Peer-review of “Rent a Car”

We were not able to build the project using gradle and there were no supplied instructions on how to build it. As such, this peer review will mainly be based on analyzing the codebase.

Comments / Javadoc

Not a single line of javadoc was found and that made it very hard to understand what the purpose of the different methods are. Since we have never seen the android api before we had no idea what anything did. So a major piece of feedback would be to add javadoc to all public methods in the code. There are some normal comments but there could be more. It would make it a lot easier for someone else to understand the code. Most variable/method/class names are verbose and class names follow a naming structure which is good.

MVC / Structure

Based on the SDD, it seems like this project aims to implement the MVC pattern. However, we can only find a package called ui. The rest of the classes do not seem to belong to any MVC package. Consequently, the overall structure of the codebase is hard to understand, especially since all the classes lack proper documentation.

Further, due to the lack of an MVC structure, the model does not seem to be as isolated as it could be. In fact, we have had a hard time identifying which classes actually belong to the model. The CarAdModel is a pure data class and is the only example of a class that definitely belongs in the model, but it should have some kind of behaviour code as well. Most other classes seem to be a mix of model, view and controller so each class doesn’t follow the single responsibility principle very well. Since many classes have several responsibilities, it seems like it could be difficult to add new functionality without modifying existing classes, which breaks the open closed principle. Because of high coupling it does seem like most of the code is not

reusable. It seems like it would be difficult to add a different view using the same model.

In SignUpPaymentFragment and AddCarAdFragment, there is quite a bit of casting (e.g. to EditText and Button). This is not very type safe. However, we have never worked with the android api before, so maybe this is necessary?

RAD/SDD

RAD:

- Definitions do not contain any definitions relevant to the domain. Why is “RAD” defined in the RAD document?
- The user stories are generally fine, although there seems to be some overlap. For instance, user stories two, three and five are pretty similar. Further, it is not clear what priority the user stories have.
- User interface looks nice. Good to get some pictures of the Figma mock up.
- Domain model looks good but is not reflected in the code. Should probably be updated. GUI can be removed from the domain model.

SDD:

- Definitions are more relevant in the SDD than the RAD.
    - Further, consider adding definitions that relate to the domain or the android api, e.g. fragment and toast.
- System architecture is not reflected in the code.
- Persistent data management section is useful for understanding some of the code base.

Tests

There are no tests. Since it is expected to cover 100% of the model writing tests should probably be a thing to focus on.

Improvements

- Higher separation between model, view and controller.
- Everything that concerns the database and the applications interaction with it we would move into the model package, these methods could then be called from the controller package.
- Add Javadoc to all public methods.
- Create tests.