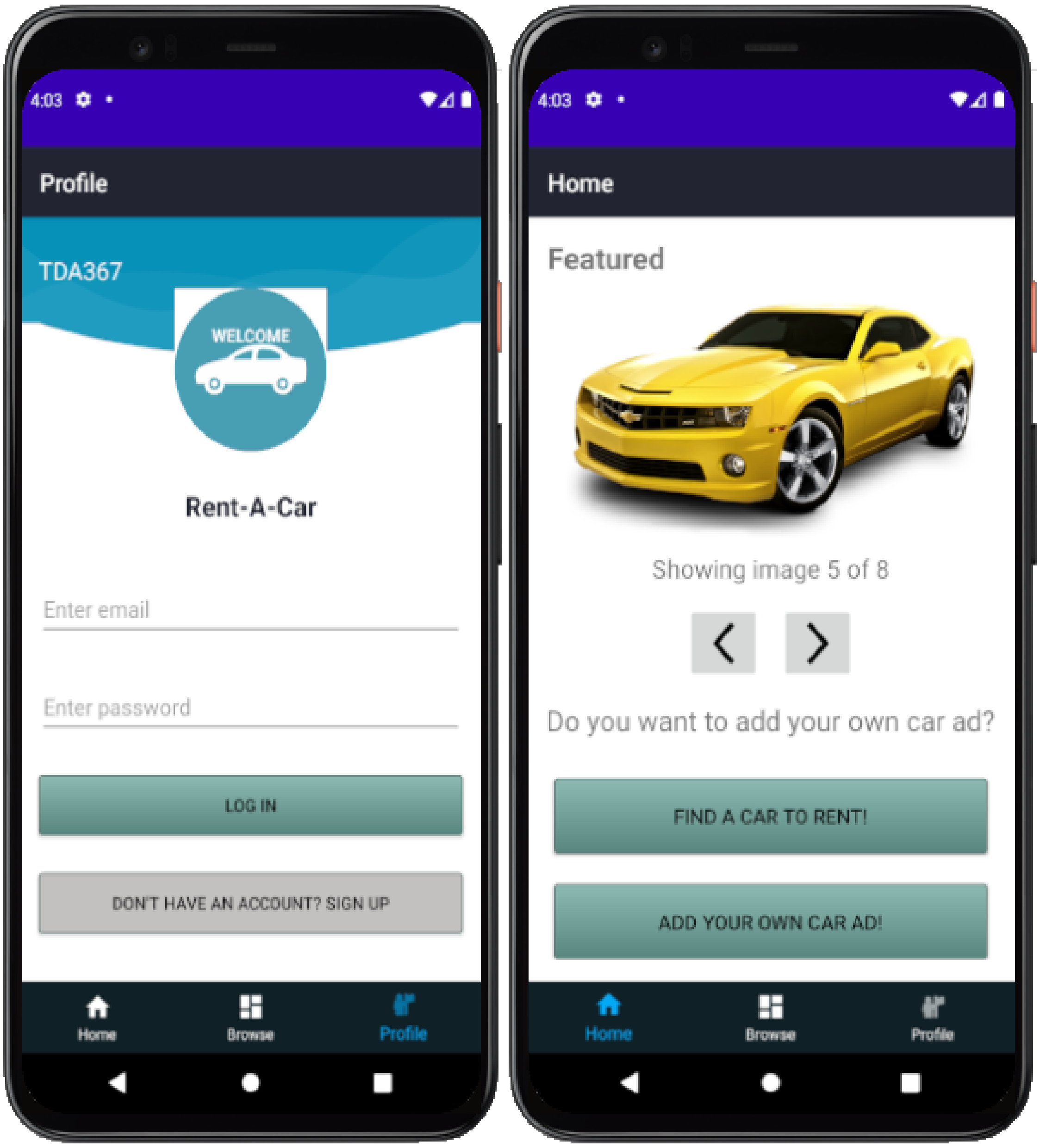
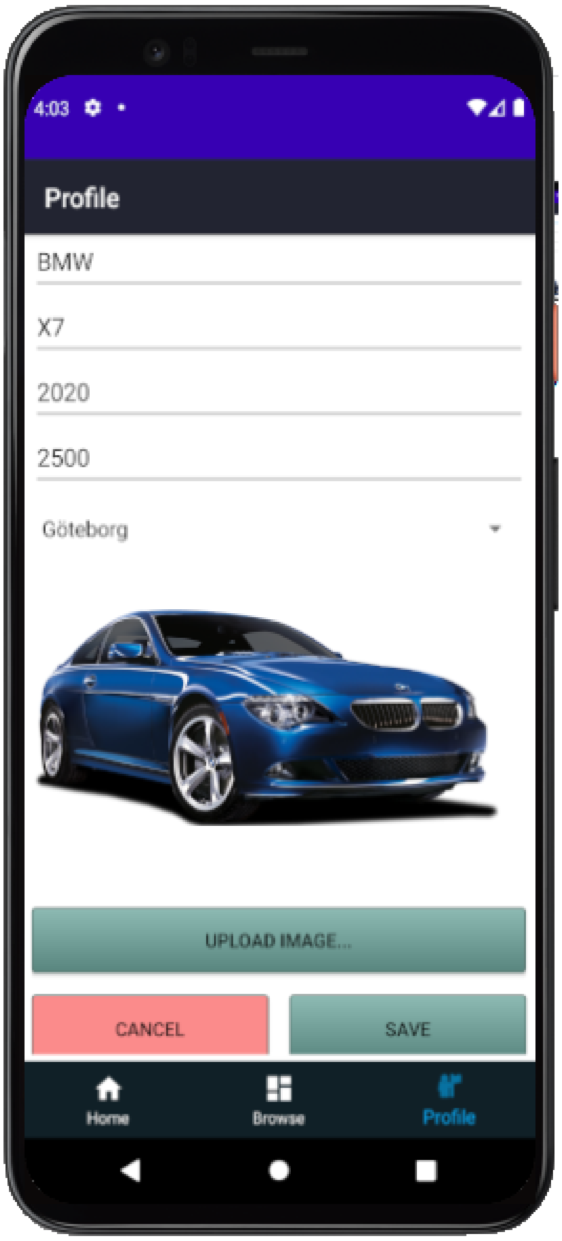


