Project Report **06 December 2020**

TECHNICAL UNIVERSITY OF DENMARK

02369 Software and Design Patterns

Android Sport Event Project

SportsEvent

PROJECT PROVIDER: This project will be completed in the interest of Morten Salomonsen.

SUPERVISED BY: Mads Nyborg

Groupe B6

Mohamad Abdulfatah Ashmar _ S176492

Ismail Kilani _ S195460

Bashar Khaled Bdewi _ S183356

Andrey Baskakov _ S147199

Table of Contents

Section 1: Introduction

We are working on the Sports Event app; this project was given to us by our PO Morten Salomonsen of the GEOTRAIL Event org. This app has the purpose of filling the space between a very simple method of keeping track of races, and one that is complex such as for example Strava, which has a lot of utilities. Our app as an end product is supposed to be able to give an arranger a tool to start events (races), and its users the ability to join the events, complete them and share their event information (Facebook etc.) if they choose to do so.

Arranger: has a clear cut way of starting a new event with a specific time and date, has the ability to delete the event before it has started, the arranger can also view a list of the participants in his events and has a choice of removing participants of their choosing.

Users: Can view all upcoming events, browse through the details of any specific event such as the information written by the arranger or view other users that have already decided to join the event, these things can help the user decide whether joining is of any merit. After joining an event, you will have a period of hours or days (depends on the arrangement) to complete the event, after its begun of course, so when an event has

started, the user will have time to complete it when it fits them best. As a user you will receive relevant information on your app during your race (time, speed, map tracking).

We have so far during the app creation process mostly valued user experience, we believe that an app has to have decent layouts and transitions for its users to want to use it, however visual smoothness is only well received with a well-functioning app, which we plan to deliver.

Section 2: Process model

We have definitely used guidelines from both SCRUM and XP. Early in the semester we have had a lot of planning to do, where we have mainly abided by the SCRUM model, to develop user stories and assign roles in our group. We also initiated a contract and made specifications about how our work would be structured, and when and where we would hold our meetings, which early on was at least twice a week, on Mondays and Thursdays as it fit with the teachings on those days and seemed most logical. In our meetings we would review our current project state and then assign each person to a task, for example, 2-3 people would work on UI and the rest would look at other missing

areas in our expected deliveries that were due. While the iterations 1, 2 and 3 are somewhat alike to sprints it fits well with the SCRUM guidelines, however as the weeks moved along, we have come to adapt a more lenient and flexible work process which fits more with the XP guidelines.

Our work and meetings have shifted to be less rigid, and we now plan them to be about once or twice a week, our next meetings are planned during the previous one, this to us makes sense because we are a small team and that is why it is perhaps fair to have more leniency instead of strict order. We develop our app by shorter iterations, about once a week we merge our branches and test out the app, then we sit together and work on it until it works as expected.

One of the bigger XP principles that we have perhaps not followed as much is constant contact with our PO (Morten Salomonsen), mostly due to the fact that we have already established a common ground as to what the app should include as a minimum. He has however provided insight and answers to a lot of aspects of the app early in the project, and we expect to ask more in the future when we feel that the app has been further implemented.

Section 3: Utilities

We used Axosoft to start the process of making the product done by using the tools in Axosoft:

- Define the important user stories in Axosoft according to the product owner
- Choose the right user stories in first sprint according to the owner needless
- Determine a specific time for the first sprint between one and two weeks.
- Distribute the user stories in the first sprint between the group's members according to experience and availability of each one in the group.
- The tools that we use it to achieve every sprint:
    - Burndown and speedometer charts, where we keep tracking of our achievement and speed, so we make sure that we are always in the comfortable zone.
    - Card view is an amazing tool that helps us to move the user story from state to another like from in progress to ready for testing and so on...
    - Board members, where we assign individual tasks for every member.

Section 4: User Stories

Here is an overview of our product backlog as well as the sprint backlogs that have been performed or are being performed at that time.

The Sports Event app must meet each organizer needs, and participant needs (preparation, implementation and completion of exercise events.)

In general

As an organizer, you must be able to

- Create and manage events.

- Receive registration and payment from participants.

- Manage participants.

As a participant you must be able to:

- Register for events

- Start participation in registered events (start timekeeping).

- Receive correct and relevant information during the race (own location on map, time used, and distance traveled).

- End participation in events (stop timing, show participant result, add result to overall overview).

The Product backlog:

We have in total ten user stories in our product backlog

The sprint backlogs:

Sprint number	**User stories** [(I want to ..)..... (so that ...)]	**Argument**
	Story [1] As a [organizer], I want [to be able to create an event] so that [I can (name events, set registration deadlines, set start period, set target period and Upload route in GPX format or equivalent, enter geographical location of start and finish.)].	We chose to set these requirements inside one sprint, so we did the minimum requirements M1 , M2
	Story [2] As a [organizer], I want [to be able to manage events] so that [I can modify the information with	

4	which the event was originally created].	, M3 together because they directly belong to the organizer actions, which make it easier for us and so more organized.
	Story [3] As a [organizer], I want [to be able to manage participants] so that [I have the ability to change information that the participant has created in registration, I have the ability to unsubscribe participants, I have the ability to hide participant results, I have the ability to extract a list of participants, which includes start number, name, email, address, mobile, registered event and (when the participant has reached the finish line) time for completion , I have the ability to reset the time for a participant].	

