Report for Hubba

Grupp 15

**Johannes Gustavsson, Nils-Martin Robeling, Li Rönning,
Alexander Selmanovic, Jian Shin, Camilla Söderlund**

Objektorienterat Programmeringsprojekt TDA367
Chalmers Tekniska Högskola
Sverige
October 2018

1 Introduction

With a schedule that can vary greatly from day to day either as a student, a businessman or as a forgetful person it can be hard too keep or create new habits.

This android application will keep track of habits the user wants to create and keep. By creating a habit to do either daily, weekly or monthly and having it to show up in the app on those days, you can easily keep track and remember to do them. It uses notifications that are set by the user, to help them remember when to do their tasks even if you don't have the app running. The app also helps motivate the user with the help of streaks and achievements.

1.1 Definitions, acronyms, and abbreviations

- **User**: A user is a person that has an account for the application and therefore can log in and use the application.
- **Habit**: A habit is an action that the user wants to repeat regularly and keep track of. For example a habit could be washing your face every morning or watering the plants once every week.
- **Group**: A user can be a part of a group of users. In this group they have shared habits. For example if the users want to work out two times a week together they can use groups and keep track of each other and stay motivated.
- **Streak**: When a user has done a habit five days in a row, the user will receive a streak next to the habit on the front page. The streak represents the number of times in a row the user has completed a habit. The streak is visible on the front page until the day the user forgets to do the habit.
- **Achievement**: Users can unlock achievements when they use the application, there are two different types of achievements:
  - **Streak achievement**: when a user has performed a habit a number of times in a row they unlock streak achievements. For example doing a habit ten days in a row.
  - **Number of habits achievement**: When the user has created a number of habits they unlock this kind of achievement. For example having a total of five habits.

2 Requirements

2.1 User Stories

1. **Story name:** Creating habits

   Description: As a user I want to be able to to create a new habit so that I can keep improving my self.

   Confirmation:

   Functional: - Be able to create a new habit.
   - Have the habit added to the right list.
   - The habit should be connected to the correct user and saved.

2. **Story name:** Delete habit

   Description: As a user I want to be able to remove habits so that I only keep track of relevant habits.

   Confirmation:

   Functional: - Being able to delete a habit.

3. **Story name:** Check off habits

   Description: As a user I want to be able to check of the habits I have done so I can see what I have left to do.

   Confirmation:

   Functional: - Boolean isDone in a Habit can be set true or false.
   - When checked of it should be moved to done list.

4. **Story name:** Undo checks

   Description: As a clumsy person I wanna be able to undo my 'check off' so that if I pressed the wrong habit it returns to its previous state.

   Confirmation:

   Functional: - Internal logic for when it's done to be reset to not done.
   - Should be shown in their previous category again.

5. **Story name:** Reset habits

   Description: As a user I want the list of daily tasks to be reset each day so I can check them off again the next day

   Confirmation:

   Functional: - The tasks that was marked as done should be set back to undone daily.
   - Streak should be kept when the daily reset goes through.

6. **Story name:** Get streaks

   Description: As a user I wanna be able to get streaks when I've done a specific task multiple days in a row to make it more fun to do my habits.

   Confirmation:

   Functional: - Keep track of number of consecutive days habit has been done.
   - To be shown when streak reaches a set number.

7. **Story name:** Change themes

   Description: As a user I wanna be able to change the color theme on the application to make the user experience more personal.

   Confirmation:

   Functional: - Being able to change theme.
   - The theme should be changed on all pages it's supposed to.

8. **Story name:** Reminders

   Description: As a user I want to be able to set a reminder so that I get reminded about a task and don't forget about it.

   Confirmation:

   Functional: - Be able to set reminders.
   - Reminder should be shown when app is turned off.
   - Reminder should be shown at the right time.

9. **Story name:** Notifications

   Description: As a user I want to be able to adjust the way the app communicates (Notifications) with me to fit my personality.

   Confirmation:

   Functional: - Enable or disable notifications.
   - Receive notifications if enabled.
   - Different types of notifications depending on set mood.

   Comment: Not Implemented.

10. **Story name:** Share habits

    Description: As a user I want to be able to add friends to the app so that we can share our habits.

    Confirmation:

    Functional: - be able to find friends
    - add friends
    - share habits with friends

    Comments: Not implemented.

11. **Story name:** Have friends

    Description: As a user I want to be able to add friends so I can quickly find them without remembering their username.

    Confirmation:

    Functional: - Being able to add friend.
    - Your friends should be shown under friends menu.

12. **Story name:** Remove friends

    Description: As a user I want to be able to remove friends so I have relevant friends in my list.

    Confirmation:

    Functional: - Being able to remove friend.

13. **Story name:** Compete with friends

    Description: As a user I want to be able to add friends so that we can compete in who completes their habits.

    Confirmation:

    Functional: - Add friends.
    - Compare achievements with friends.

    Comments: Not implemented.

14. **Story name:** Achievements

    Description: As a user I want confirmation that i am progressing through achievements so that I feel encouraged to continue.

    Confirmation:

    Functional: - Having Achievements to fulfill.
    - Receive achievements when goals are reached.
    - Being able to see achievements your achievements
    - Getting a notification when an achievement is fulfilled.

    Comments: Notifications not implemented for achievements yet.

15. **Story name:** Statistics

    Description: As a user I want to keep track of my progress so that I feel motivated.

    Confirmation:

    Functional: - Access information about done habits over time.
    - Put saved information together and show to user

    Comments: Not implemented.