Figure 7: Profile and Home interfaces



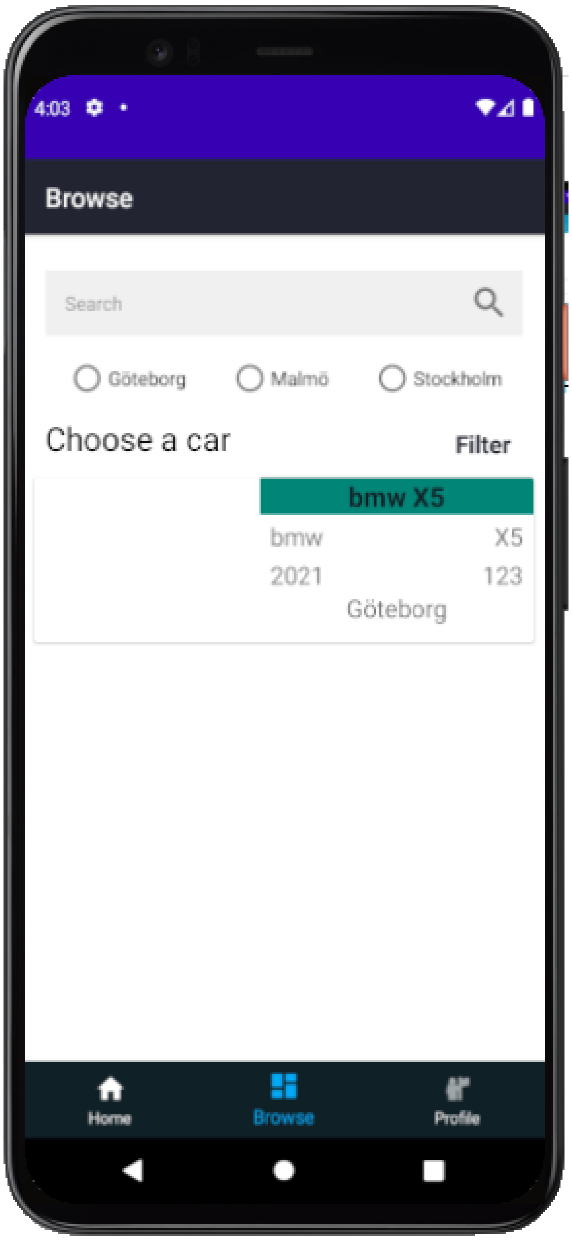


Figure 8: Add Car (Profile) and Browse interfaces

3. Domain model

The program is a type of car rental app that uses user input and input from various APIs to produce informative results which is then fed back to the user through a GUI. The user can also browse the cars without having to enter anything into the program, but he cannot rent a car without logging in or creating an account.