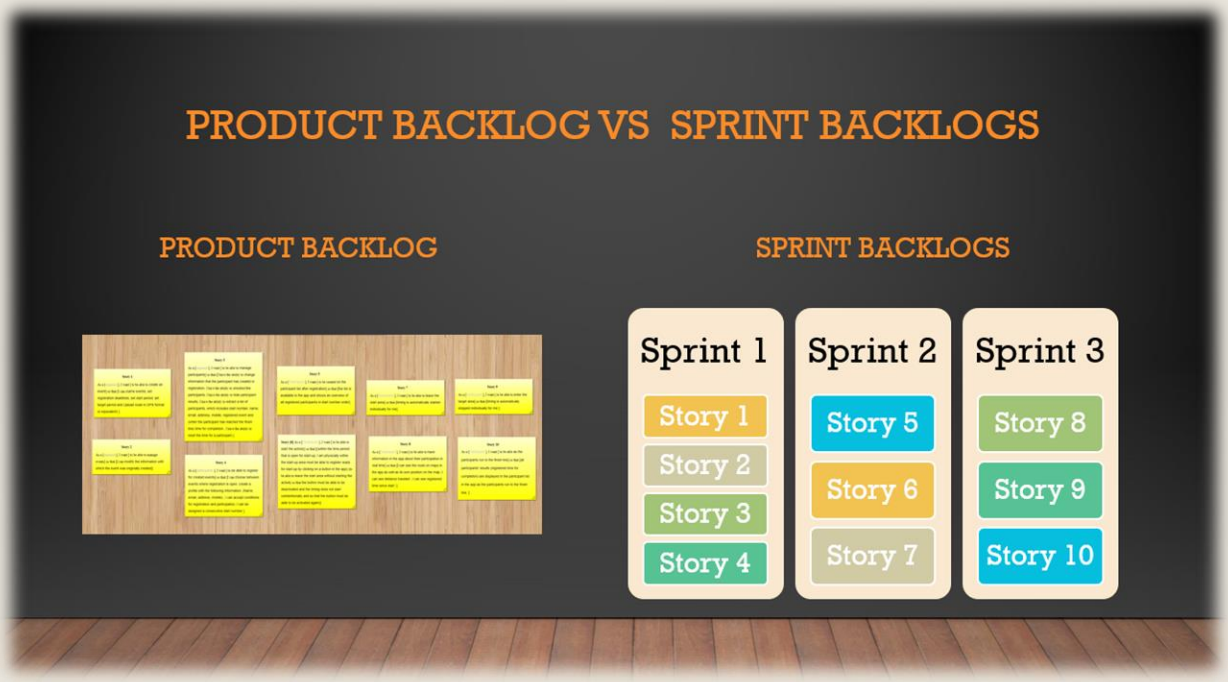
1-2 (need logic)	**Story [4]** As a [participants], I want [to be able to register for created events] so that [I can choose between events where registration is open, create a profile with the following information: (Name, email, address, mobile) , I can accept conditions for registration and participation, I can be assigned a consecutive start number].	We need to send every action that the user makes to the Firebase to be registered. The user can make a registration to the app and choose which event would like to join according to the opening time of the event. But the logic needs to be more precise according to the opening time.
3 (need logic)	**Story [5]** As a [Participant], I want [to be created on the participant list after registration] so that [the list is available in the app and shows an	The list of participants is ready, but the logic still needs to be done, like when the user

	overview of all registered participants in start number order].	joins any event will automatically join this list.
1	**Story [6]** As a [Participant], I want [to be able to start the activity] so that [(within the time period that is open for start-up, I am physically within the start-up area must be able to register ready for start-up by clicking on a button in the app) (to be able to leave the start area without starting the activity so that the button must be able to be deactivated and the timing does not start unintentionally and so that the button must be able to be activated again)].	We send every action the user makes to the Firebase to be registered, but we have to customize the finish button to make sure that the user can't press it until reaches 15 m close to the finish race point.

between	**Story [7]** As a [Participant], I want [to be able to leave the start area] so that [timing is automatically started individually for me].	We argued with the organizer that it is fine if we have just a button to make the user navigate with the start race event and finish race event. So, we don't need it acutely to make it automatically.
1	**Story [8]** As a [Participant], I want [to be able to have information in the app about their participation in real time] so that [I can see the route on maps in the app as well as its own position on the map, I can see distance traveled , I can see registered time since start].	Here the user will go to google maps so the user will see the route, position and distance traveled. When the user is done the race, so the user can see registered time. But here we make an

		argument with the organizer that he wants the user to navigate with the maps inside the app (50 % is done).
between	**Story [9]** As a [Participant], I want [to be able to enter the target area] so that [timing is automatically stopped individually for me].	The Manual timing setting is ready. We argued with the organizer to at least make it manually. That it's fine if we have just a button to make the user navigate with the start race event and finish race event. So we don't need acutely to make it automatically.

Not done	● **Story [10]** As a [Participant], I want [to be able as the participants run to the finish line] so that [all participants' results (registered time for completion) are displayed in the participant list in the app as the participants run to the finish line.].	



