Peer review

Eric Erlandsson Hollgren, Sally Chung, Amanda Dyrell Papacosta, Jonatan Jageklint, Zoé Opendries

October 2022

1 RAD

The structure of RAD is following the given template. The introduction of the RAD is consistent and presents the project in a good way. It explains abbreviations and defines the mentioned concept in an adequate way.

The user stories are not fully following the given template. Each user story, that is mentioned in the document, is given a name, priority, description, confirmation criteria and an id. However, the confirmation criteria is left empty in comparison to the other sections. Hence, making it difficult to know what every user story's confirmation criteria as accepted was. Are they all connected to the definition of done or some sort?

Another thing to consider is how the developers/writers has defined their priority list. Several of them as of now are considered high and some of the high ones are perhaps too large of an user story. For instance, the "Up-to-Date" is not really of an user story, it feels, when reading it, more like a on-going task or should even be considered as an acceptance criteria? Perhaps, that it even can become a part of "Definition of Done". The id 7 is not necessary to be a user story, as this is probably something that is on-going for the project.

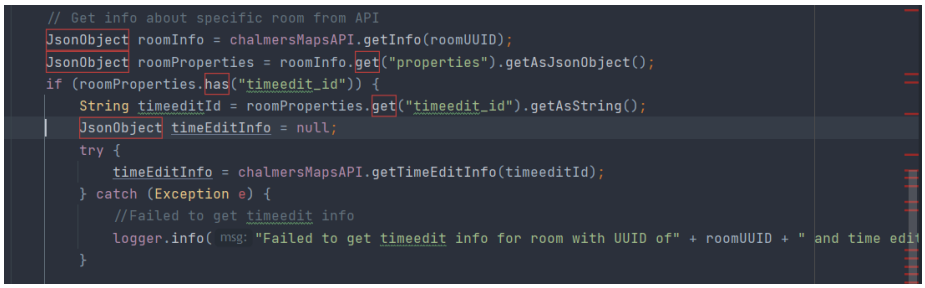
A good "size" of a user story is the "Usable on phones"- story that mentions that the user want to be able to use the app on smartphone. Considering that they are designing an application that should be available on smartphone, it is reasonable that it is also highly prioritized. Furthermore, some of them could be paired up and be reconstructed to an EPIC and then scaled down to a user story. For instance, 6, 8 and 10 could have a bigger story as an EPIC (since they all are a part of early stages of the project. The user story number 8 could also be removed,in regards of designing the UML, since this is also something on-going and decision-making will happen consistently.

2 Code

In RoomProvider the function

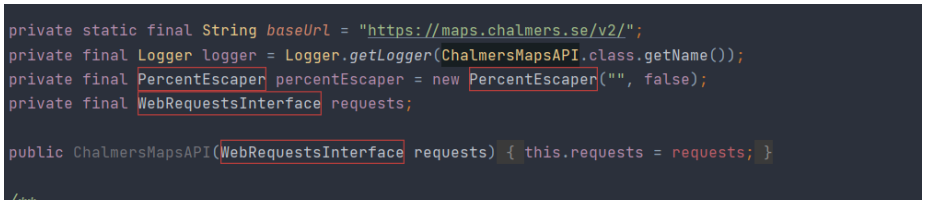
`List<Room> getNewDataToCache()`

can be split up to multiple functions since it is a rather large and complex function as is. Using functional decomposition it can be split up into at least two functions where the code in the picture below could be put in its own function called something like getRoomInfo(UUID roomUUID). This will make the code easier to read and more adhere to the definition of done where it says "try to keep methods as small as possible".



```
// Get info about specific room from API
JsonObject roomInfo = chalmersMapsAPI.getInfo(roomUUID);
JsonObject roomProperties = roomInfo.get("properties").getAsJsonObject();
if (roomProperties.has("timeedit_id")) {
    String timeeditId = roomProperties.get("timeedit_id").getAsString();
    JsonObject timeEditInfo = null;
    try {
        timeEditInfo = chalmersMapsAPI.getTimeEditInfo(timeeditId);
    } catch (Exception e) {
        //Failed to get timeedit info
        logger.info( msg: "Failed to get timeedit info for room with UUID of" + roomUUID + " and time edi
    }
```

For some classes constructors are missing and in the class ChalmersMapsApi a constructor is used but the private variables are not instantiated within the constructor. See image below



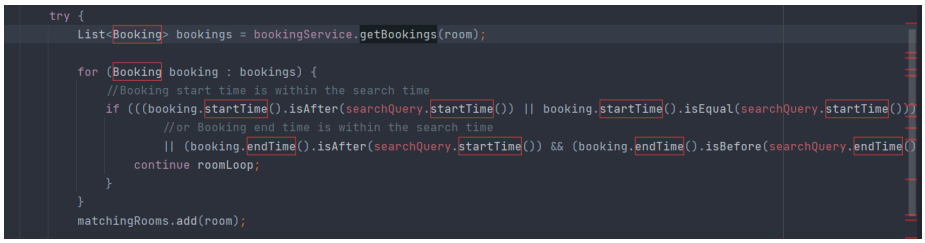
```
private static final String baseUrl = "https://maps.chalmers.se/v2/";
private final Logger logger = Logger.getLogger(ChalmersMapsAPI.class.getName());
private final PercentEscaper percentEscaper = new PercentEscaper("", false);
private final WebRequestsInterface requests;

public ChalmersMapsAPI(WebRequestsInterface requests) { this.requests = requests; }
```

The same thing can be found in other classes such as RoomProvider, RoomService, RouteService, Search service, TimeEditApi, and TimeEditBookingProvider.

In a comment in GetARoomFacade it says the class should be the only one accesiable from outside the model package, this is true but all the other classes could have their accesiability changed to protected in order to further insure that GetARoomFacade is the only accessible object from the client code.

SearchService can use some functional decompositioning. The for loop in the code below can be extracted into a new function making the code easier on the eyes.



The UML diagram does not reflect the actually code. It has a lot of missing public classes that exist in the code but not in the UML. Apart from that in overall the UML looks good.

In the class ChalmersMapsApi the methods use very similar code. The class could use a single call method taking in abstract request objects containing the base url, a custom url part and parameters for the call. When client code wants to call the api the client simply creates a new concrete implementation of the abstract request object. The function calling the api then extracts the data from the request object and calls the api. This is known as the command pattern. For further explanation see example code below.

```
    interface IRequest{
string getBaseURL();
List<string> getParams();
string extraURL();
}

class informationBoardRequest implements IRequest{
private final string baseURL;
private final List<string> params;
private final string extraURL;

public informationBoardRequest(string baseURL, List<string> params, string extraURL){
this.baseURL = baseURL;
this.params = params;
this.extraURL = extraURL;
}
@Override
string getBaseURL{...}
@Override
List<string> getParams{...}
@Override
string extraURL{...}
```

```
}

class timeEditScheduleRequest implements IRequest{
private final string baseURL;
private final List<string> params;
private final string extraURL;

public timeEditScheduleBoardRequest(string baseURL, List<string> params, string extraURL){
this.baseURL = baseURL;
this.params = params;
this.extraURL = extraURL;
}
@Override
string getBaseURL{...}
@Override
List<string> getParams{...}
@Override
string extraURL{...}
}

Client Code:
IRequest test = new timeEditScheduleRequest(...)
chalmersapi.call(test);
```