The user is the main information object of the program and the user is the owner of a number of properties. The properties in the app hold most of the information necessary in order to rent a car.

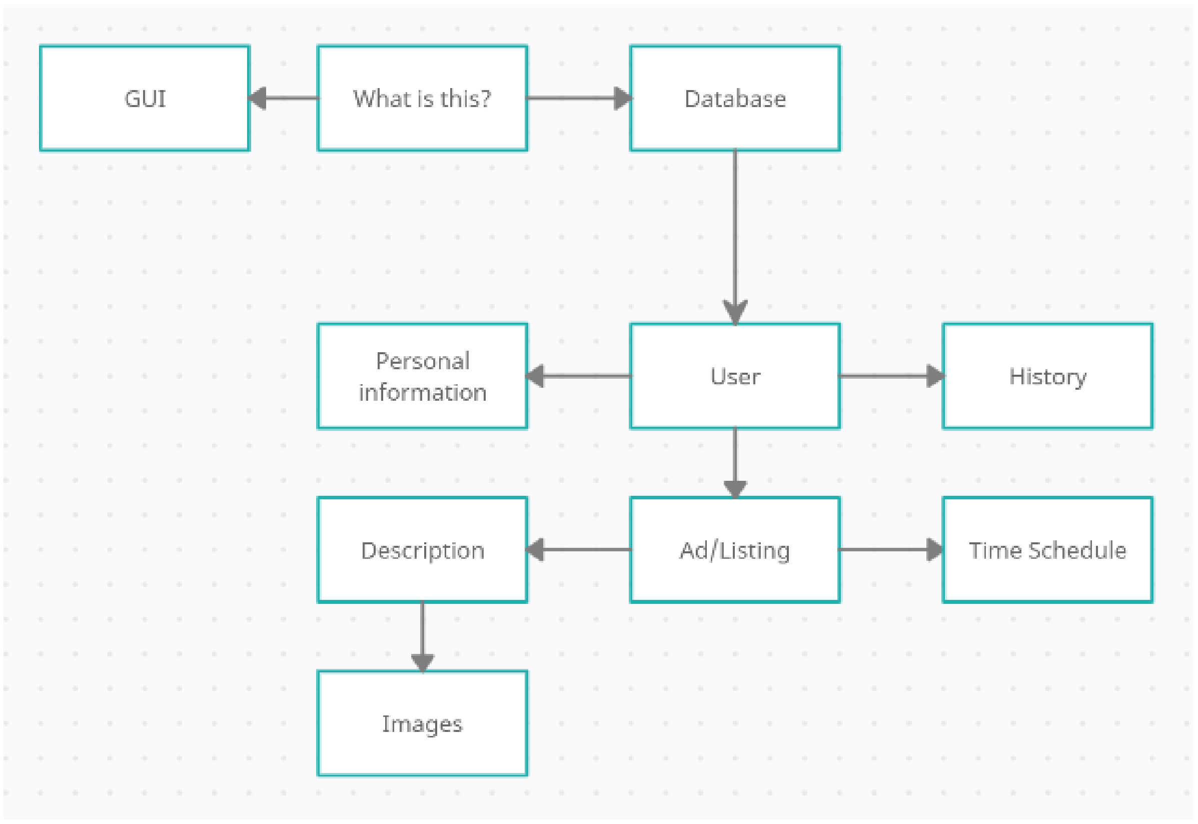


Figure: The Domain of the app

3.1 Class responsibilities

The program uses the MVC design pattern. The GUI part of our model makes up the View. It consists of our MainActivity and Fragments that are part of the Android API.

The Controller part is made up of ViewModels that are also a part of the Android API. This class is used to get information from the Model to our View.

The Model contains our data and logic for the program.

The User class is used to keep information about users such as name, email, card information, etc.

The Listing class is used to keep information about every individual listing in our program. It stores information such as what car is up for rent, price, what dates the car is available, etc.

