System Design Document for Habba

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

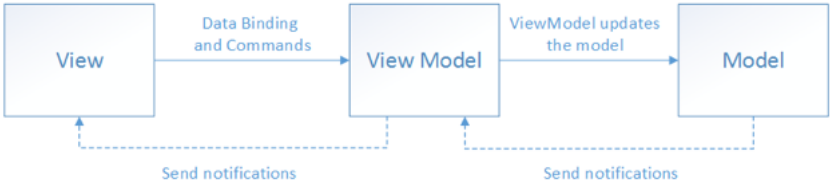
This document describes the design choices made when constructing the android application Hubba.

1.1 Definitions, acronyms, and abbreviations

- **Habit**: A habit is an action that the user wants repeat and keep track of. For example a habit could be washing their face every morning or water the plants once every week.
- **User**: A user is a person that has an account in the app and uses the application.
- **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.
- **Achievement**: When users have a number of habits or have performed their habits a number of times they unlock achievements.
- **Streak**: When a user has done a Habit five days in row, the user will receive a streak. The streak is present until the day the user forgets to do the habit.

2 System architecture

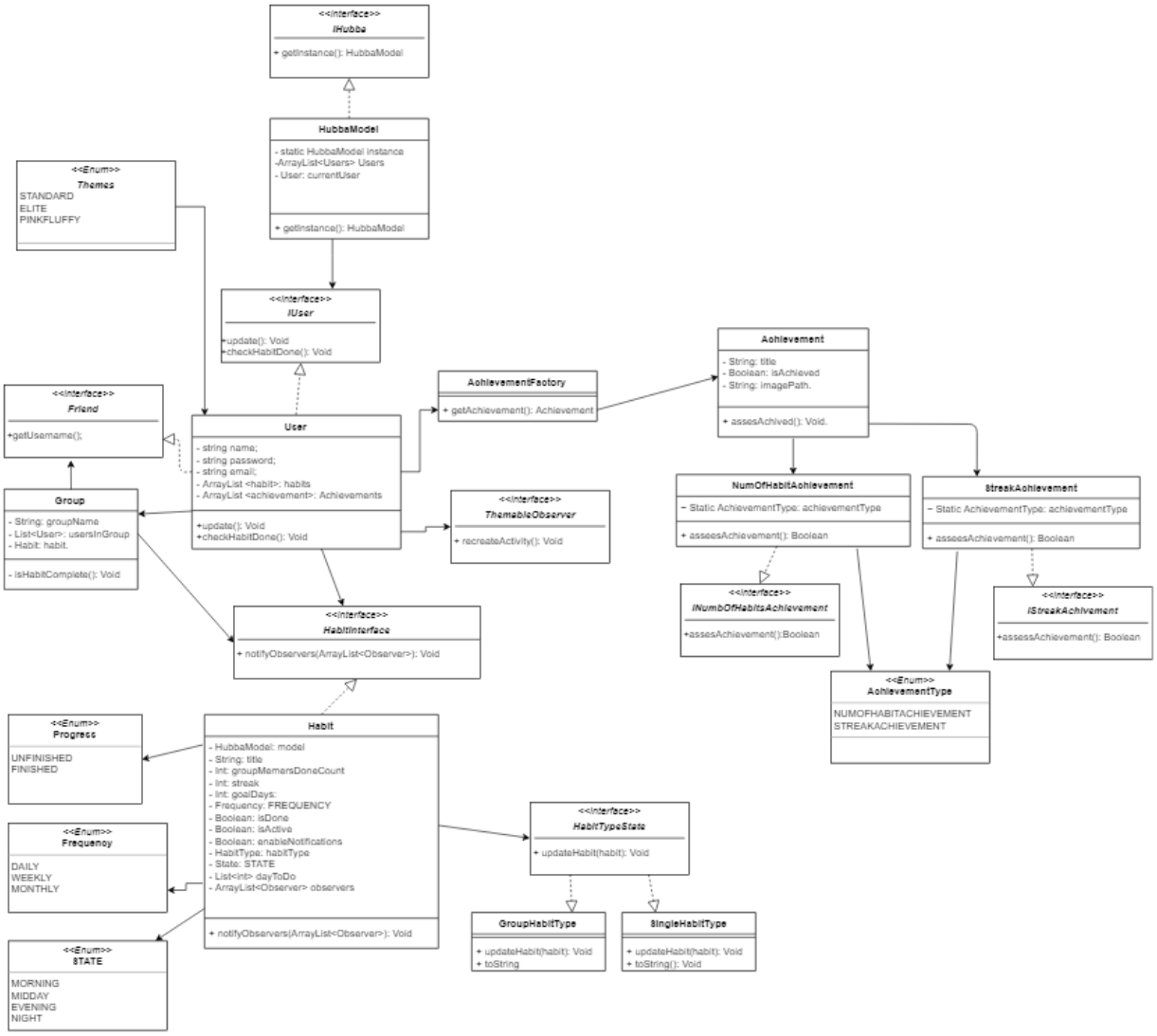
The system is constructed according to the Model-View-ViewModel (MVVM) pattern, an overview of the pattern can be seen in the figure below.



