System Design Document for

TDA367

Group 11

Pauline Björk, Hanna Harnesk,
Kristin Hulling, Malin Kihlström

Department of Infromation Technology
Chalmers University of Technology
Gothenburg, Sweden 2022

Innehåll

1 Introduction

In this document you will read about the system design for the project TidyApp developed during the course TDA367. The system design of TidyApp contains the applications architecture, how its components are designed and connected to each other as well as how they are tested.

This is an application developed to help multi- person households to be more engaged in doing chores by making a competition of it where the main purpose is gaining points. There are functionalities like adding and customizing new chores when needed and creating multiple profiles if new people are added to the household.

1.1 Definitions, acronyms, and abbreviations

- MVVM: Model-View-Viewmodel
- MVC : Model-View-Controller

2 System architecture

At start, the application will first start the Auth Activity which has the fragments that uses Firebase authenticator to sign in and register households. Once a household is signed in the user id will be written on a text file. Its purpose is to handle who is currently signed in household. Now the application will get the currently signed in household from the Firebase Realtime database and list all the profiles in a recycle view. Upon clicking on a profile, the current profile will be set in the GetCurrentProfile class.

When the user of the application has chosen a profile the Main Activity will be showed. The Main Activity contains household chores that are fetched from the database. The profile can add chores to the household, which everyone in the household can see. Chores can also be checked as completed and that gives points to the profile that completed the chore, and the chore in question is moved to the profile's list of completed chores. The database is also updated when a chore is completed and a profile's points is increased.

The activity's xml file contains a Navigation host that can switch between different fragments. The navigation controller is set in the activities with the navigation host. There is also a xml file for the navigation which contains actions between the fragments which can then set a onClickListener on something in the UI and use the Navigation controller to navigate with the actions that is created between fragments. There is also a navigation menu on the Main Activity which is configured with a appBarConfiguration.

3 System design

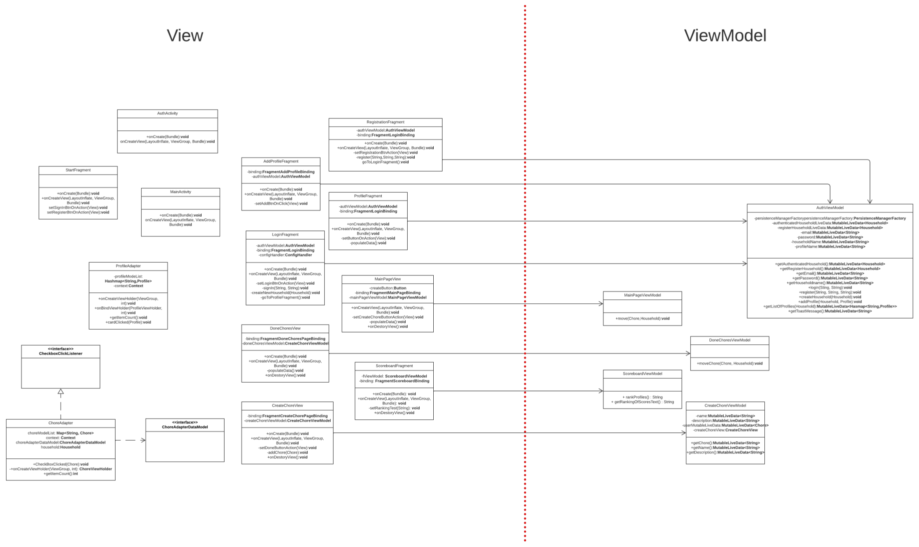
This section will be focusing on the structure of the application, how it is divided into packages, the design model as well as design patterns and the dynamic design.

Package Diagram

The code is divided in multiple different packages. Three of these are created in a MVVM structure, which is another variant of the MVC pattern. In this modification there is a Viewmodel instead of a Controller but the model is still independent of the View and the Viewmodel. The Viewmodel simply acts as a bridge of communication between the View and the Model. In our implementation there are four strong associations between and multiple looser dependencies in between the packages.