References

[1] Figma, “Mockup tool,” 2019. [Online]. Tillgänglig: https://www.figma.com

[2] Firebase

[3] Android Studio

[4] Dependency analysis matrix java.
https://www.jetbrains.com/help/idea/dsm-analysis.html

[5] Android Studio emulator https://developer.android.com/studio/run/emulator

[6] Find bugs http://findbugs.sourceforge.net

[7] pmd https://pmd.github.io

[8]Junit https://junit.org/junit5/

Chalmers University of Technology

OOP Projekt

System design document (SDD)

Rent A Car - Stulb

Albert W, Hannes T, Jamal M, Johan S, Josef N

24/10-2021
v.2.0

---

1. Introduction

Car Rental Application is based on a concept to book a car and taxi online. Here at first the user has to login or sign up to get access. Then the user can list, search for cars according to their needs, check each car’s description, book prices and book easily with help of various payment methods. All the available cars can be rated by the users too. The user can filter, search or sort cars by price for searching cars.

The application also displays a reserved car’s list. With the help of this application, online car booking has become easier for the customers. It can be used in several android gadgets such as smartphones, tablets, television. This project is easy to operate and understood by the user.

1.1 Design goals

The goal is to have a loosely coupled and modular application that is easy to extend.It should be simple to add new accounts. Android is an open source so that developers find it easy to establish and expand new features.

1.2 Definitions, acronyms, and abbreviations

GUI ”Graphical User Interface” also ”User Interface”, the part of the program that the user sees and interacts with.

User Story short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

API ”Application Programming Interface”, an interface or communication proto-col between a client and a server intended to simplify the building of client-side software.

Hi-Fi ”High fidelity”, a sketch with much detail.

UML ”Unified Modeling Language”, A visual language for mapping and repre-senting systems. Can be used for modelling och class diagrams, sequence diagrams and more. In this project used for Domain and design model.

MVC “Model, View and Controller”

NoSQL NoSQL originally referring to "non-SQL" or "non-relational” [1]. It is a database which provides a mechanism for data storage and retrieval of data. That is not modeled with tabular relations used in relational databases.

JSON (JavaScript Object Notation). Is a data-interchange format used because it is both easy for humans to read and write and furthermore machines can easily parse and generate it.

Firebase Realtime Database The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real time.

2. System architecture

The overall structure of the application is following the MVC-pattern. It requires only one android phone or android emulator on a computer to function. The MVC-pattern is a software design pattern commonly used when designing GUIs. The pattern is divided into three different parts, model, view and controller. The first part is the model that is responsible for the datan and the logic and it should not know anything about the part who is not a part of the model . The second part is the view, the view is responsible for the graphics of the software. Finally is the controller which is responsible for user interactions, processing events that can result in changes in both the view and the model. The software application starts by building the gradle files and then running the executable file and stops by closing the emulator.

2.1 Model

The Model Package holds classes representing data as well as the core logic of the program. The model handles data about the cars stored in the database. It also handles dates for the reservations and the user's payment details.

2.2 View

The view package holds classes that are used for the visual representation of elements from the model package and takes use of the library **.xml** in order to accomplish this.
The view is together with the controller responsible for rendering the user interface. Since we are using xml files.

2.3 Controller

The controller package holds classes and interfaces that are used to manage views in the view package and to handle user input such as key presses and then it passes the action on to the model who then manages the action.
The controller package is divided into three different packages, home, browse and profile.

2.4 Singleton-pattern

Singleton-pattern is used in the Database class. It ensures that there is only one one instance of the class and it also provides a global access point to the only instance.

2.4 MVC-pattern

Model View Controller-pattern the system architecture follows a MVC-pattern. The application is divided into three different packages, the model-package, the view-package and the controller-package.

2.5 resources

The resource package holds resources or data that can be accessed by the code. Images, icons, fonts and **XML** files can all be found in the resources package.

2.6 Communication at a high level

When the program is started an asynchronous Firebase query is sent to retrieve the data from the Firebase database. When the query is finished the program will load the database into its memory. As this is happening the program is launching the views and view controllers to display the user interface. When a view requests information from the database the model is called from a controller and the Database class that exists in the application memory hands the controller the information which in turn hands the information over to the view. When the application is closed the Database class sends the information to the Firebase database and the updated information is saved.

3. System design

3.1 Model View Controller

Since the program has a graphical user interface representation, a **MVC** pattern was used to separate the different components of the program. The program is based on **XML** graphical components.