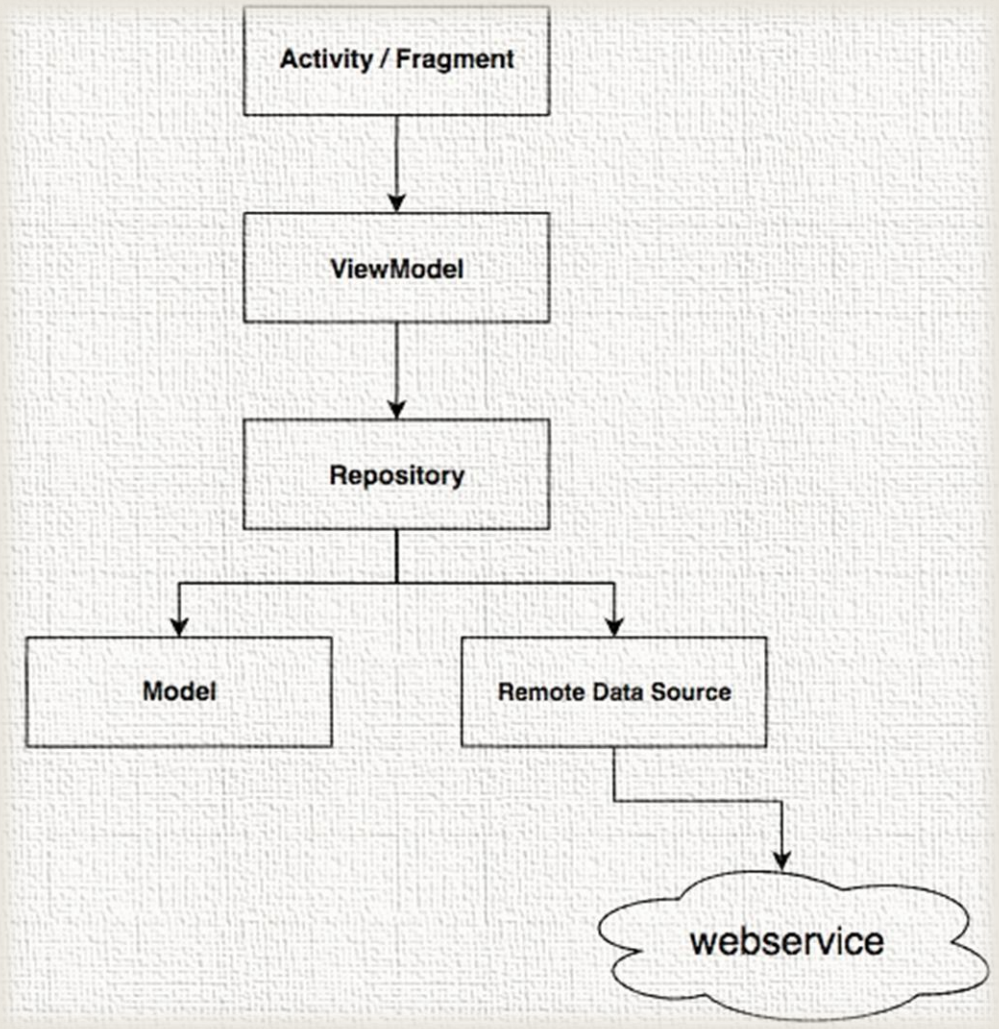
(Comparing between the product backlog and the sprint backlogs)

Section 5: Design considerations

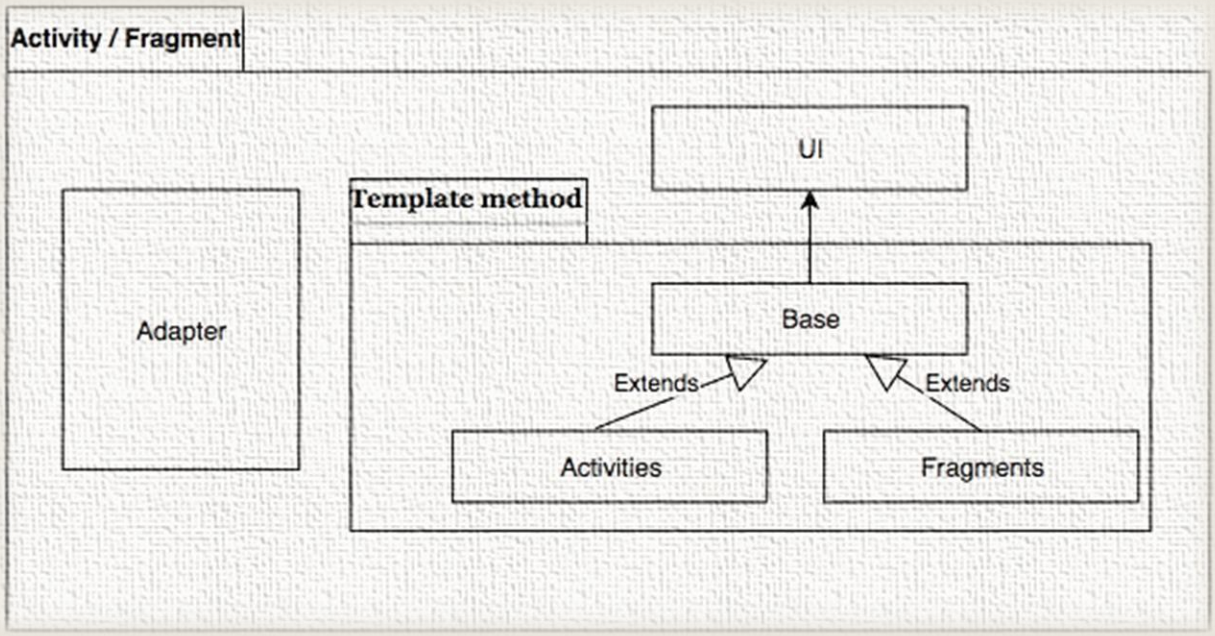
The architecture pattern that we use it in our project is Model view view model (MVVM):

The idea behind using MVVM is to separate Activity / Fragment from the logic. Activity / Fragment should not make any decision, they should just display the data from View Model. Another reason for this is separation of concern, so we do not mix it up. Every component has its responsibility.

The amazing part is the test part, because when we will make a test for View Model and Repository it will be done by just using Junit Test. For example to test the logic of the app inside View Model, but making a test for UI(Activity/Fragment) is a little bit harder, because we have to actually start an emulator or real device to test them, and this is a slow part.



MVVM consists of four things: UI Controller (activities & fragments), View Model, Repository and Data Sources



UI controllers are activities, fragments, and custom views. They observe the fields of the View Model and update themselves as well. At any time if the user makes an action with the UI, they will knock View Model in some points and express whatever the user wants.

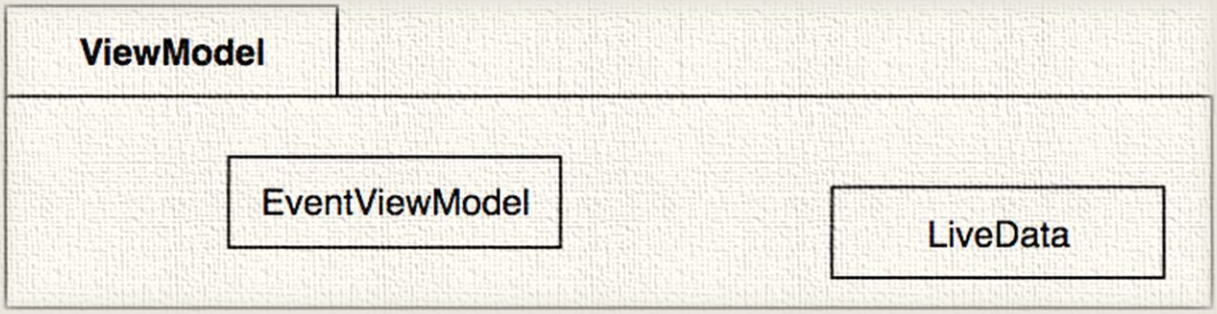
Template Method Pattern:

Base Activity and Base Fragment inside the base package will extend AppCompatActivity and Fragment. Then Activities and Fragments classes will extend Base Activity and Base Fragment.