16. **Story name:** Access account

    Description: As a user I want to be able to create an account so that if I get a new phone I can have access to my account on the new one.

    Confirmation:

    Functional: - Create a user account.
    - Externally save accounts.

    Non-functional: When logged out, should not be able to press back to get into the account again.

    Comments: Only local accounts are available.

17. **Story name:** Save data

    Description: As a user I want my user account and habits to be saved when I exit the application so that what I've done won't disappear.

    Confirmation:

    Functional: - All important data should be saved.
    - Data should be saved when exiting the application.
    - Data should be saved when logged out.

    Non-functional: When new types of data is added it should be easy to have it saved.

    Comments: Current system is a temporary solution so that functionality that is dependent on the save and load functions will work. A better more sustainable one need to be implemented later.

18. **Story name:** Load data

    Description: As a user I want my user account and habits to be loaded so that when I start the application I can continue working on my good habits.

    Confirmation:

    Functional: - Correct data should be loaded.
    - Data should be loaded when logged in.

    Non-functional:

19. **Story name:** Have an overview

    Description: As a user I want to be able to overlook upcoming and past habits so I can easily take them into account when planning.

    Confirmation:

    Functional: - Implement a calendar to have an overview.
    - Keep information of habit frequencies to show in calendar.

    Non-functional: -

2.2 User Interface

The user interface has received less attention due to the focus of the project being to create an object oriented program and the design of the code. In this section the GUI design choices will be presented and explained.

The first part of the application that the user meets is the login screen (see fig 1 (left)). It has a basic layout with two fields with input prompts where the user can enter their username and password. After entering valid credentials and pressing the login button the user is taken to the applications main page. If the credentials are invalid, the user will remain on the log in page.

If the user doesn't have an account there is a button called "NEW USER" that will take the user to a page where they can create one.

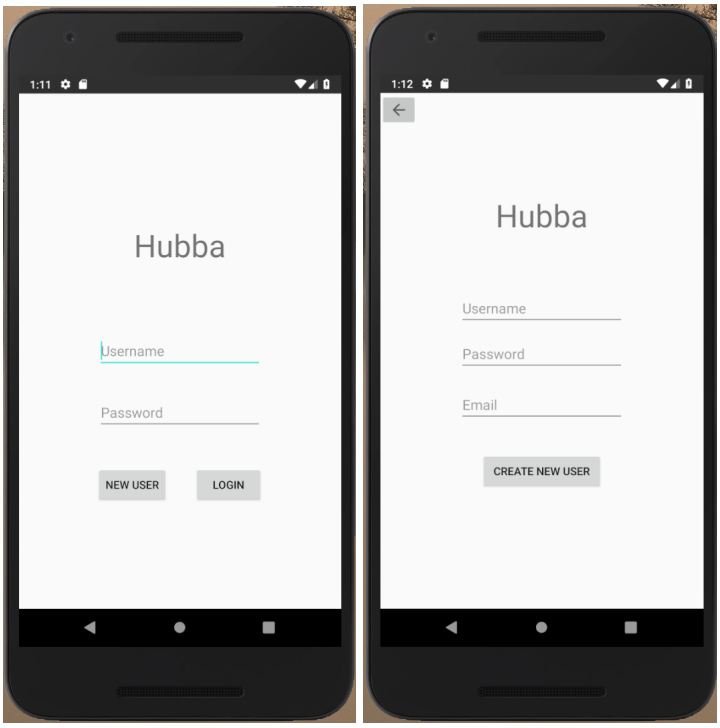

Figure 1: Screenshot of login screen(left) and Screenshot of create new user(right)

To create a new user there are some fields that are necessary to fill in so that the app will have enough information to be able to operate. The username and password are needed to be able to login. The email is required for future functionality, such as recovering a username or resetting a password. At the bottom of the interface a big button that says "CREATE NEW USER" can be seen (See fig 1 (right)). When pressed, an account is created for the user and they can use it to log into the app.

On the main page the habits are shown under the categories "Morning", "Midday", "Evening" or "Night" depending on what state and category the habit has been given, as you can see in fig 2. It can also be placed under "Done" if the user has checked it off. These categories are there so that the user easily can get an overview of what habits should be done during the day with clear separations between the different categories.

On the main page there are three navigational buttons. In the top left corner there is a menu button which takes you too the menu. In the top right corner there is a button to get to a calendar view of when the habits are to be done and down in the right corner there is a button that lets you create habits on a separate screen. Due to this being an habit application for phones and that most people are right handed the button to add habits was put down to the right for easy access.

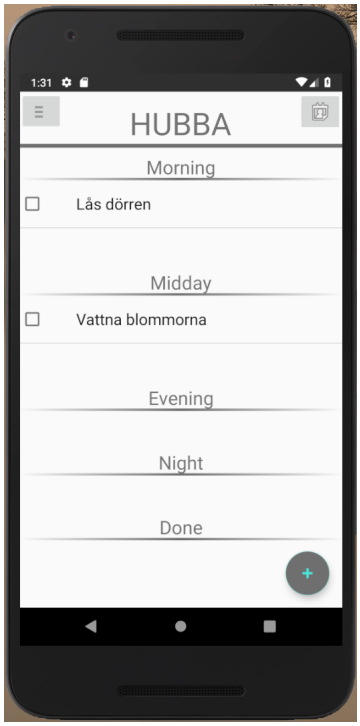

Figure 2: The front/main page with a few habits added

When the button for adding a new habit is pressed a new activity is shown, create habit, that has with only have a few buttons. These buttons impact the GUI and depending on the users choice different options will be shown as you can see in figure 3.