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.

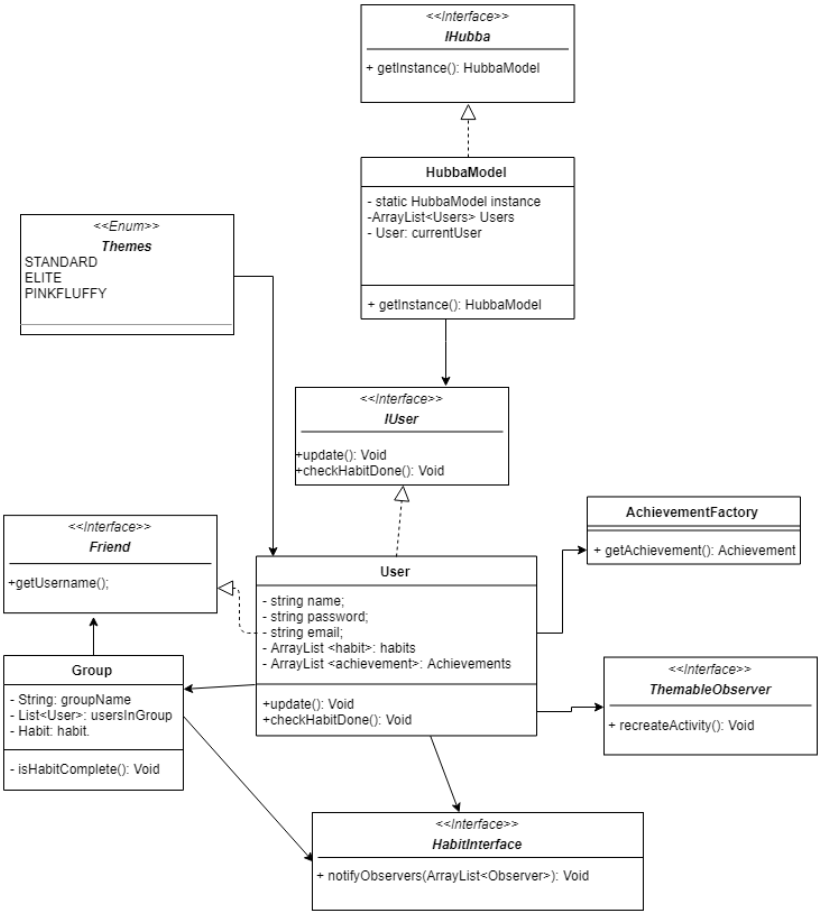
- The model represents the data, state, and business logic. It is completely independent of View or View Model
- The view binds to actions and variables in View Model. It can also be a package consisting of the XML-files [2].
- The view model binds the other two modules together. It provides a bridge for the view to pass events to the model, and wraps the model and the data to fit the view. The view model is not tied to the view.

