System design document for Boardbook

Aaron Sandgren, Alexander Ohlin, Carl Holmberg, Dino Pasalic, Rasmus Rosengren

2019-10-25
Final Version

1 Introduction

This document will describe the Android application Boardbook by explaining its design and structure. This explanation will include the architecture of the program and the different components that interact within in. Furthermore, it will both show and talk about the design of the application and the design patterns that have been implemented. It will also give a basic rundown on how a user would from start to finish interact with the application. Lastly, the document will detail how the application tests itself and the account security is handled.

Boardbook is an Android application where users can enter results of matches played for supported board games. This allows the application to generate statistics such as overall board game win rate or win rate for specific games. Boardbook will also allow the user to add friends and view their matches as well. Boardbook will be able to show what game was played, roles if there are any and if the person won or lost said match. The user will also be able to get achievements by fulfilling certain criteria like playing ten board games and so on. In order to make Boardbook an encouraging platform for board games, the application will allow users to chat with one another.

1.1 Definitions, acronyms, and abbreviations

MVP: Model-View-Presenter
SDD: System Design Document
Activity: A specific screen in the user interface
Fragment: A subsection of an activity, used for some specific functionality

2 System architecture

Top level design

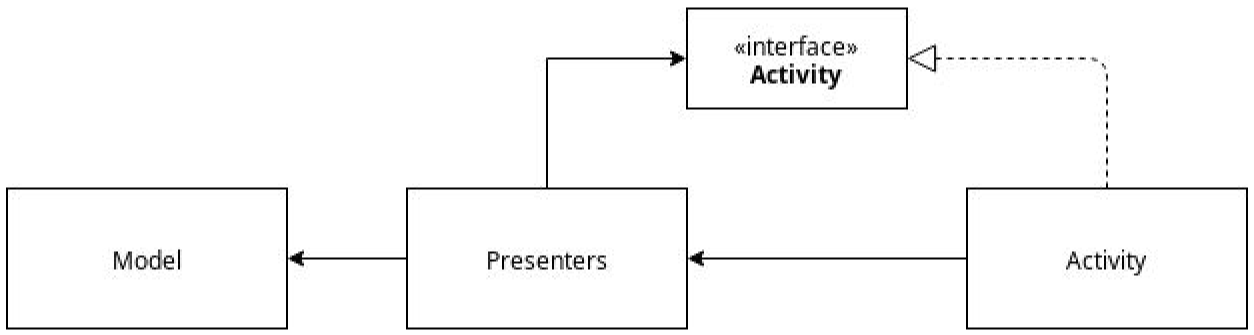
Boardbook only has one component, the application itself. It manages every feature that the application is able to perform.

Flow

When the user first start the application they will be prompted to either log into Boardbook or create a new account. Whichever option they choose will then take the user to the home screen. Here they will be presented with matches that they have previously added to their account or matches other people have added that include their account. It is also here that the user can find the “Add match” button which can be used to add the presumably recently played match of a board game to the users account. The user will then have to select what board game was played, choose the players who played the match, divy the players to potential different teams or role in the boardgame and finally choose which players won or lost. This process exemplifies the most common interaction with the application.

3 System design

Top level:



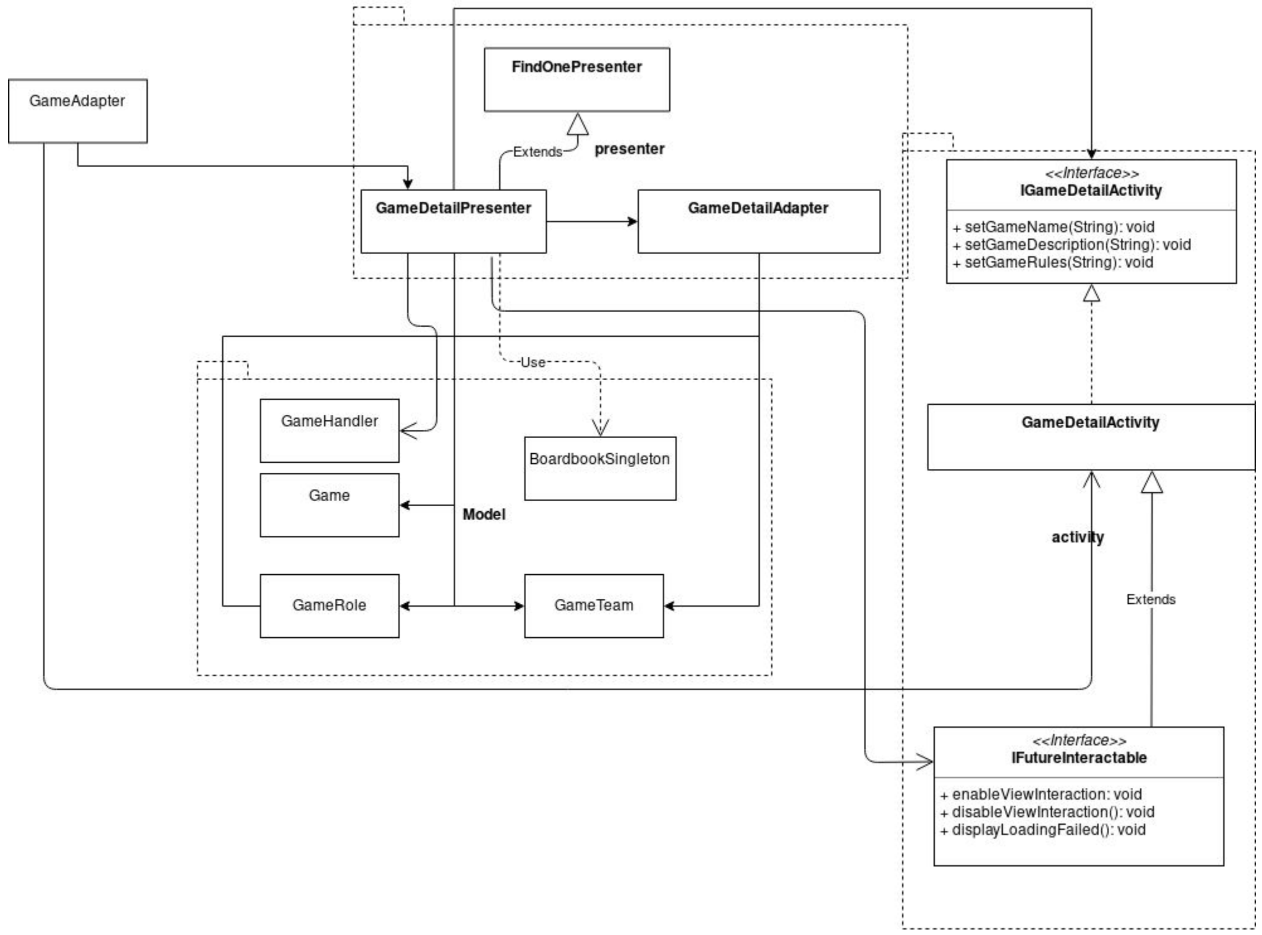
Boardbook has been developed using the MVP design architecture in mind. In this application, the responsibilities goes as following.

Activities and their fragments are dumb views with as little logic as possible. The activities delegate user events to presenters, classes in charge of communication between the model and the view.

Presenters are created by their respective views and respond to their events. They then interact with the model and then tell the activities how to respond.