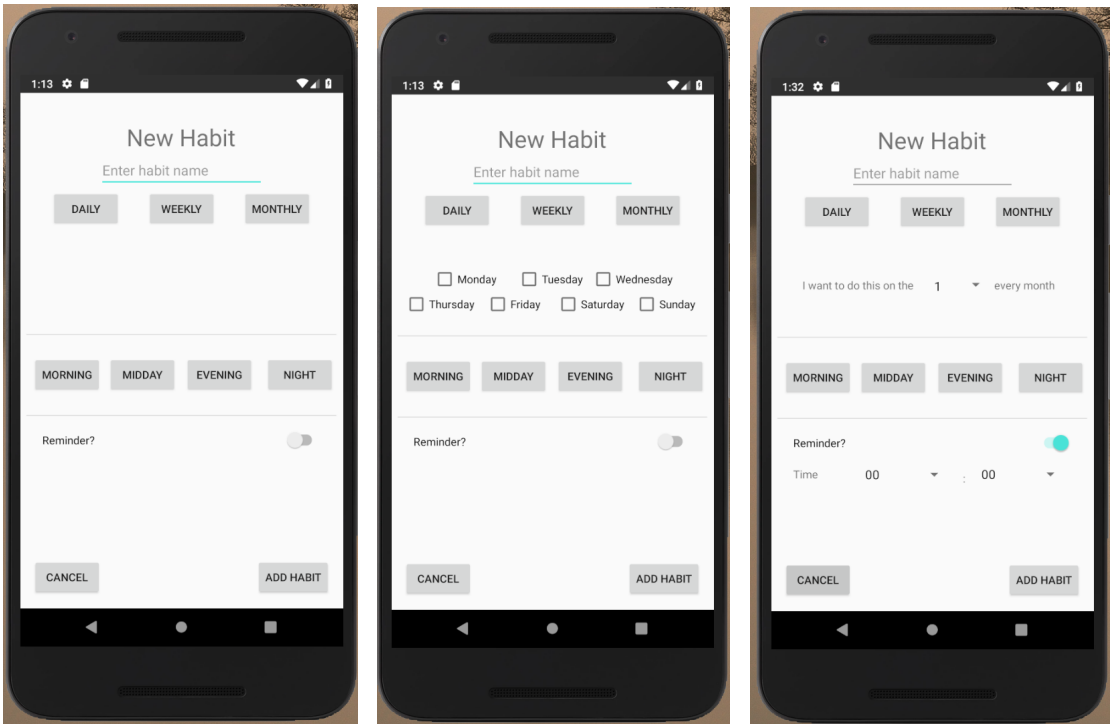

Figure 3: Different layout depending on if daily, weekly or monthly have been chosen in that order

- Daily: If the habit is to be done daily there is no special options to be chosen.
- Weakly: When weekly is chosen the user is able to chose on what days of the week it is supposed to be done.
- Monthly: When monthly was chosen the user are able to chose what date to do it on.

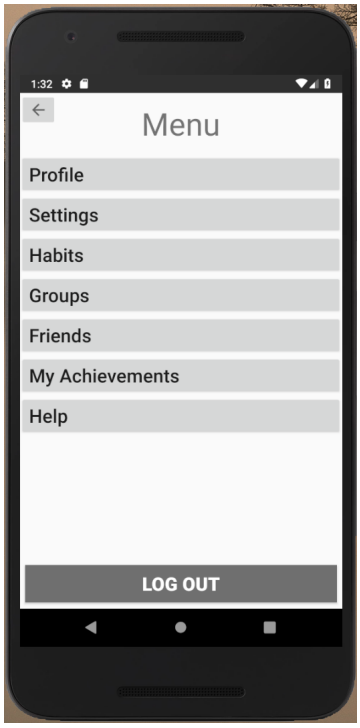

Figure 4: Layout for menu page

When you are open the menu from the front page the menu will be shown as fig 4. From this page you navigate around to different pages. To make the user more sure about the app and log in function, the app gives the user an option to log out with a prominent done button.

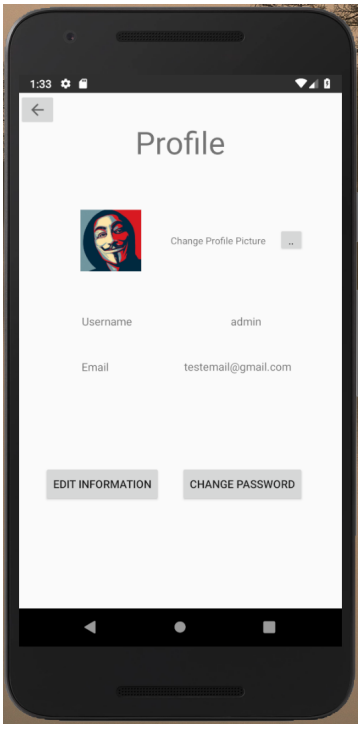

Figure 5: The user's profile page.

On the profile page the user can overview their current profile information, including a profile picture (see fig 5). It is also possible to change the information.