2.0.1 Design models

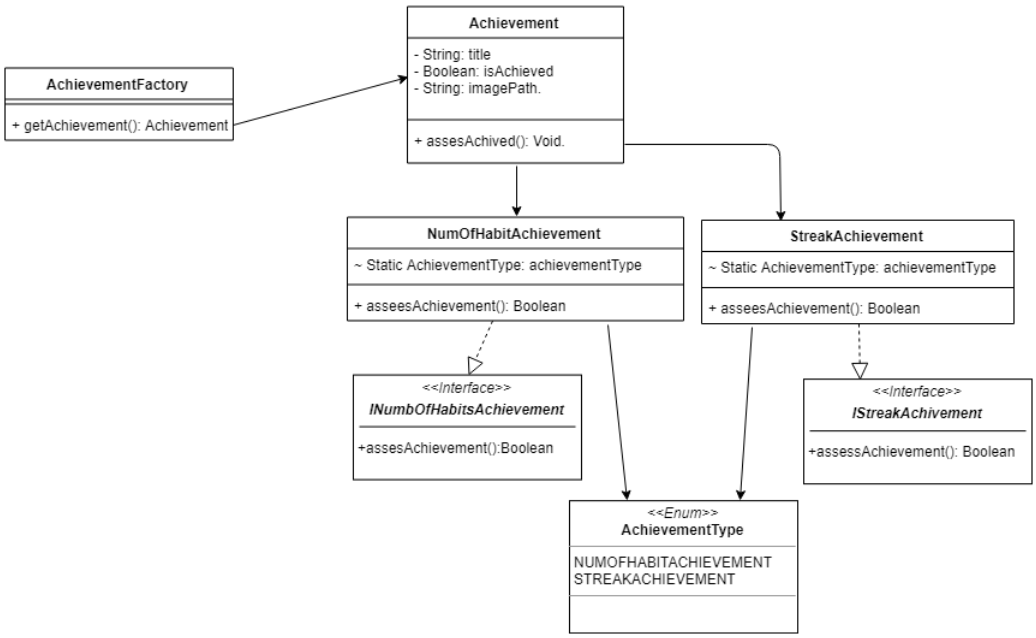


Figur 1: Design model for Hubba

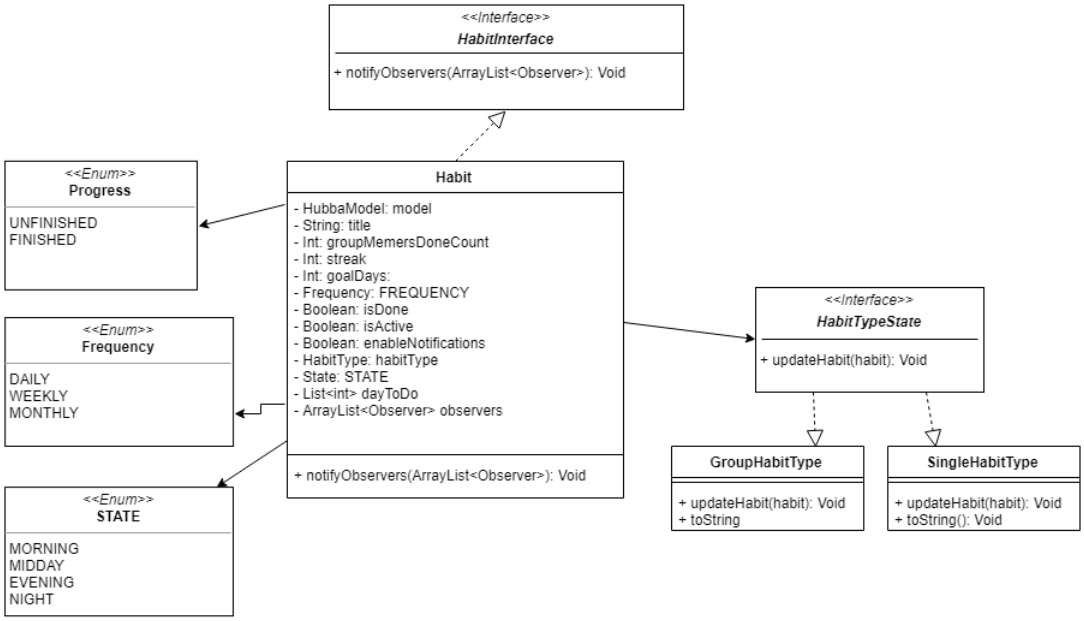
.



Figur 2: Closeup of design model 1



Figur 3: Closeup of design model 2



Figur 4: Closeup of design model 3

2.1 Subsystem decomposition

Iterator pattern provides a way to access objects in for example a list, without exposing the underlying pattern. 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.