The Viewmodel has a dependency to the Model. More specifically, the AuthViewmodel is dependent on the PersistanceManagerFactory within the model, from which it gathers all information required to present the information from the model and its database via the view.
There is also more loose dependencies from the Viewmodel as DoneChoresViewModel and MainPageViewModel implements an interface from the view and uses the Household and Chore classes from the model. This is not ideal, and if time was not as limited another solution could be to create a facade pattern around the Model which would create one single outlet instead of having direct contact between objects in different packages.

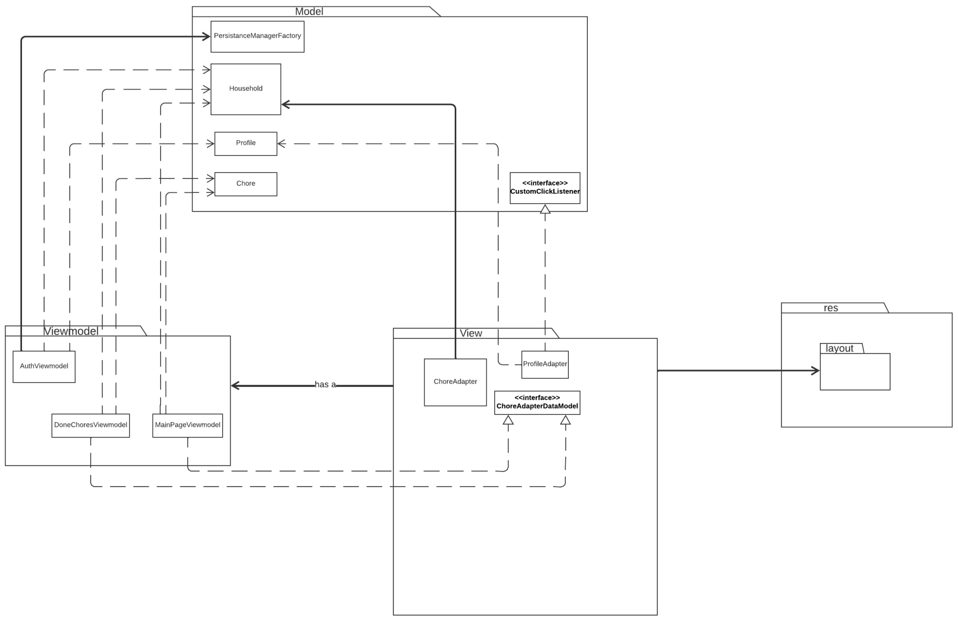
The View is strongly dependent on the Viewmodel as every fragment in the View associates to on one Viewmodel in order for the Viewmodel to transmit information from the view to the model and vice versa. The view is also strongly dependent on the Model, more specifically the ChoreAdapter has an instance of the Household. There is also a dependency between the view and the layout package within the res package. Every fragment has an instance of their own FragmentBinding which is the connection between the values input or output from the components of the UI and the View.



Figur 1: UML class

Since there is no strong dependency from any package to the View, its the interface is more narrow than the other packages' as it is limited to the Fragment interface, the CheckboxClickListener interface and the CustomCLickListener interface.

The Model is completely independent of the View and the Viewmodel which is in accordance with the MVC pattern. All components of the Model is testable and one could in practice run the application without the use of the graphical interface. However, as mentioned earlier, there are better more nice solutions that reduces the dependencies from other packages to the model.