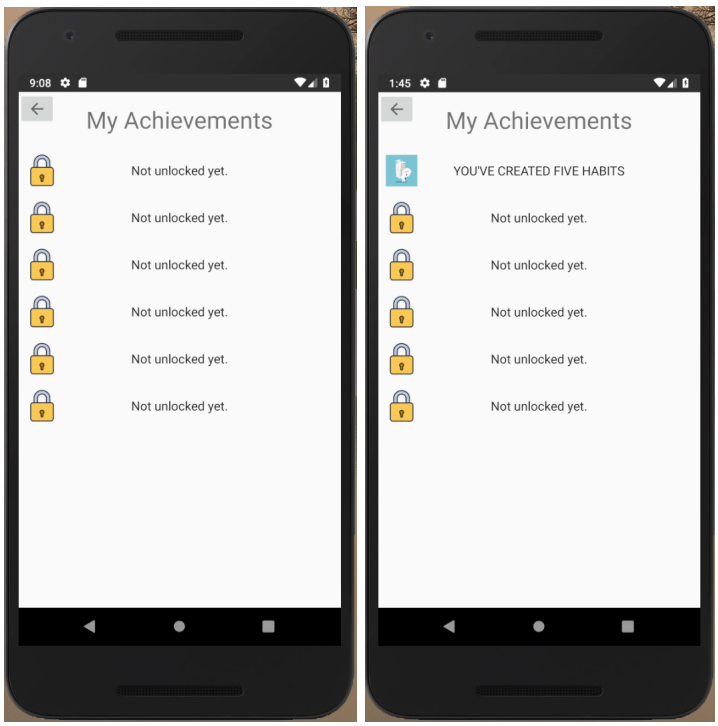

Figure 6: When one achievement has been unlocked

When a user reaches the criteria for a certain (preset) achievement, said achievement is unlocked on the achievements page (see fig 6).

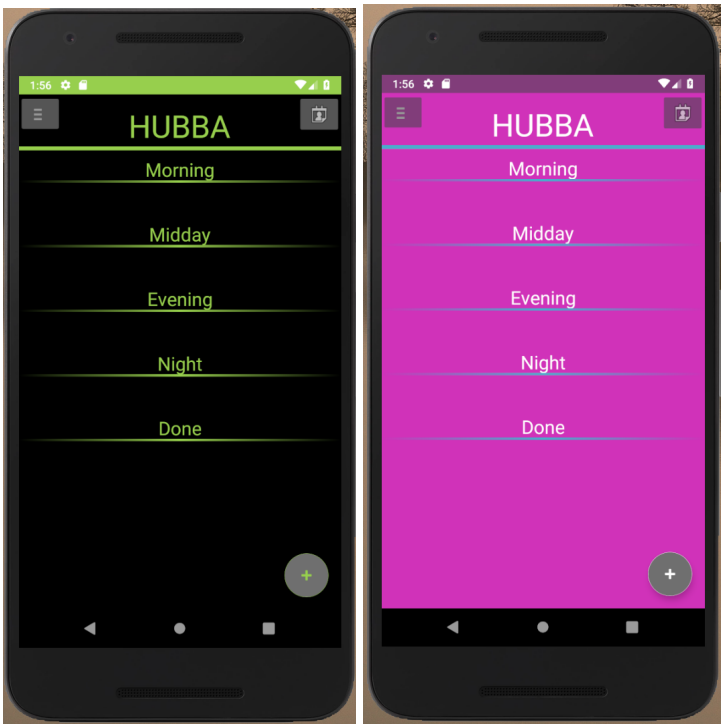

Figure 7: The front page shown in Elite theme on the left, Pink fluffy theme on the right.

A user can choose between two built in themes as shown in fig 7. When a theme is set, all the pages in the app changes appearance accordingly.

3 Domain Model

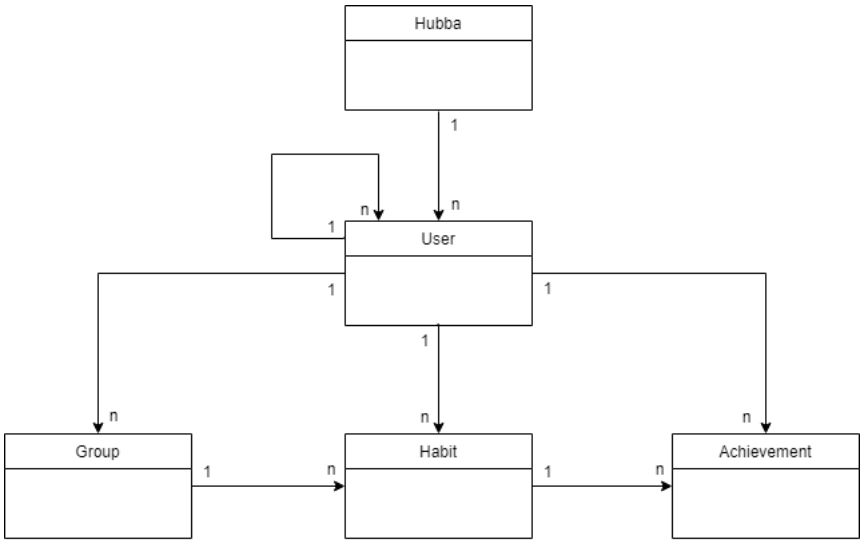

Figure 8: Hubbas domain model

3.1 Class Responsibilities

The different classes that are shown in the domain model have different responsibilities. They all belong to the model module, to which there are more classes, but these are the most relevant ones.

3.2 Hubba

Hubba works as a facade for the model module, which mainly consists of the classes shown in the domain model. All communication between the model and the other modules, view and ViewModel, go through Hubba. Another one of the main responsibilities for this class is to keep a list of all the users, and to know which user is currently using the app.

3.3 User

The user class represents the user and keeps information about the users name, email, password, habits, groups and achievements. It also keeps track of whether the user wants notifications from the app or not, and sound settings. So when the saved user logs in the settings will be the same as they were last time.