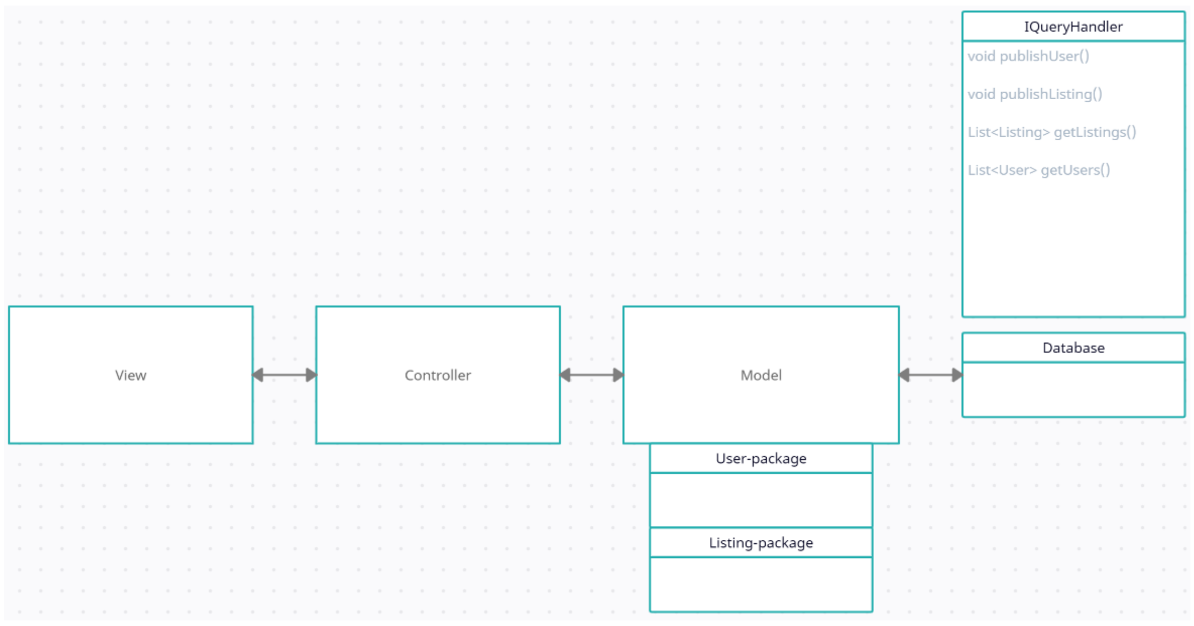


Figure 1: Blocks representing the design model of the program, a clear MVC pattern

The structure follows **MVC** which first and foremost requires that the model package is not dependent on any other part of the program. This makes the code modular and creates the possibility of using different views and controllers for the same model.