Figur 2: Package diagram

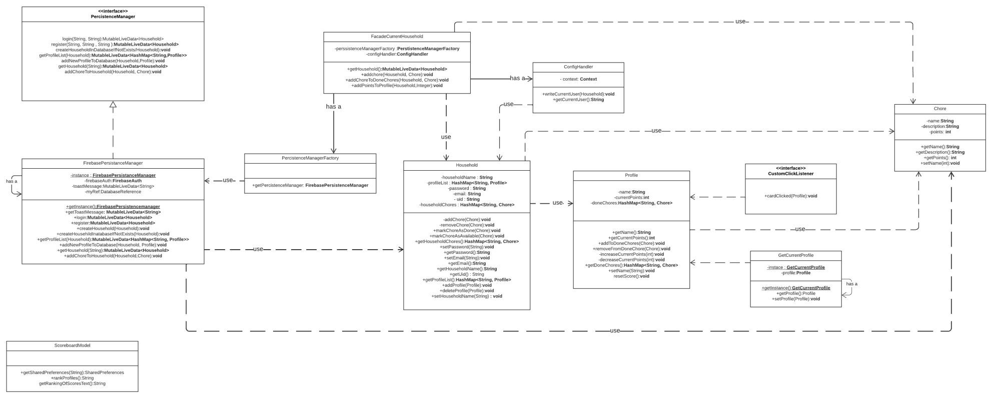
Design model and domain model

Within the design model there are four main objects; Household, Profile, Chore and Scoreboard, much like the objects in the domain model. There is also an application in the domain model which is separate from the design model. These components have the same relation to each other in the domain model

However, there are fewer components in the design model than what are presented in the domain model. Some objects, such as individual score, available/done chore, were not independent enough to be implemented as separate objects. Instead, these components were implemented within other objects.

In the design model, the individual score is an attribute on the profile object. The different states of the chore, there are instead different places where the chore resides depending on whether the chore is completed by a profile or not. If the chore is ävailable"it resides in a list in the Household object, and when it is completed it is moved to a list within the profile who completed the chore.

The scoreboard belongs to a Household in the domain model. However, because of lack of time the Scoreboard is "hanging loose"in the design model. It was intended for the Scoreboard to have the same relation in the design model as it has in the domain model.


<<interface>>
PercistenceManager
login(String, String):MutableLiveData<Household>
register(String, String , String):MutableLiveData<Household>
createHouseholdInDatabaseIfNotExists/Household):void
getProfileList(Household):MutableLiveData<HashMap<String,Profile>>
addNewProfileToDatabase(Household,Profile):void
getHousehold(String):MutableLiveData<Household>
addChoreToHousehold(Household, Chore):void
FacadeCurrentHousehold
-perssistenceManagerFactory :PerstistenceManagerFactory
-configHandler:ConfigHandler
+getHousehold():MutableLiveData<Household>
+addchore(Household, Chore):void
+addChoreToDoneChores(Household, Chore):void
+addPointsToProfile(Household,Integer):void
has a
ConfigHandler
- context: Context
+writeCurrentUser(Household):void
+getCurrentUser():String
use
has a
use
use
FirebasePersistanceManager
-instance : FirebasePersistanceManager
-firebaseAuth:FirebaseAuth
-toastMessage:MutableLiveData<String>
-myRef:DatabaseReference
+getInstance():FirebasePersistencemanager
+getToastMessage: MutableLiveData<String>
+login:MutableLiveData<Household>
+register:MutableLiveData<Household>
+createHousehold(Household):void
+createHousehildInatabaseIfNotExists(Household):void
+getProfileList(Household):MutableLiveData<HashMap<String, Profile>>
+addNewProfileToDatabase(Household, Profile):void
+getHousehold(String):MutableLiveData<Household>
+addChoreToHousehold(Household,Chore):void
has a
use
PercistenceManagerFactory
+getPercistenceManager: FirebasePersistenceManager
Household
-householdName : String
-profileList : HashMap<String, Profile>
-password : String
-email: String
- uid : String
-householdChores : HashMap<String, Chore>
-addChore(Chore):void
-removeChore(Chore):void
+markChoreAsDone(Chore):void
+markChoreAsAvailable(Chore):void
+getHouseholdChores():HashMap<String, Chore>
+setPassword(String):void
+getPassword():String
+setEmail(String):void
+getEmail():String
+getHouseholdName():String
+getUid() : String
+getProfileList():HashMap<String, Profile>
+addProfile(Profile):void
+deleteProfile(Profile):void
+setHouseholdName(String) : void
use
use
Profile
-name:String
-currentPoints:int
-doneChores:HashMap<String, Chore>
+getName():String
+getCurrentPoints():int
+addToDoneChores(Chore):void
+removeFromDoneChore(Chore):void
-increaseCurrentPoints(int):void
-decreaseCurrentPoints(int):void
+getDoneChores():HashMap<String, Chore>
+setName(String):void
resetScore():void
<<interface>>
CustomClickListener
+cardClicked(Profile):void
use
GetCurrentProfile
-instace : GetCurrentProfile
-profile:Profile
+getInstance():GetCurrentProfile
+getProfile():Profile
+setProfile(Profile):void
has a
Chore
-name:String
-description:String
-points: int
+getName():String
+getDescription():String
+getPoints(): int
+setName(int):void
use
use
ScoreboardModel
+getSharedPreferences(String):SharedPreferences
+rankProfiles():String
getRankingOfScoresText():String