3.4 Group

If the user wants, it is possible to create group within the app, and share a habit with the other users in the group. This class is responsible for keeping the name of the group, a list of the users in the group, and a habit that the group shares. It also checks if the habit is completely done, meaning that all users have completed the habit.

3.5 Habit

This class represents a habit, which is what the application is based on. It holds all the detailed information about a habit:

- Title: what the habit is called.
- Streak: how many consecutive times the habit has been performed.
- State: a habit has several different states. Such as time of day, how frequently they should be done, and if it's a single user habit or for a group.

The class also makes sure that the different states can be updated.

3.6 Achievement

To keep the user motivated and to make them continue to build good habits, achievements are added. This class has the information of an achievement, such as name, a target number to reach, if it has been achieved or not and method to evaluate if it's achieved.

4 System architecture

In this section the system architecture and necessary knowledge of the application will be presented if the project is picked up by a new programming team or member.

4.1 Android Basics

The application is developed and tested in Android studio with SDK28 and built in Gradle. It supports API28 and added support for lower APIs are to come in a later stage. This choice was made to as quick as possible get a working application to present with the most important functionality implemented.

4.2 Project structure

The system is constructed according to the Model-View-ViewModel (MVVM) pattern, an overview of the pattern can be seen in figure 9 below .

4.3 About MVVM

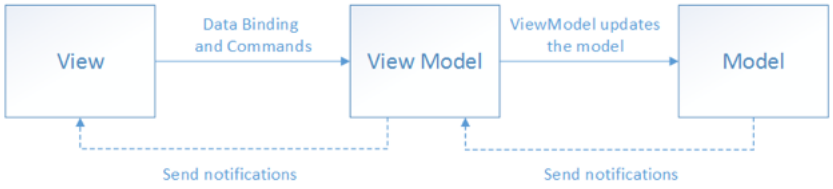

Figure 9: MVVM

This pattern is similar to the MVC, but the controller is replaced with a View Model. The modules can be single classes or packages with several classes.

4.3.1 The Model

The model encapsulates the applications data and is invisible to the user. It should represent the domain model of the project and be completely independent of the view and view model.

4.3.2 The View

The view is responsible of what the user sees on screen, meaning structure, layout and appearance. Each part of the View is defined in an XML document which does not contain any business logic. The view communicates with only the view model through commands or events like buttons and selecting an item.

4.3.3 The View Model

The view model holds properties and commands to which the view can data-bind (the process of binding the UI together with the business logic) to. The ViewModel should be holding all the functionality the view displays but the view should choose how to display it. Another responsibility introduced to view model is to keep track of any interactions with the model the view might request, the view model may then send data back or expose the model to the view (the model should support data-binding with the view in this case).

4.3.4 Benefits using MVVM

- Unit tests can be written for the model and view model without having to depend on the view and still behaving as if the view was there. [1]
- if there is an existing implementation of the model that encapsulates present logic it could prove either difficult or risky to change it. In this event, the view model can exist as an adapter for the model to avoid making any major changes.

- The view can be changed or completely re-implemented without touching the code since it is all XML.
- It's easy to work with parallel as a team since GUI-design can happen at the same time as development of the view model and model.

4.4 MVVM in the application

In this section you will get an short overview of how the application have implemented the MVVM pattern.

4.4.1 Model

The model package is based on the applications domain model. It contains classes for users, habits, groups, achievements and the application. It is also independent of other packages and of android specific implementations.

4.4.2 View

The view package contains all the xml files. The xml files contains names for the parts the user can interact with and builds upp the graphical user interface. This part of the application tells the ViewModel what has happened and then the ViewModel handles that information.

4.4.3 ViewModel

The ViewModel package contains all files that keep state for the View and also updates the model.

It also receives notifications from the model to update the view when something has changed. When it receives these notifications it sends a notification to the view so that the view updates itself accordingly.