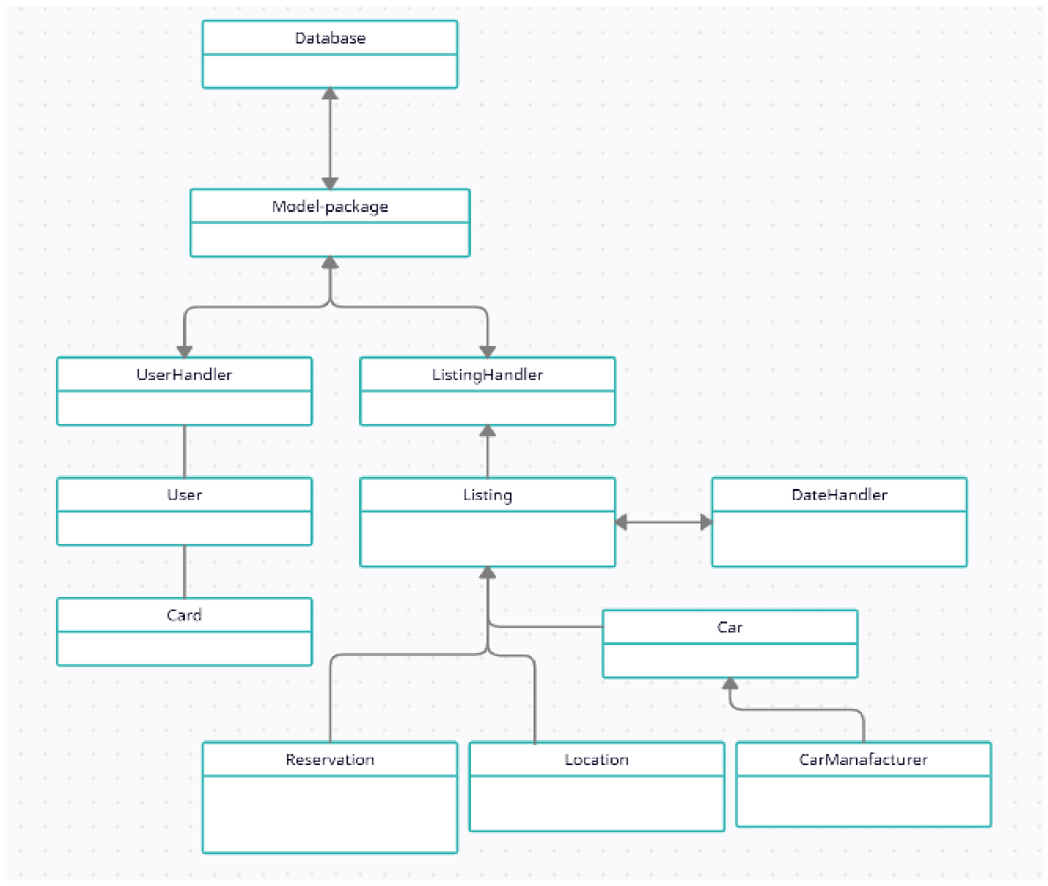


Figure 2: The model block

3.2 Relation between domain model and design model

The domain model and the design model were closely related, however during our process we have changed our design model as we received feedback from our teacher and another group. Our domain model shows the application from a module perspective while the design model shows the application from a class perspective and how the classes are related to each other. Our design model was redesigned to better suit our application needs and for this reason the original domain model is not very close to the implemented design model.

4. Persistent data management

To store data we use the Google product Firebase Realtime Database which is a NoSQL cloud database used to store and sync data. From the Firebase database the data can be synced at any time and across different clients such as, the web, android and IOS. The data is stored in JSON format which updates in real time with the connected client.

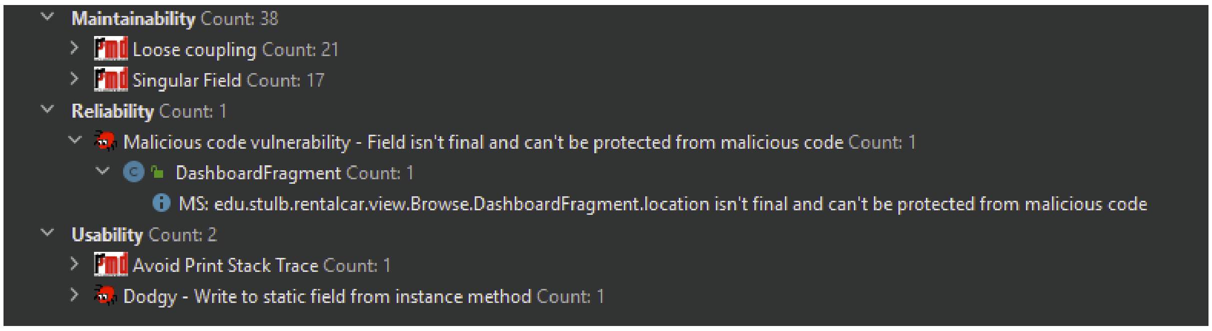
There are a lot of advantages of using Firebase Realtime Database with the main advantage being that the data is updated in real-time. Therefore there is no need to make requests for the data changes or updates. With the database use of data synchronization when data changes as every time there is a change it will reflect the connected user instantly. Another advantage of Firebase is that the apps you create will remain responsive even if the connection to the database is lost. When the connection is established again the user will receive the data changes from the database. Finally the data in the database is easily accessible through the web and it's possible to manage your database from both PC and mobile devices.

As we developed our program we noticed that a realtime database came with its difficulties. Every query or request to the database was handled using asynchronous functions which we decided we were not capable of managing in the time frame of this project. For this reason we used a temporary solution to only sync with the Firebase Database on program startup and when it is exited. Note that program exited means when you leave the Android home screen.

5. Quality

In order to test the program the testing framework **JUnit** [2] was used. This made it easy to test functionality without a graphical interface initially. The tests are located in a separate folder test.