We use this technique to define an abstract method in superclass, and make sure every class that extends Base’s classes should have an abstract method in superclass. BUT in template method pattern doesn't depend on just abstract methods, If there is any method that should be defined in multiple classes with the same content inside the method, but different messages come from parameters, so we define it in Base's classes, then we use it inside Activities or Fragments. Example: Toast massage, default data in memory, and so on ...

Adapter design pattern:

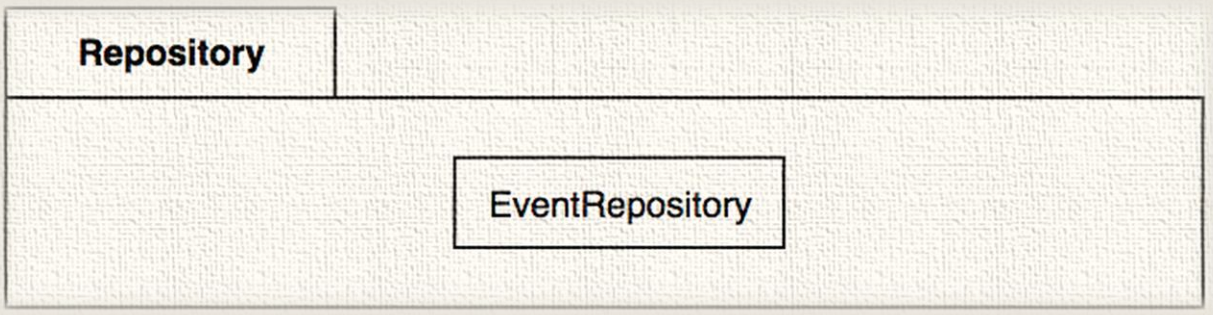
Convert the interface of a class into another interface, which the client will expect. An Adapter object acts as a bridge between an Adapter View and the underlying data for that view. The Adapter is also responsible for making a View for each item in the data set.



The View Model has the responsibility to get the data, it would be liveData, and the view model does not have any idea about UI, so there is no effect to view model, if any activity tries to recreate.

Observer design pattern:

Fragments will be a subscriber to EventViewModel, if there is any update or a new event is being created, so the EventViewModel will notify all fragments, that are subscribers, to update their data.

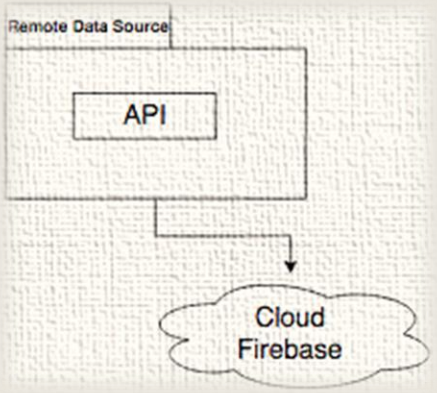


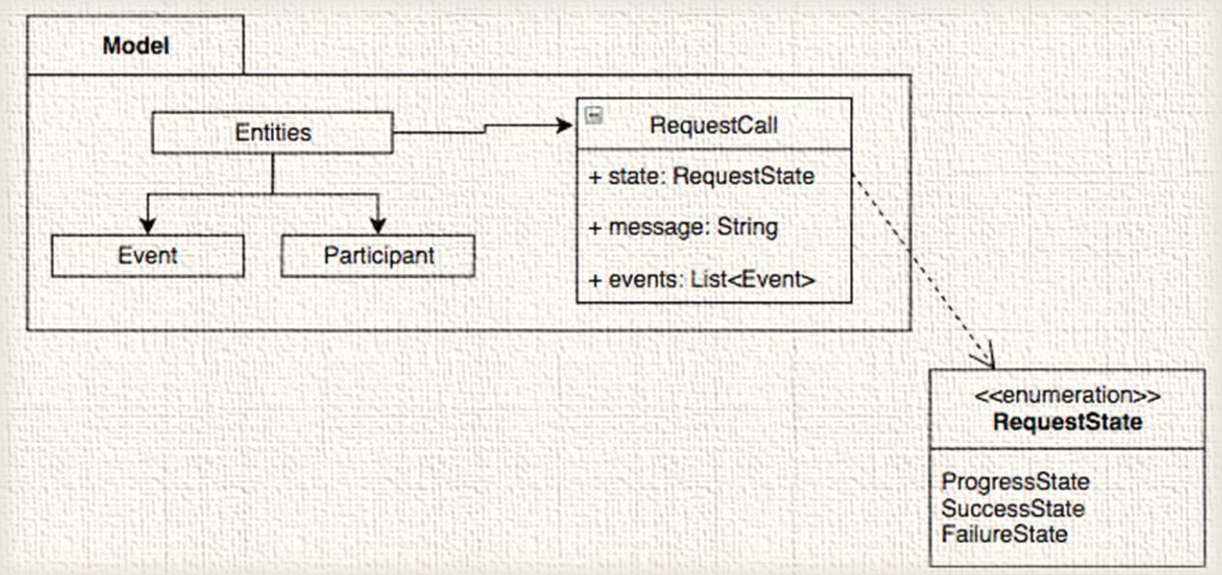
Repository is considered as a data store for the whole application, so we can pass an event ID, and it will return Live Data of Event, because it knows which API it should call to get the data from. The perfect calling to the repository is mediators between different data sources.