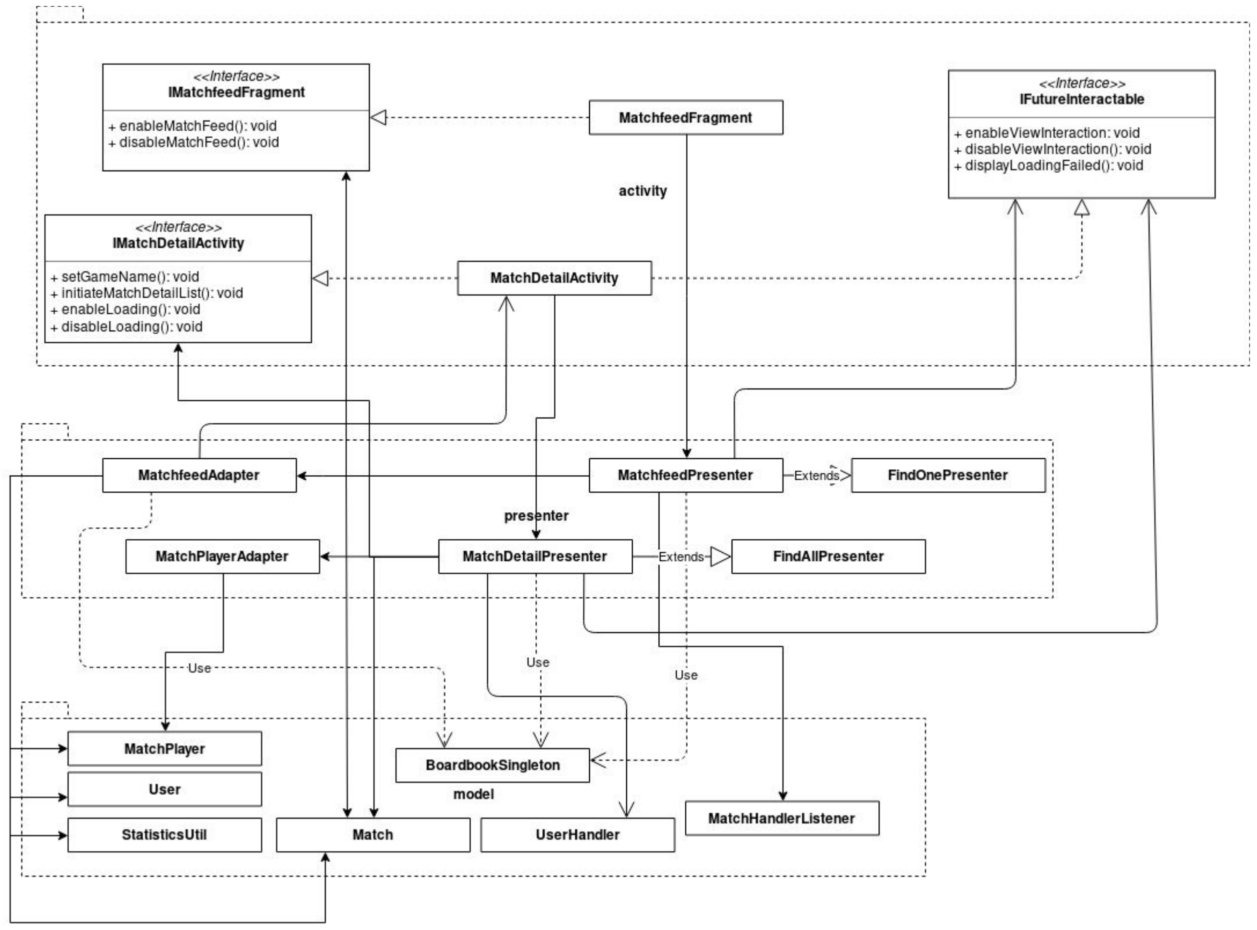
The model is in charge of handling and storing data of games, matches, users and other entities in the application. It does this through the usage of various data handlers whom then in turn interact with repositories. The repositories interact with a database and require a specific implementation, which in our app is done through the usage of Google firebase.

GameDetail:

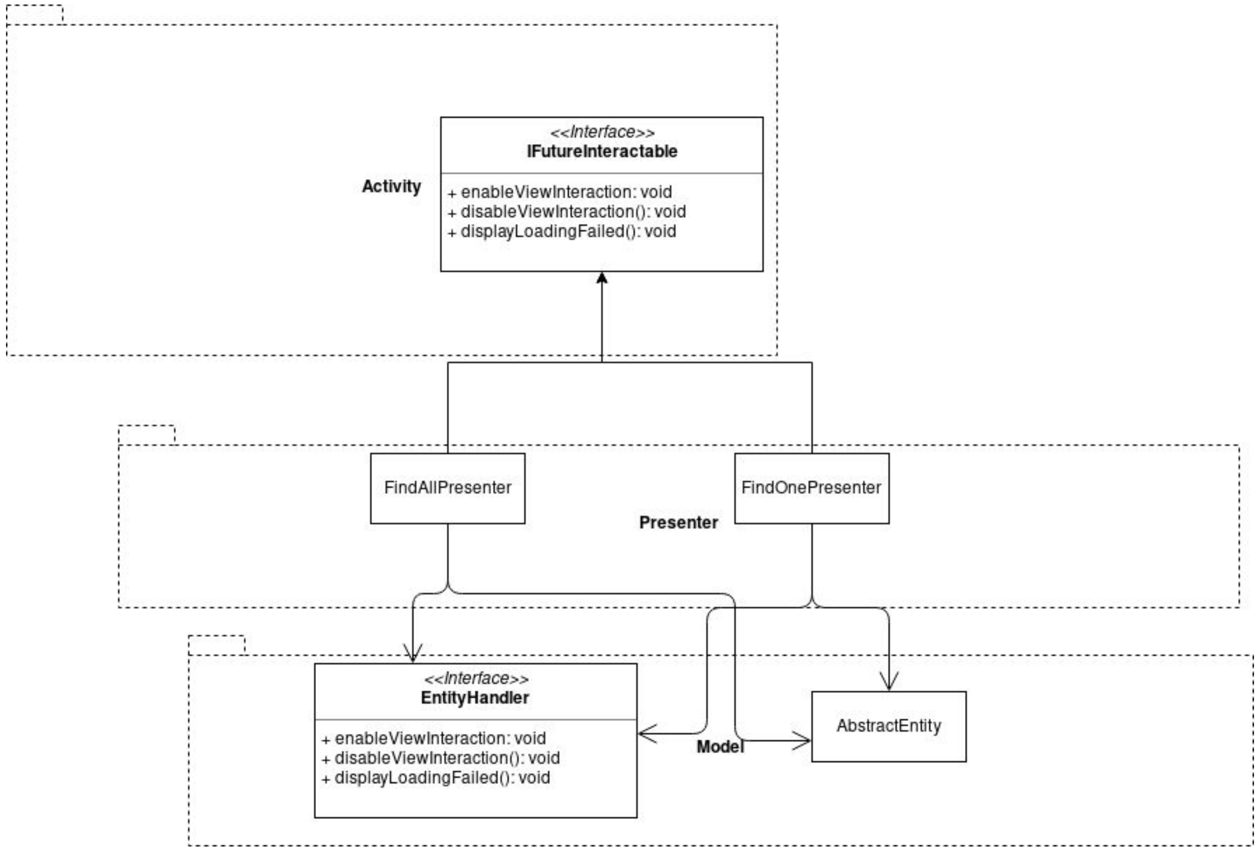


GameAdapter
FindOnePresenter
Extends
presenter
GameDetailPresenter
GameDetailAdapter
<<Interface>>
IGameDetailActivity
+ setGameName(String): void
+ setGameDescription(String): void
+ setGameRules(String): void
Use
GameHandler
BoardbookSingleton
Game
Model
GameRole
GameTeam
GameDetailActivity
activity
Extends
<<Interface>>
IFutureInteractable
+ enableViewInteraction: void
+ disableViewInteraction(): void
+ displayLoadingFailed(): void