Figur 3: MODEL class diagram

Patterns

There are multiple usages of different design patterns within the application. The singleton pattern is used on the class FirebasePersistenceManager. The class is the only one who handles Firebase. The singleton pattern is implemented because the need of sharing one single object between different parts of the program. The Singleton pattern is also implemented in the class GetCurrentProfile.

The program makes use of the factory pattern in PercistanceManagerFactory in order to be able to exchange the current database Firebase, for any given database.

The facade pattern is used in the class FacadeCurrentHousehold in order to be able to provide functionality for the necessary calls to and from the database, and at the same time limiting the knowledge and exposure of the ConfigHandler, PersitenceManager and the database.

Another type of design pattern meant to be in the code is Chain of Responsibility. The Household was supposed to have a scoreboard to delegate operations sent from the Viewmodel regarding the ranking of profiles and scores to the scoreboard. However, due to lack of time, this has not yet been implemented.

Dynamic design

There are a few functionalities that have been put into dynamic diagram. Mainly logging in and signing up, creating a chore and completing a chore.

4 Persistent data management

The users information is stored in Firebase Realtime database. The database makes it possible to listen to changes and update whenever something happens. The only class that handles the database is FirebasePersistence Manger which is a singelton class, this is made to limit the access to the database. If the choice of database handler in the future will change, a interface with the methods that the firebase manager class has, have been created. A PersistenceManagerFactory has also been created to enable future change of persistenceManager.

5 Quality

The application is tested through unit-test where each method in a class is tested. Unit-test have been done on the classes in the package models. The class Persistence manager is difficult to test due to it's connection to the database Firebase, but the class could be tested trough mocking. Due to the fact that we have not had time to learn about mocking, this class remains untested. FacadeGetCurrentHousehold and PersistenceManagerFactory makes use of PersistenceManager, testing them would also require mocking. These classes therefore also remains untested.

Test can be found in:
app/src/test/java/com/example/dat367_projekt_11, and:
app/src/androidTest/java/com/example/dat367_projekt_11.

All known issues:

- The checkbox-function on a chore in the done chores-list is does not work due to the fact that it has not yet been implemented. In addition, the checkbox on a chore that has been completed does not remain checked when the chore is places in the profiles done chores-list.
- At times, when creating a profile or a new chore, the profile or chore is not created.
- The scoreboard not connected to program
- The amount of point a chore can be assigned is not displayed when creating a new chore.
- If the household is removed from the database, the application will crash if they try to login. Instead of giving an error-message, communication to the user that the user is not found.

5.1 Access control and security

The application uses Firebase authentication service to handle the authentication of users such as login and registration. Firebase only lets the developer see users email and user id for each register user. Firebase authenticator allows the developer to use methods for sign in and registration where the authenticator can see if the password and email is linked to each other.

6 References

https://firebase.google.com/docs
https://developer.android.com/guide