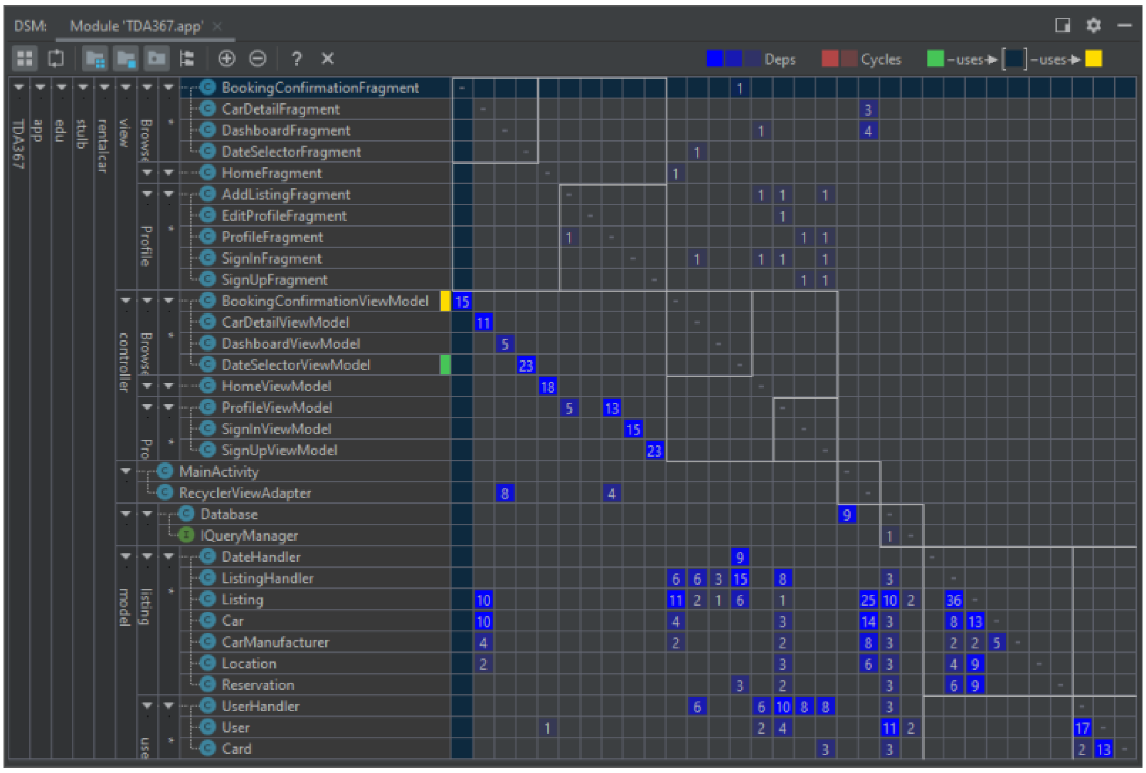
Analysis result from PMD and Find bugs:



```
Maintainability Count: 38
    Loose coupling Count: 21
    Singular Field Count: 17
Reliability Count: 1
    Malicious code vulnerability - Field isn't final and can't be protected from malicious code Count: 1
        DashboardFragment Count: 1
            MS: edu.stulb.rentalcar.view.Browse.DashboardFragment.location isn't final and can't be protected from malicious code
Usability Count: 2
    Avoid Print Stack Trace Count: 1
    Dodgy - Write to static field from instance method Count: 1
```

Figure 1: PMD and Find bugs result [3][4].

Analysis result from Intellij code analyzer dependency matrix :


DSM: Module 'TDA367.app'

Deps Cycles –uses→ –uses→

TDA367 app edu stub rentalcar view Browse Profile controller Browse Pro model listing use

BookingConfirmationFragment	-													1																		
CarDetailFragment		-																		3												
DashboardFragment			-												1					4												
DateSelectorFragment				-								1																				
HomeFragment					-						1																					
AddListingFragment						-									1	1		1														
EditProfileFragment							-									1																
ProfileFragment						1		-									1	1														
SignInFragment									-			1			1	1		1														
SignUpFragment										-							1	1														
BookingConfirmationViewModel	15										-																					
CarDetailViewModel		11										-																				
DashboardViewModel			5										-																			
DateSelectorViewModel				23										-																		
HomeViewModel					18										-																	
ProfileViewModel						5		13								-																
SignInViewModel									15								-															
SignUpViewModel										23								-														
MainActivity																			-													
RecyclerViewAdapter			8					4												-												
Database																			9		-											
IQueryManager																					1	-										
DateHandler														9									-									
ListingHandler											6	6	3	15		8					3			-								
Listing		10									11	2	1	6		1				25	10	2		36	-							
Car		10									4					3				14	3			8	13	-						
CarManufacturer		4									2					2				8	3			2	2	5	-					
Location		2														3				6	3			4	9			-				
Reservation														3		2					3			6	9				-			
UserHandler												6			6	10	8	8			3									-		
User					1										2	4					11	2								17	-	
Card																		3			3									2	13	-

Figure 2: Intellij code analyzer dependency matrix result [5].