(Abstraction layer means: hiding the working details of a subsystem)

Repository builds an abstraction layout to whatever is happening below. For example: if we have a local database for cashing the View Model will not care about where this data comes from.

Data Sources, it could be anything Retrofit, SQLite storage, Room, or talking to other content providers API (like a rest API client. We can think of an API as a mediator between the client and the resources they want to get).





Request State:

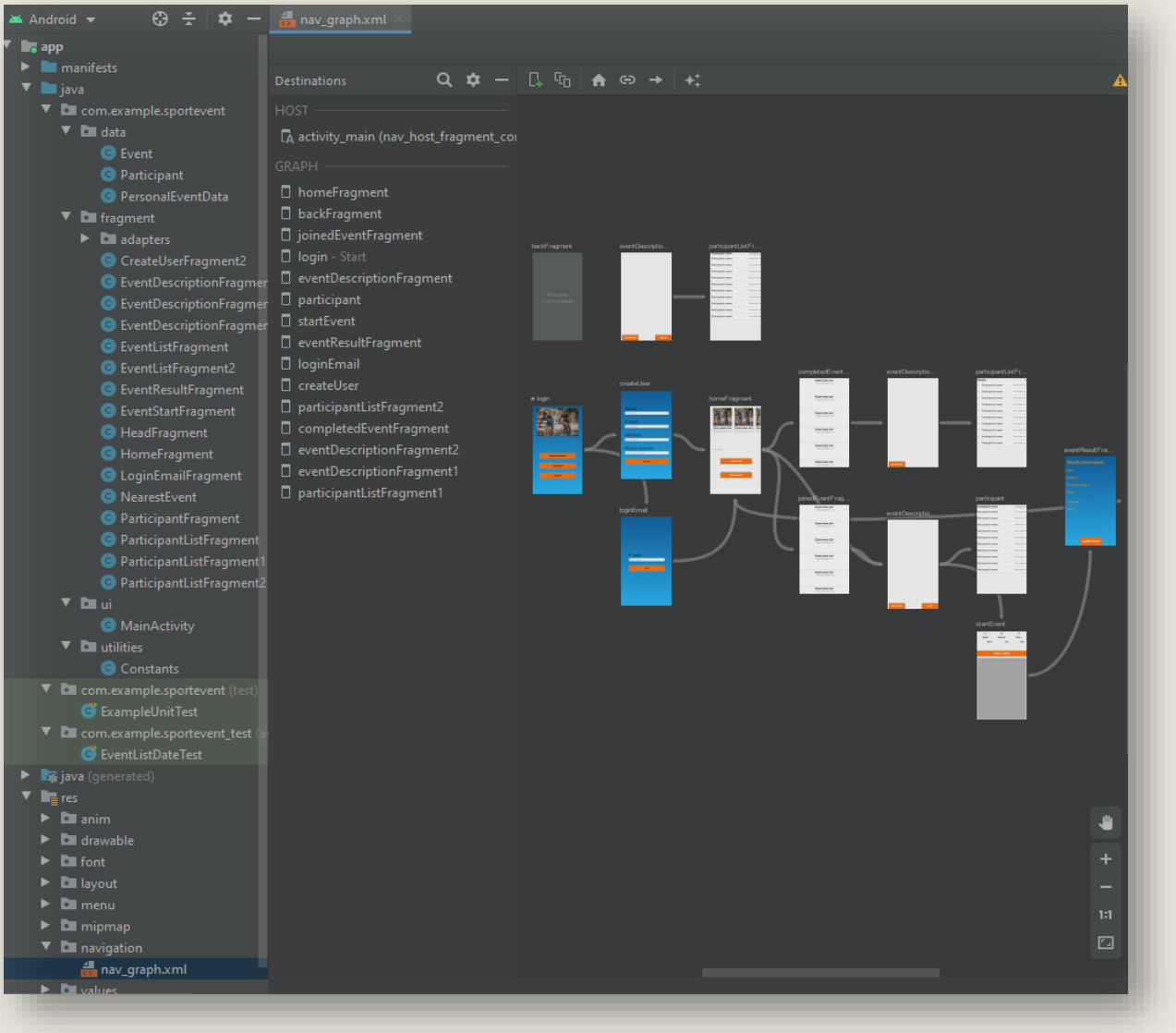
Request Call class will have Request State some attributes to keep tracking of the state of accessing the data from firebase.

When the repository sends the request to firebase, so the request will be in progress state, then when the repository gets response back, two possibilities could happen: success request or failure request, and the state of Request Call class will be changed according to that.

Section 6: Evaluation and documentation of sub-products and process progress

So far throughout the project, our work process has been to hold a meeting, assign tasks to each person or “work-duo”, and then showcase and explain the progress made to the group on the next meeting, by share-screen on zoom.

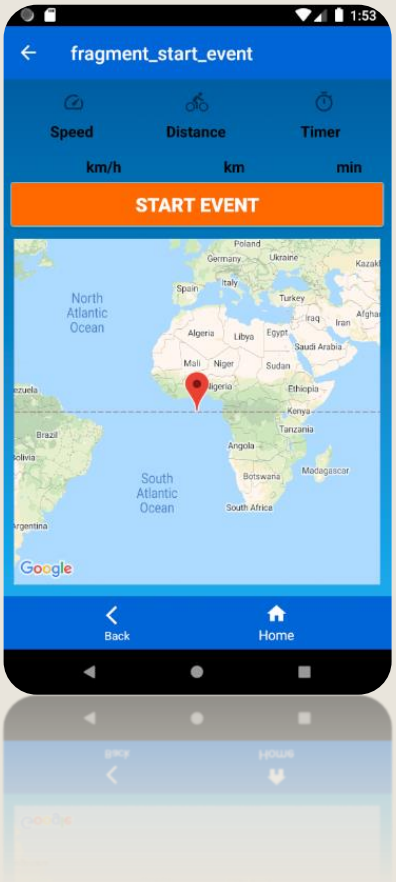
What we have to present at this moment is mainly the foundation of the app we are trying to create, the “skeleton” if you will! At this moment we are very close to knowing how a user will be navigating through the app, what is left is mainly the implementation of main functions (GPS tracking, firebase, general logic implementation) and in the end perhaps UI refinements.