4.5 Design model

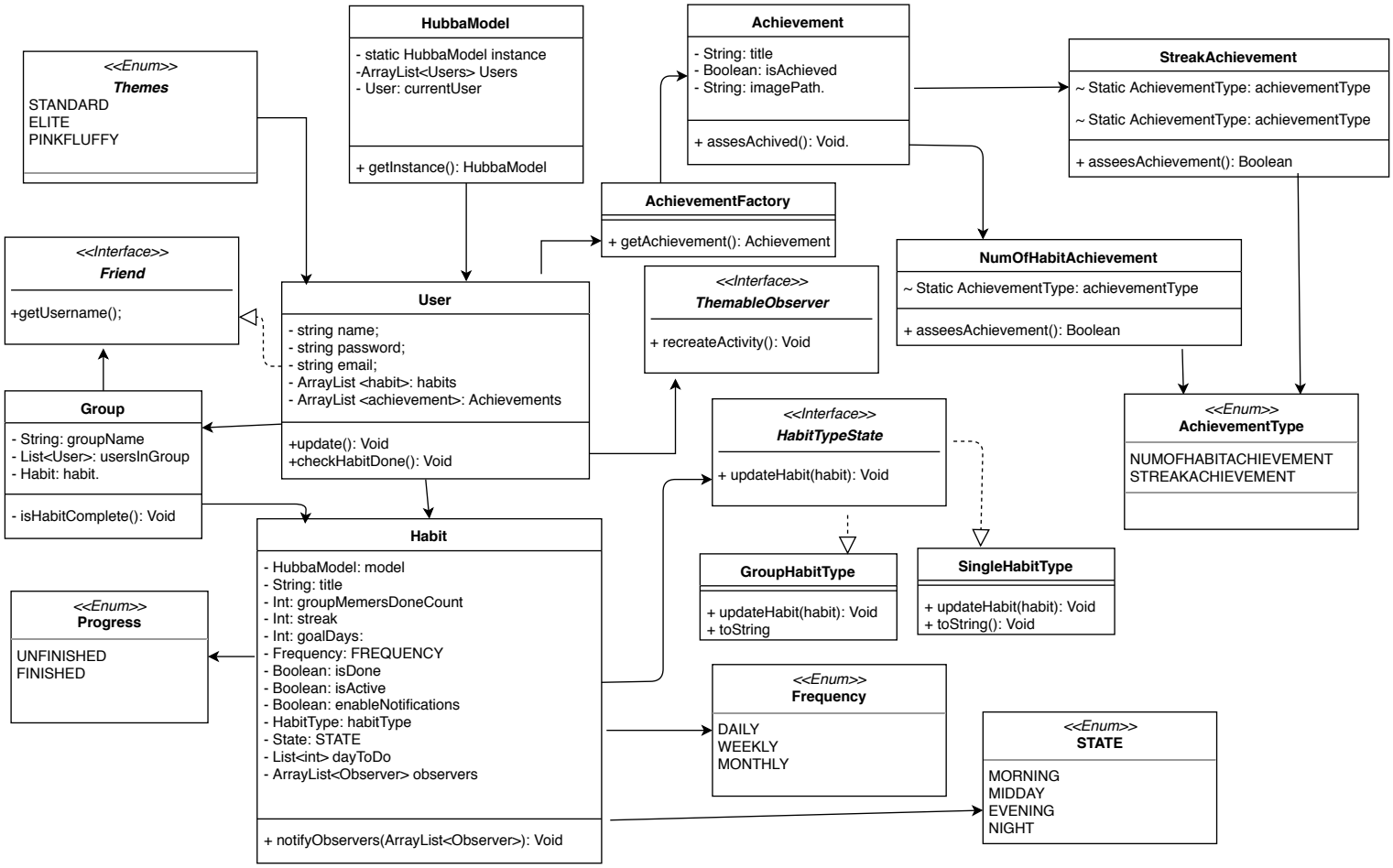

Figure 10: Design model for Hubba

4.6 Subsystem decomposition

State pattern lets the object change between different states, changing the functionality of the object with the state. [2] In the application the pattern lets habit change between (currently) two states, which could be easily extended to several more.

Iterator pattern provides a way to access objects in for example a list, without exposing the underlying pattern [3]. In this case the built in Iterator is used in MainActivityVM to iterate through lists with strings of the habit titles. This is to not cause any exceptions when the lists are changing while being run through.

Adapter patterns allows clients that otherwise could not work together to do so by converting one of them into something that the other expects. Here it is used to load the habit list items into the main page [4].

The observer pattern is used for one-to-many dependencies between a subject and observers [5]. By having all observers implementing an interface the subject can treat all observers as one type and call the methods existing in the interface on all of them when it have been updated.

This is used when the app switches themes. Since all activities need to restart for the theme to be implemented we make all activities that change theme ThemableObservers. If the theme is changed the method `recreate()` from the interface will be called in all ThemableObservers.

HubbaModel is a singleton class. The singleton pattern is used to ensure that only one instance of a class is created [6]. This pattern is used since HubbaModel is a facade for the rest of the classes in the model. This makes HubbaModel the main way to talk with the model. If it was possible for the HubbaModel to have more than one instance, there would be more than one representation of the application which would be both confusing and very bad.