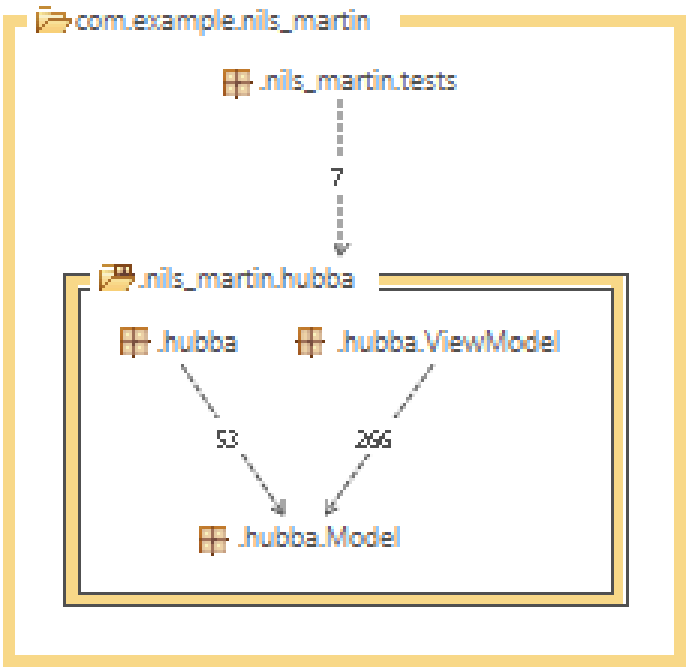
Adapter patterns allows clients that otherwise could not work together to do so by converting one of them into something that the other expects. Here it is used to load the habit list items into the main page.

HubbaModel is a singleton class. The singleton class is used to ensure that only one instance of a class is created [1]. This is used full because Hubbamodel only is a facade for the other model classes and it is the main way to talk with the model. If it was possible for the HubbaModel to have more than one instance.

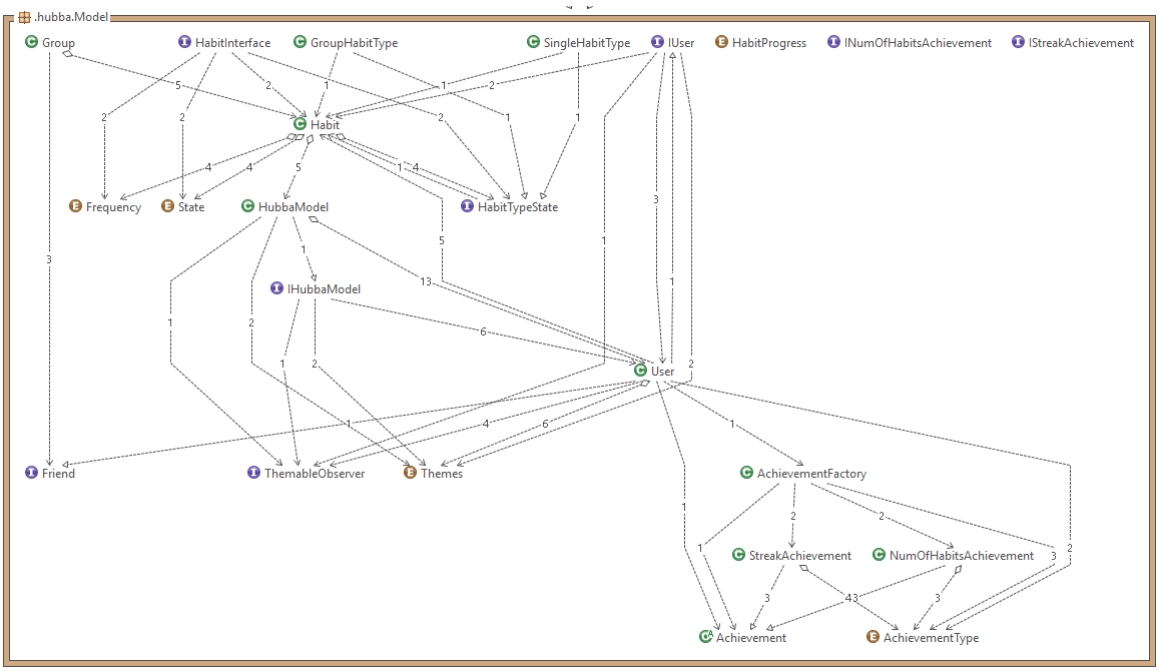
2.2 Model

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.