Above is a screenshot of what our main project branch currently looks like, it does deviate a little bit in our personal branches however not by a lot, so this picture is a good reference of our current project state. The xml file shown is our current (and likely final) method of navigating between the screens, it gives us a nice overview of our screens, and makes transitions smooth and easy to implement. Our PO (Morten Salomonsen) has

been helpful as well in deciding the structure of how a user would navigate through the screens, so what you see on the picture above is cumulated effort by both our PO and us.

Currently we have one activity that holds all the other screens as fragments, with a few lingering buttons that will provide a QoL experience for users, such as the Home button in the example app screen.



Burndown Chart:

We are using the Time Estimate, we can know the amount of work in the sprint (or release).

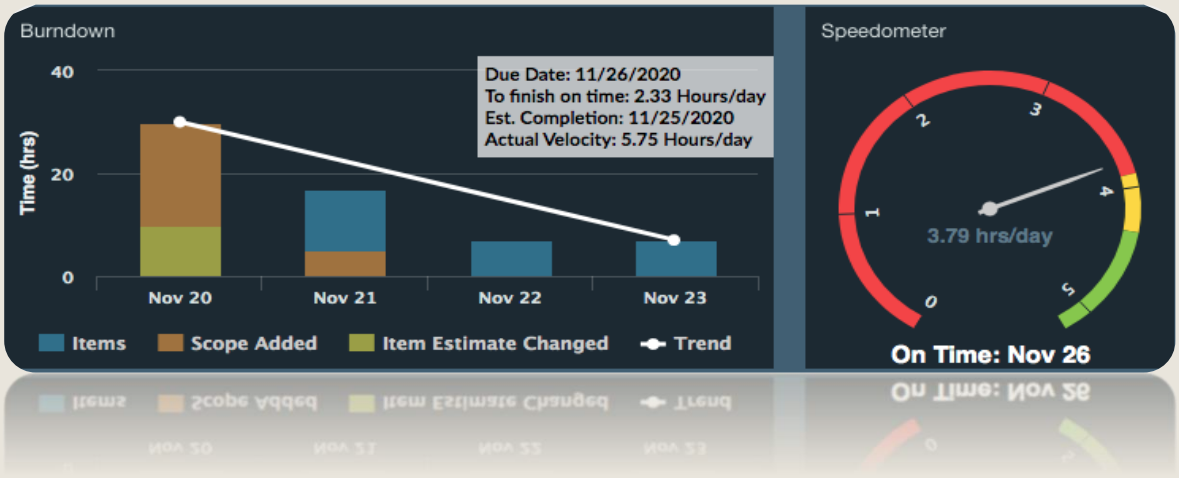
Burndown charts calculate effectively how much time is remaining in our sprint and if we will ship on time within the required velocity, or late. Axosoft calculates velocity

based on the difference between the work remaining for all items in the release when they were added, and the current work remaining.

The remaining estimate, or the amount of time presented per day for items is decreased typically in one of two ways: by adding work logs to items to show work done, and by completing items which updates their remaining estimate to zero.

We can update this value every day and plot it on the chart.

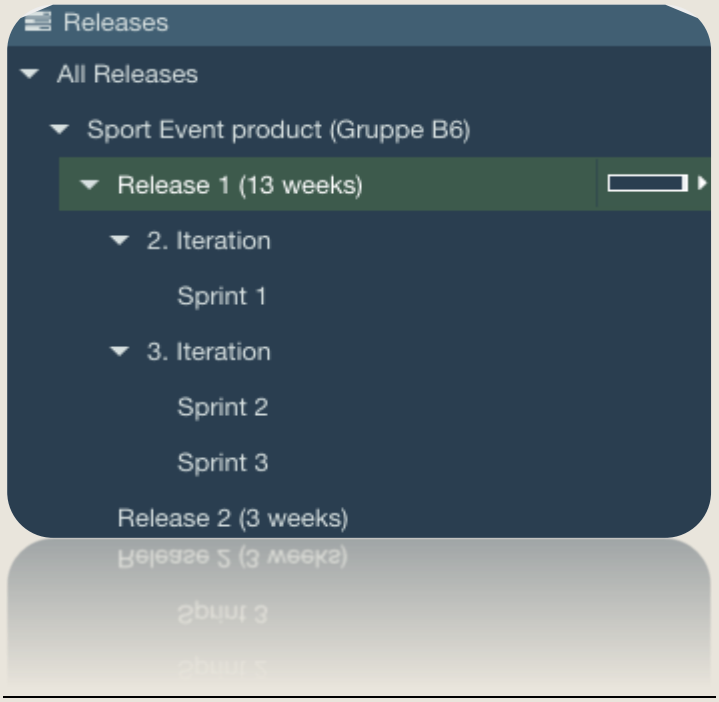
Release due dates are tracked against velocity when determining a projected ship date to display on the Speedometer. If the ship date is on time or earlier than the due date, meeting required velocity, it will display in green. If our release is running behind, it will be red with a warning indicating that it is late. Burndown Chart helps us to know if we are on track or late.