4.6.1 Stan

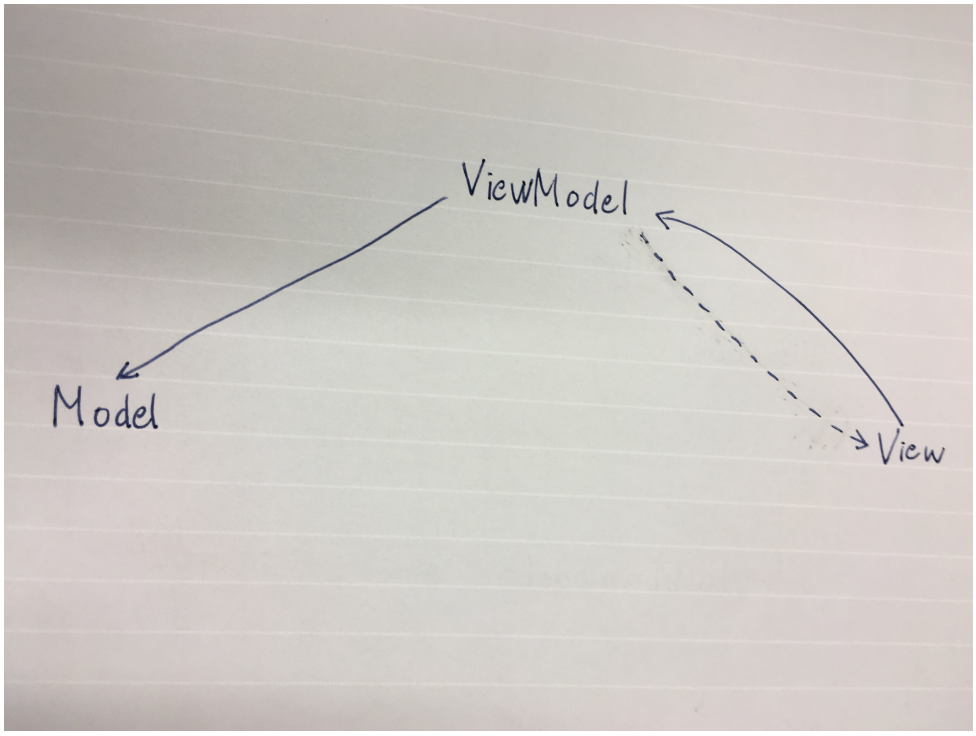

Figure 11: A diagram over the package structure

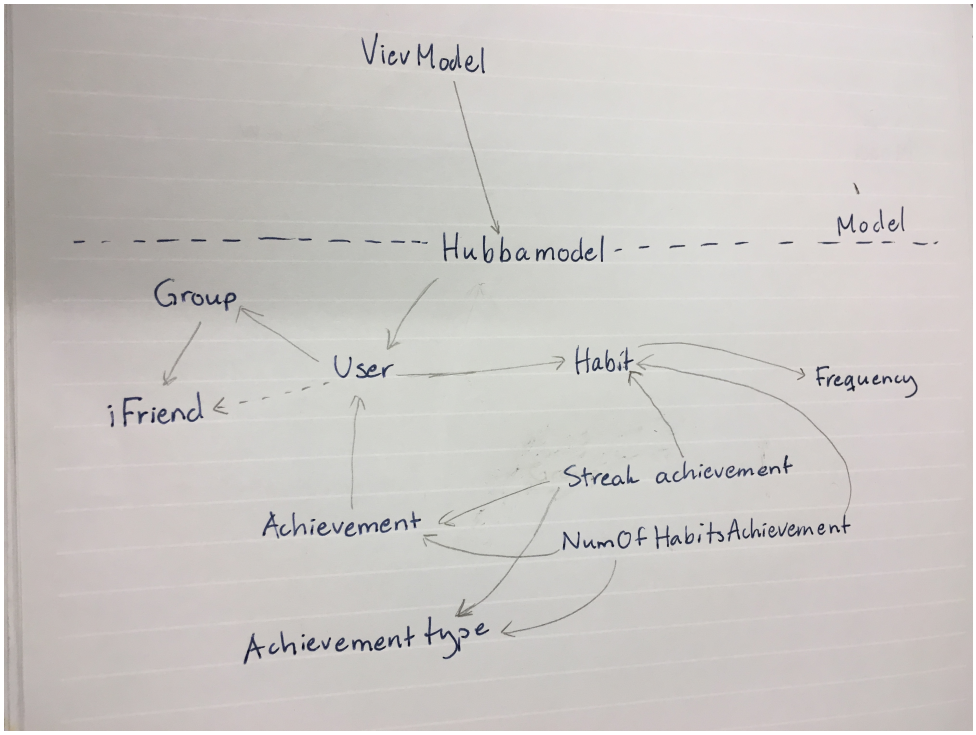

Figure 12: A diagram showing the structure of the model

4.6.2 UML sequence diagram

To make sure that no unnecessary dependency was created, a sequence diagram was made. In figure 13 it is illustrated how the commands go through the classes and code when habits are shown when the user chooses a specific date in the calendar view.

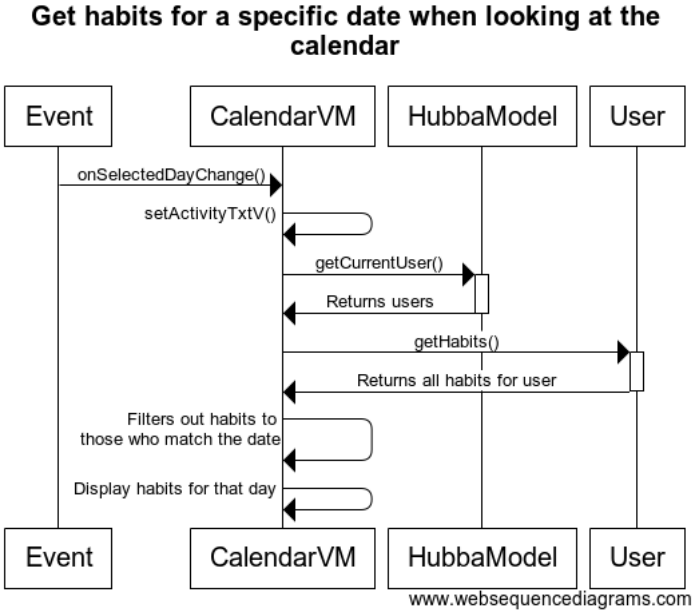

Figure 13: Diagram over the flow of display a habit in the calendar

4.6.3 Test

`AchievementTest` - This testClass contains the tests `testStreakAchievement`, `testStreakAchievementFalse`, `testNumOfHabitsAchievementFalse` and `testNumOfHabitsAchievement`.

`GroupTest` - This testClass contain the test `testCreateGroup`

`HabitTest` - This testClass contains the tests `testCreateHabit`, `testSetDone`, `testSetHabitTypeStateGroup` and `testSetHabitTypeStateSingle`.

`HubbaModelTest` - This testClass contains the tests `testGetInstance`, `testGetUser`, `testGetCurrentUser`, `testGetUsers` and `testThemeChange`.

`UserTest` - This testClass contains the tests `testCreateUser`, `testAddHabit`, `testRemoveHabit`, `testAddFriend` and `testRemoveFriend`.

5 Persistent data management

The application stores data with the help of `SharedPreferences` and the editor in `SharedPreferences` that stores the data in the form of strings into a file with he help of `JSONObject` and `JSONArray`. The data is loaded with the help of `SharedPreferences` and in most cases google's Gson library, `JSONObject` and `JSONArray`.