2.2.1 Stan

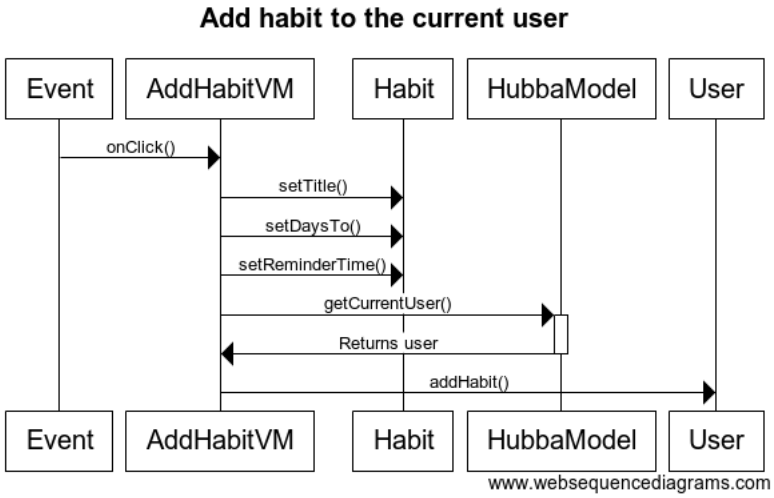


Figur 5: A Stan diagram over the package structure

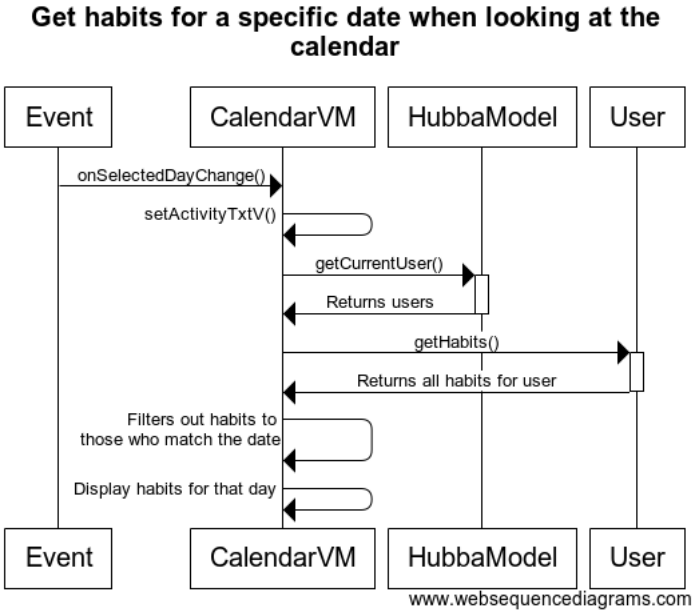


Figur 6: A Stan diagram showing the structure of the model

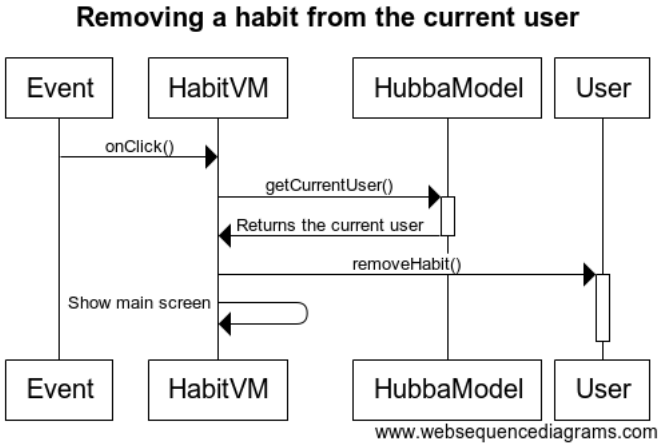
2.2.2 UML sequence diagram



Figur 7: Diagram over the flow of create a habit



Figur 8: Diagram over the flow of display a habit in the calendar



Figur 9: Diagram over the flow of remove a habit

2.2.3 Test

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

`ExampleUnitTest` - This testClass contains the tests `testCreateHabit, testCreateUser, testThemeChange` and `testUpStreak`.

`HabitUnitTest` - This testClass contains the tests `testCreateHabit, testSetDOne, testSetHabitTypeStateGroup` and `testSetHabitTypeStateSingle`.

`HabitViewTest` - This testClass contain the test `testString`.

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

2.3 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 uodate 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.

2.4 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 handels that information.

3 Persistent data management

The application stores and loads data with the help of google's Gson library, Json and android's SharedPreferences.

3.1 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. Second, a gson object is created which is used to put the userlist into a String in Json-format. The editor then matches the string with a key and sets in in the editor which stores it back to `SharedPreferences` as soon as `apply()` och `commit()` is called.

3.2 Loading data

`SharedPreferences` is once again created and a new instance of a Gson-object is created. The json-string is then assigned to a String variable which cojoined with a Type-variable sets the userlist to the saved value. An if statement then makes sure the userlist is not null.

4 Access control and security

In this application there are no different role such as admin etc. Everyone that uses the application are users and have no impact on each other expect 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.

Referenser

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

[2]