(The picture here shows the progress of our work during sprint 3)

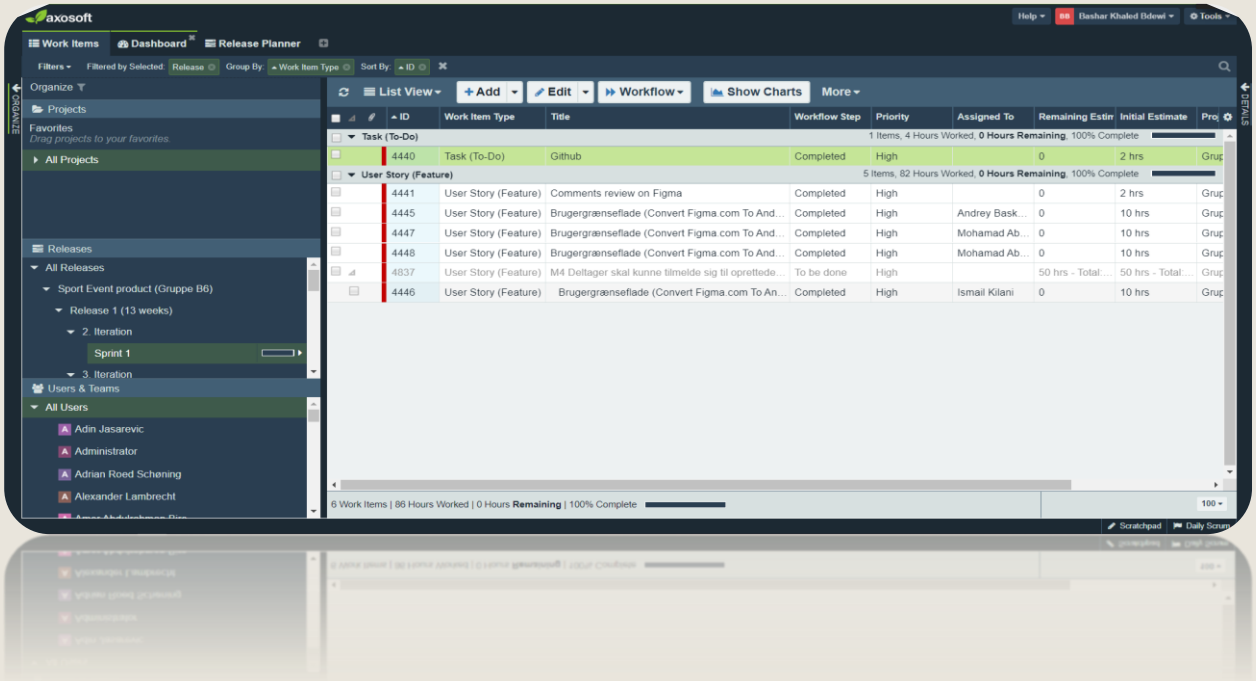
Release Planning

Our project is structured and divided based on our use of Axosoft



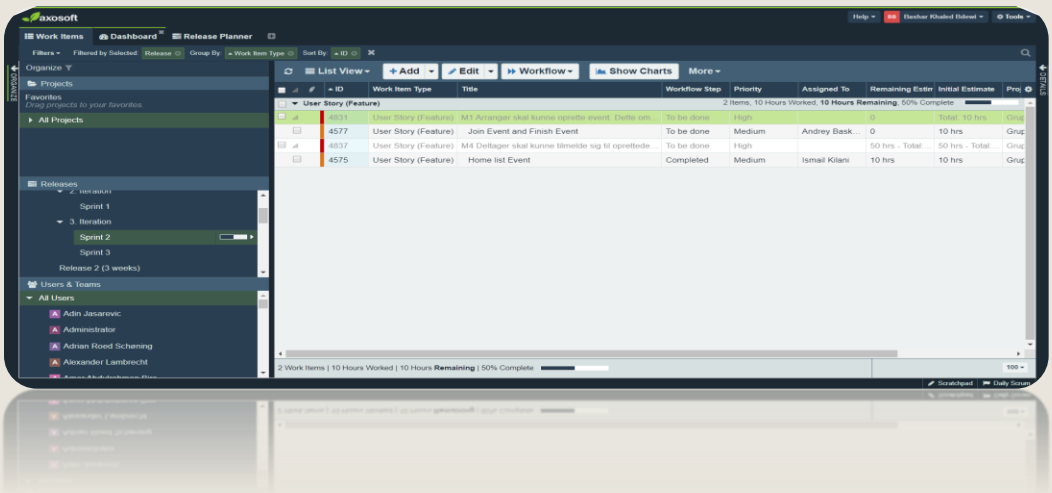
Sprint 1:

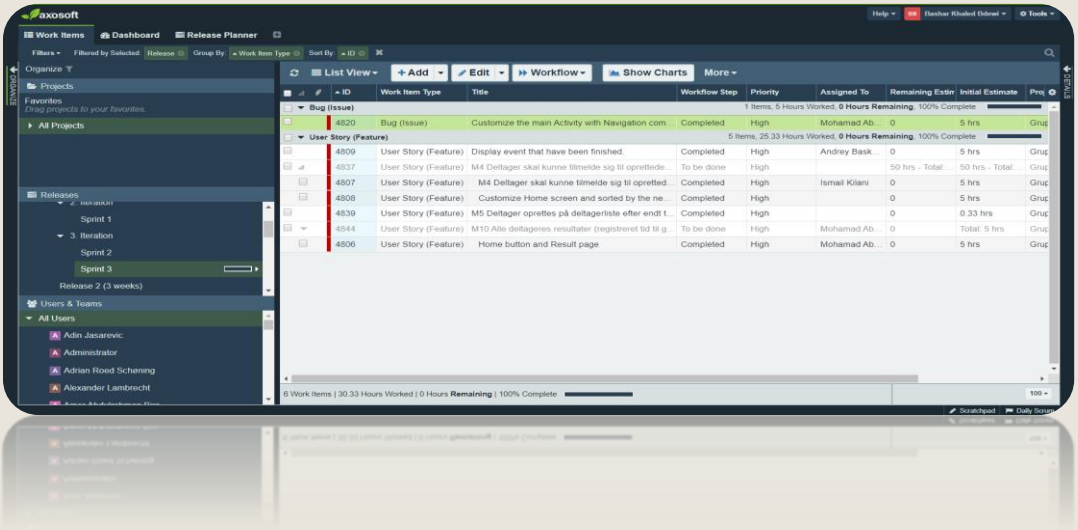
According to the use of Axosoft in our project, we have divided our work into two releases, the first release is the release during the13 weeks period of the semester, which includes two iterations. In iteration 2 we have only 1 sprint according to the stories should be according to the product owner choose and according to the requirement of assignment in that iteration.