5.1 Storing data

An instance of `SharedPreferences` is created from which an editor is also opened. The `SharedPreferences.Editor` is an interface which allows modification of the `SharedPreferences` object. After that the data that needs to be saved is transformed to `String` to be able to be saved. This is done with the help of `JSONObject` and `JSONArray` that convert data into `String`. The save functionality is divided into multiple parts that all convert the data from the object to `String`. That `String` is then used by `SharedPreferences` and stored in a file with a key on the local device with the command `apply`.

5.2 Loading data

`SharedPreferences` is once again created and the data is loaded with the help of the key created earlier to store the data in a `String`. For the most types of data a `Gson` object is created as well as a `JSONObject` to handle the stored data and assign it to the required places. Some types of data cannot be cast so a new `Type` and `TypeToken` is created to handle the data. In some other cases the data cannot be loaded with `Gson` or `JSONObject` and must therefore be loaded from the`String` and then assigned to the right place with the help of `StringBuilder`.

5.3 Reasoning behind the functionality of the save and load functions

The way the application's data is stored and loaded is not extendable, but was still implemented because it was the only way that storing and loading data seemed to work after many attempts, and yet some data was not able to be loaded from the stored state, for example is it not possible at the moment to load the habit that a group has. This is a known problem and will be updated and implemented in the future.

6 Access control and security

In this application there aren't different roles such as admin etc. Everyone that uses the application are users and they have no impact on each other except as friends.

The user can be treated as a friend from another users perspective. The other user can only see his friends name, common habits and nothing more. The friends is used so the user can make group with other users in which they can do habits together.

7 Peer Review

Does the design and implementation follow design principles?
Does the project use a consistent coding style?
Yes, small classes, small methods for the most part, with exceptions for some bigger classes with methods that could be broken down into smaller parts. For example a lot of if-statements in a row in RegisterPresenter

Is the code reusable?
Some methods would be more reusable if they were decomposed into smaller methods. There are methods returning interfaces which lowers dependencies.

Is it easy to maintain?
Would be easier if some methods used more functional decomposition. But for the most part yes.

Can we easily add/remove functionality?
Uses interfaces and factories for abstraction and extendability. The code can be adjusted to use a real database instead of the internally stored files that are used now. Both this, and the fact that the user password is not encrypted is reasoned for in the SDD (not really necessary for this project).

Are design patterns used?
Singleton pattern is used in the project as described in the project SDD. MVP is used, for more information see "Is the code easy to understand? Does it have an MVC structure?" Iterator and Adapter pattern was found when the project was read on first glance. An example of this can be seen in the class EventCollection where an iterator is used to iterate over the different collection of events.

Is the code documented?
The amount of comments varies between different parts of the code. Some classes and methods are well documented while others are still missing their documentation. This will probably be added in the future.

Some of the comments can be seen as unnecessary when it comes to commenting getters and setters since these in general have a obvious purpose. This makes the code less readable on first glance.

Are proper names used?
Proper name are mostly used in the project, which means that when reading the name of methods/classes/interfaces it is possible to understand what that part of the project is supposed to do without looking at the code.

The definition of "event", "task" and "deadline" is confusing. "Task" and "deadline" is defined as the same as "event" when in fact "task" should be something to be done and "deadline"

is when to do it. This is written in both the SDD and as a documentation in the class Event.

In some of the classes, for example MonthCalendarPresenter, the local variables in some methods have very inexplicit names so that it is more difficult to understand what is happening.

Is the design modular?
The design is partially modular, it is packaged in the larger groups: (model, presenters, services and views). It might be even more modular if the classes would be grouped by the class's area of use, for instance could everything connected to 'Event' in the model-package be put in a smaller package to be even more modular.

Does the code use proper abstractions?
The code uses proper abstractions, which can be seen in the code with the implementations of a lot of different interfaces.

Is the code well tested?
The code has some tests connected to it, but since we cannot build the project it is impossible to see if they pass or not. A lot of classes and bigger methods are missing tests.

Are there any security problems, are there any performance issues?
There is some security for example ZxcvBn. They don't explain what Zxcvbn is or does. When first viewed it is very confusing and required Google to understand.

Is the code easy to understand? Does it have an MVC structure?
They have used MVP pattern which seems to be followed. It could be further explained in the SDD so that people with little to no knowledge about the pattern could understand what it is about.

Can the design or code be improved? Are there better solutions?
Yes, overall it's very good, but some things could be implemented for clarification and easier overview. For example more functional decomposition in larger methods in combination with some documentation of the external libraries used in the project.

References

[1] Microsoft docs,"The Model-View-View model Pattern". [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm, accessed: 2018-10-28.

[2] Source making, "State Design Pattern". [Online]. Available: https://sourcemaking.com/design_patterns/state, accessed: 2018-10-28.

[3] Source making, "Iterator Design Pattern". [Online]. Available: https://sourcemaking.com/design_patterns/iterator, Accessed: 2018-10-26.

[4] Source making, "Adapter Design Pattern". [Online]. Available: https://sourcemaking.com/design_patterns/adapter, Accessed: 2018-10-26.

[5] Source making, "Observer Design Pattern". [Online]. Available: https://sourcemaking.com/design_patterns/observer, Accessed: 2018-10-26.

[6] Source making, "Singleton Design Pattern". [Online]. Available: https://sourcemaking.com/design_patterns/singleton, Accessed: 2018-10-21.