5.1 Access control and security

Rent A Car is an application which can be used by anyone. All registered and signed in users can use all the functionality of the app. The only other “role” apart from a signed in user is a user that is not signed in. When the user is not signed in they can not use all of the apps functionality they won't be able to list their own ad or book a car. However the non-signed user can use the search function and scroll through the different views like “home” and “dashboard”.

References

[1] Google product Firebase Realtime Database- https://firebase.google.com/
[2] JUnit. [Online]. Available: https://junit.org/junit5/
[3] Find bugs http://findbugs.sourceforge.net
[4] pmd https://pmd.github.io
[5] Dependency analysis matrix java.
https://www.jetbrains.com/help/idea/dsm-analysis.html

Do the design and implementation follow design principles?

+ The code follows MVC properly which is an application of Separation of Concern.

+ Open-closed principle are used in classes which are open for extension and closed

for modification using private variables and fields.

+ The Single Responsibility Principle is used in many of the smaller classes which

contains only one method like ProductJsonFileParser.

+ The Interface Segregation Principle is used commonly. For example in the package

favorites where they have public interface IProductListIO which is implemented by

classes as ProductListFileIO. They also use interfaces for IProductParser.

- At some places like the controller they break against The Law of Demeter which has

the rule "only talk to friends". Used to reduce coupling between classes or objects.

-

Does the project use a consistent coding style?

+ Yes the project follows a consistent coding convention.

+ Yes, code chunks are well-organised. The comments are found above the code.

+ CamelCases har commonly used throughout the project.

+ Clear dividing between packages with interfaces implemented by classes.

Is the code reusable?

+ Yes, because it is well structured in the way that it is well separated.

+ The use of interfaces makes the code easy to separate.

Is it easy to maintain?

+ Yes, as the code is easy to read and structured.

+ Because all components are separated from each other, they are easy to change.

+ Also the dependencies of the modules are well thought out.

Can we easily add/remove functionality?

+ Yes, as the code is easy to read and structured.

+ Also the dependencies of the modules are well thought out.

+ Classes and functions are generally small.

+ The extended usage and implementation of factory pattern makes it easier to add

new functionality, as new factory methods can be added to create new objects,

without having to change the code in too many places.

Are design patterns used?

+ The model module uses the facade pattern to mask and simplify the backend.

+ As it is listed in the provided SDD, the application implements a handful of design

patterns such as facade pattern which is used to put a facade over the rest of the

code.

Is the code documented?

+ Some code is well-documented using javadocs and some code has no

documentation. If the rest of the code i as well-documented as the parts that has

been commented,

Are proper names used?

+ Yes, the naming is mostly fine and the names files give an understanding of what the files contain, with a proper use of camelcase.

Is the design modular? Are there any unnecessary dependencies?

+ The code is highly separated into independent modules. Almost every package has a

public interface to hide the internal details of the package. Modularity also means that

the code is easy to maintain, refactor and test.

+ The program is modular because it follows an MVC architecture without circular or

other unnecessary dependencies.

Does the code use proper abstractions?

+ The command package is well written, as it is easy to remove/add new commands

and incorporate them into the design without having to rewrite a lot of code.

+ Good use of interfaces to make the code more abstract.

Is the code well tested?

+ Yes. There are plenty of tests (especially for model packages) that cover the most

essential parts of the code.

Are there any security problems, are there any performance issues?

+ Favorites are working

- Images are loading slow, gets image from Systembolaget on every part of the

navigation

- Tests weren't working properly

- The FileReader and FileWriter constructors instantiate, thats causing issues with

garbage collection when the finalized methods are called.

- Instead of new FileWriter(fileName) use

Files.newBufferedWriter(Paths.get(fileName))

Is the code easy to understand? Does it have an MVC structure, and is the model isolated

from the other parts?

+ Yes, the code is distributed in different folders containing the different parts that make

up the structure.

+ It seems that the code makes use of a MVC pattern.

Can the design or code be improved? Are there better solutions?

+ The code is well commented and follows a common style.

+ Naming of packages and classes is at times clear.

- The view is fixed in size. If the user had a different sized window or changed window

size, the view would remain the same.

- It's possible to implement an Observer-pattern into the code to notify multiple objects

about events that happen to the object that they are observing.