Sprint 2 and Sprint 3:

In iteration 2 we split our work to 2 sprints, where the first sprint we continued working with the user stories that are not done in sprint 1, and in sprint 3 we worked with more user stories according to the product owner and requirement to iteration 3.





Our second release will be launched at the beginning of January, we will be working during a 3-week period. You can see the plan under the plan section.

Sprint Review:

Review the sprint between the team and Scrum Master during every meeting.

We had a discuss about three points, every one explain:

- What did he complete?
- What will he do today?
- Are there any problems?

Sprint Retrospektive:

It was with the Product Owner and Scrum team.

We had a discuss about three points, every one explain:

- What went well?
- What could be improved?
- What will we do in the next Sprint?

Section 7: Conclusion and plan for the 3-week period

Generally, this project has run smoothly, however the work that was delegated to each member most of the sprint has not always been completed, and so we as a team part of us might be slightly behind our own schedule. This could maybe have been avoided had we better communicated our issues related to the project to the other members and worked them out in small groups before the sprint ends.

For the 3-weeks period our plan is to focus much more on the logic part of the app, now that we have a clear overview of our project and we are somewhat in the middle of it.

We have also focused a lot on how to make the app more user-friendly, and made ideas of how the app generally might function and transition from screen to screen, what is now mostly left is part of functionality. We might work with wish-requirements, if we are done with the must-requirements in the 3 weeks period.

The plan is to have three sprints:

- Sprint 4: Continue working with the users stories
- Sprint 5: Correct and customize sprint 4 according to the product owner review, and make all must requirements done (users stories)
- Sprint 6: Test the app according to the test plan and correct if there is anything should be corrected

Test Plan:

1. Is the content presented in a way that is easy to find and understand?Can Has anyone completed a task successfully?
2. In lab or in field:
    - Try to start the app and ride a bicycle then finish the app and see how the was it?

- In-person testing shares the phone screen and user cam with us so we can see such as body language and user interaction.

3. Try to test with people (1-3) which have different ages. Maybe try to cover the experience with the app in different weather?

Section 8: Bibliography

https://docs.google.com/presentation/d/1wNqd17dj0AeNm6yFdtucFLZCzZa0z3d9ewPnhZlfWBU/edit#slide=id.g2685f6ea6c_1_110

https://docs.google.com/presentation/d/1pNWuyxkI3dKINoaHh8LHhoYmyYJCX11zv5PATNOU5fM/edit#slide=id.g98f9110974_0_5

Design Patterns by Grady Booch

https://developer.android.com/jetpack/guide

https://docs.google.com/presentation/d/1O-dBrDDxZbLiVJvAmia4j-781c0_0vRhgoguqikWUOM/edit#slide=id.g24e247e3a0_2_0

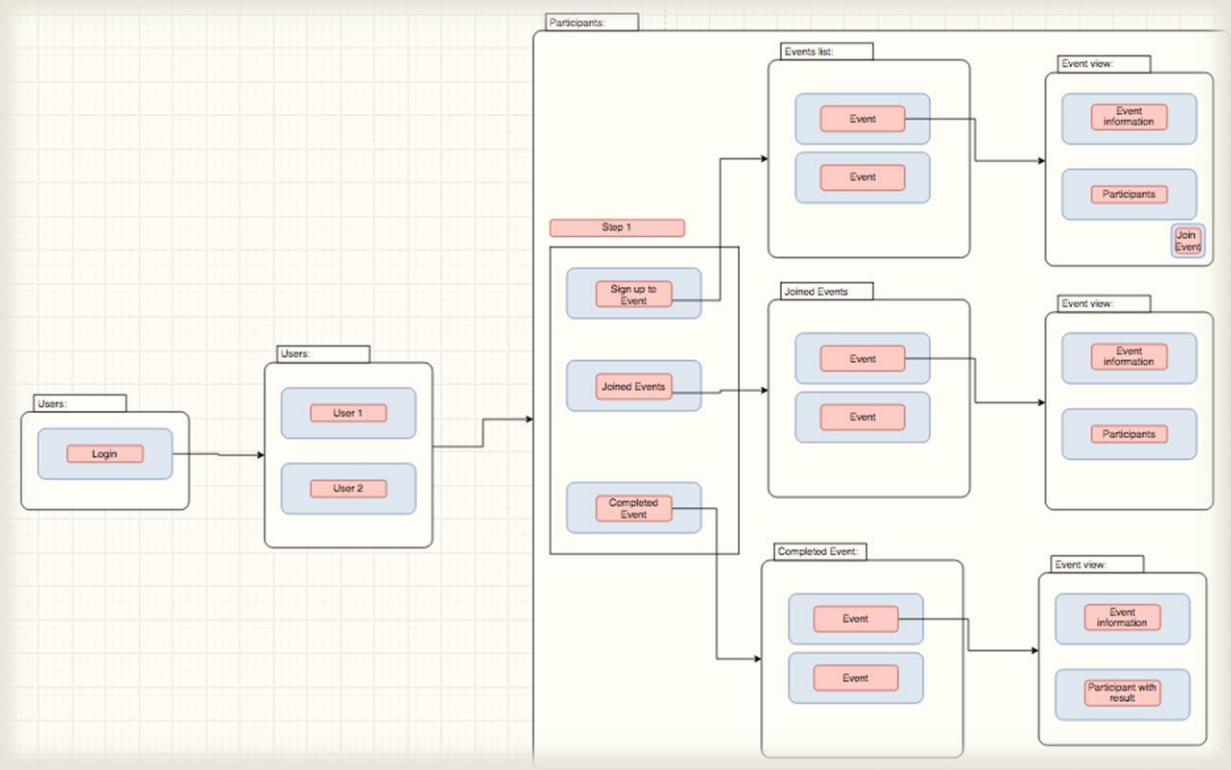
https://en.wikipedia.org/wiki/Abstraction_layer

https://drive.google.com/drive/folders/1KmQ5MrhesZ8QVPIrL9VZWgw4b-Epe2gZ

Section 9: Appendix

State diagram:

The first view of our product, which describes the navigation between the user and the product.



Architecture pattern Model view - view model (MVVM):