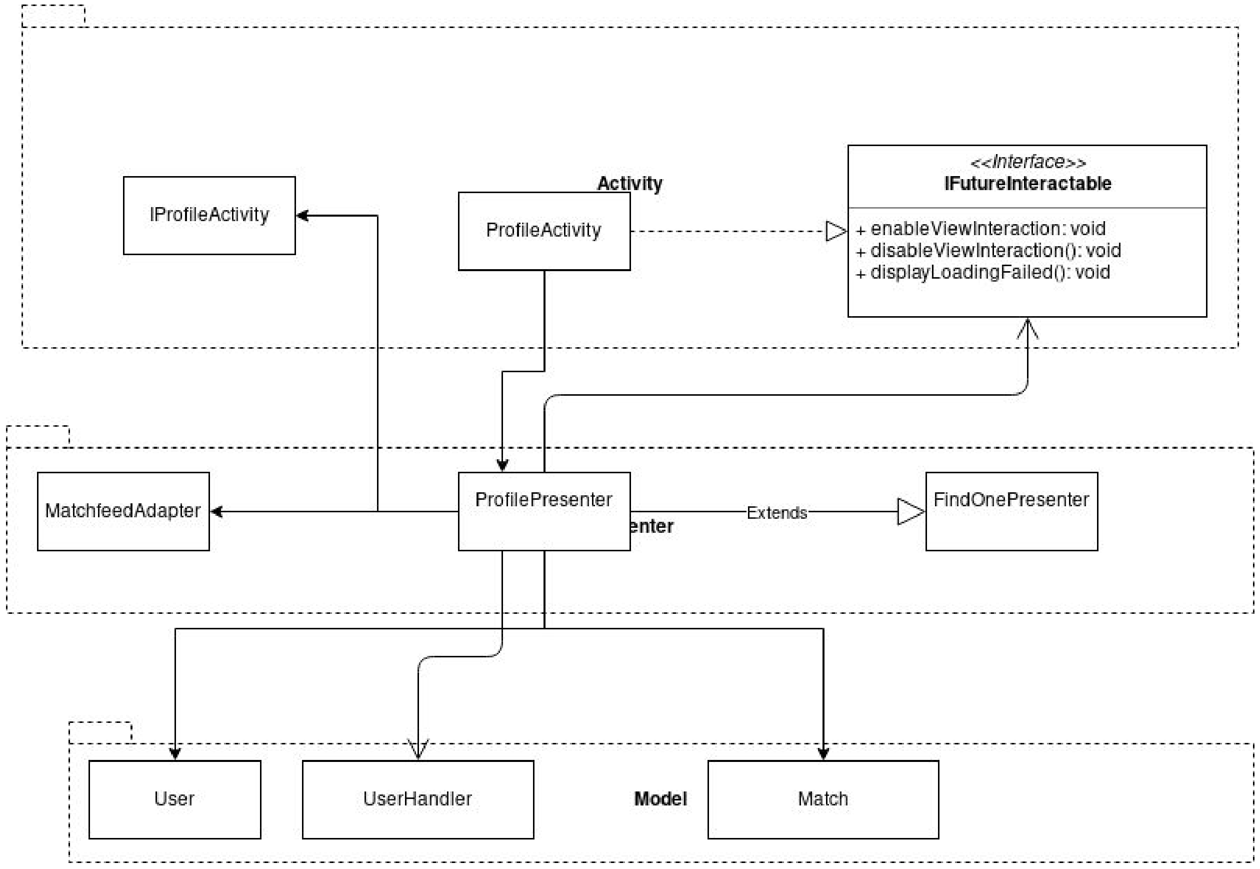
MatchFeed + MatchDetail:



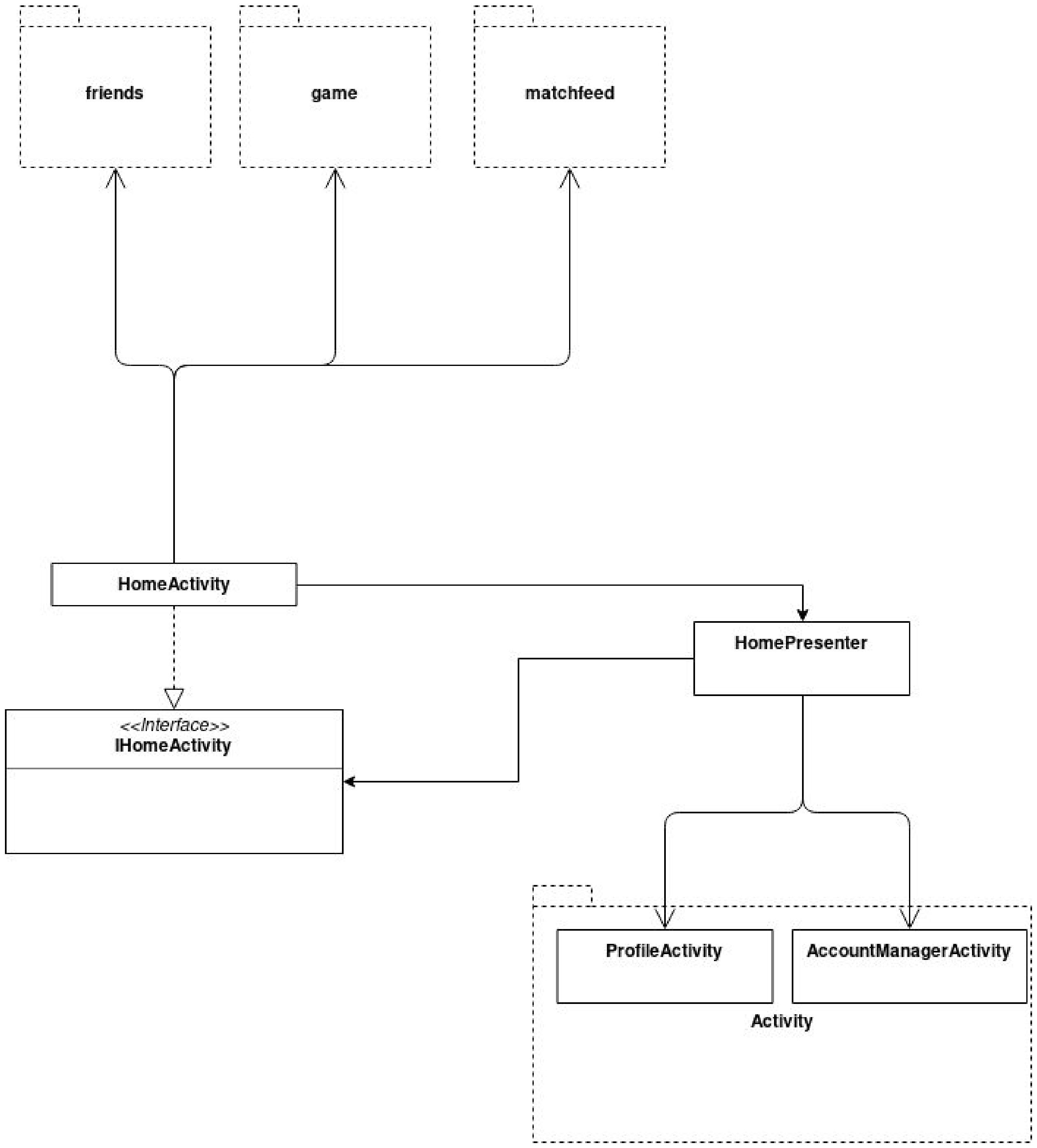
AbstractPresenter:


Activity
<<Interface>>
IFutureInteractable
+ enableViewInteraction: void
+ disableViewInteraction(): void
+ displayLoadingFailed(): void
FindAllPresenter
Presenter
FindOnePresenter
<<Interface>>
EntityHandler
+ enableViewInteraction: void
+ disableViewInteraction(): void
+ displayLoadingFailed(): void
Model
AbstractEntity

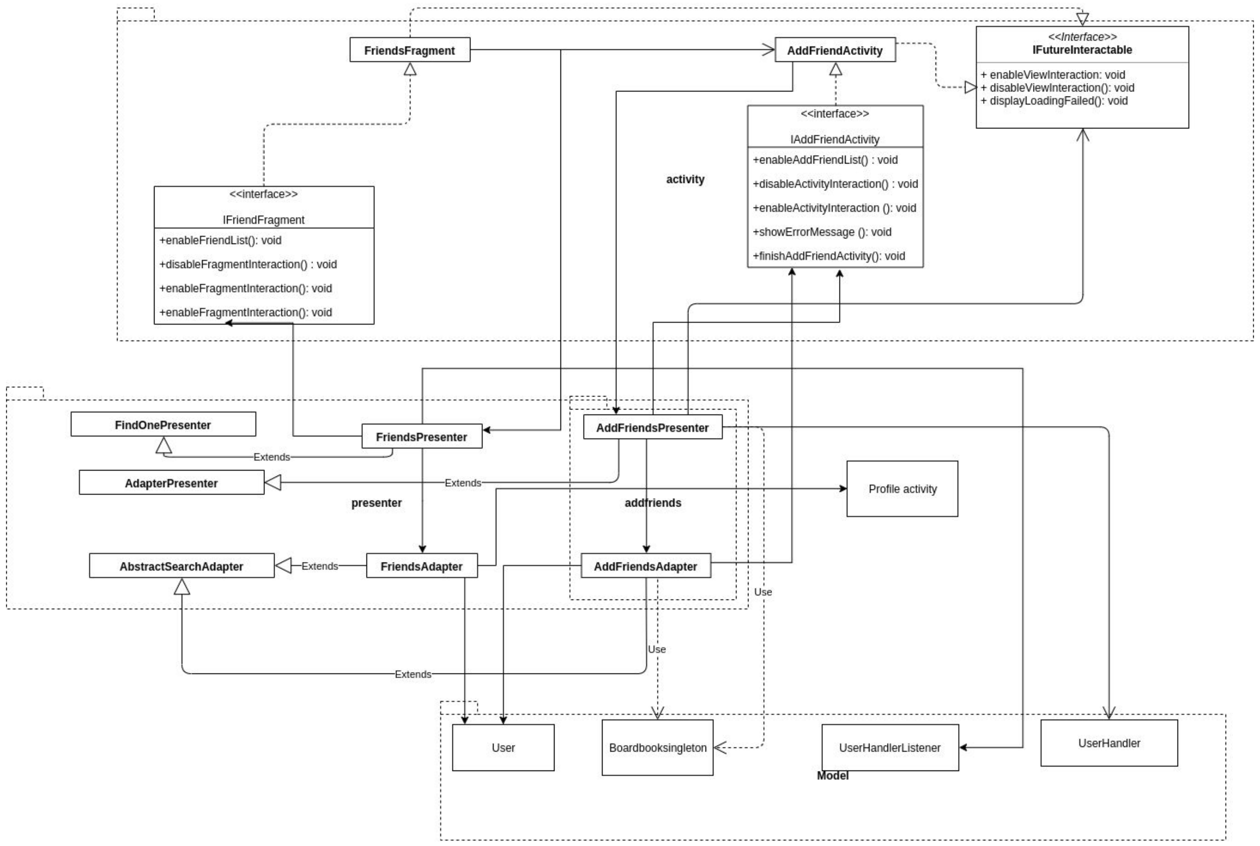
Profile:


Activity
IProfileActivity
ProfileActivity
<<Interface>>
IFutureInteractable
+ enableViewInteraction: void
+ disableViewInteraction(): void
+ displayLoadingFailed(): void
MatchfeedAdapter
ProfilePresenter
enter
Extends
FindOnePresenter
User
UserHandler
Model
Match

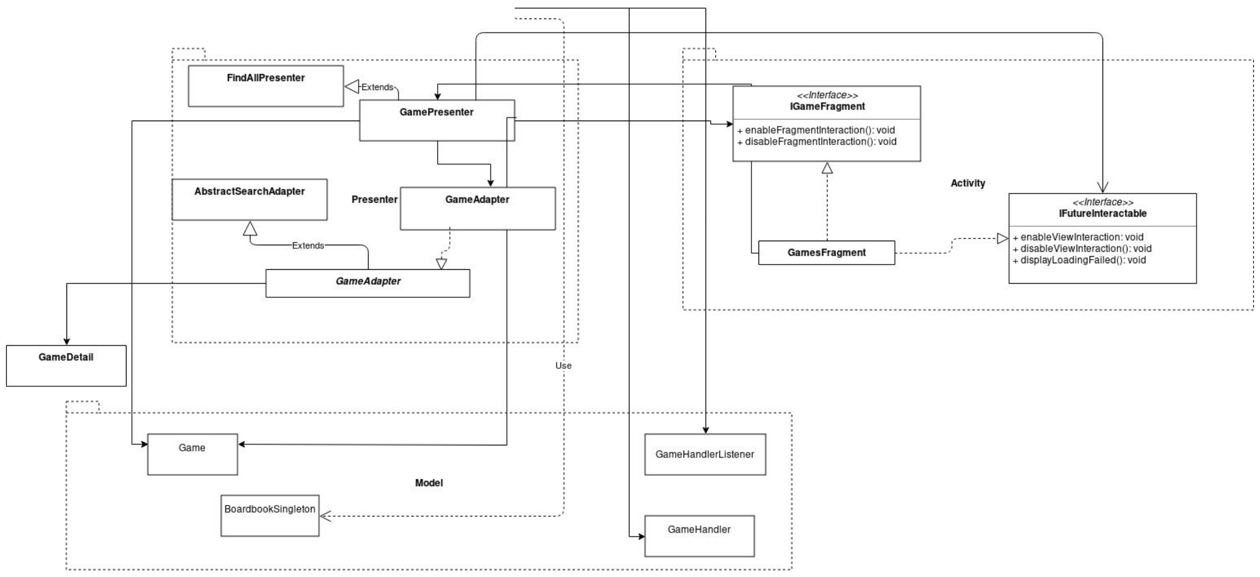
HomeActivity:



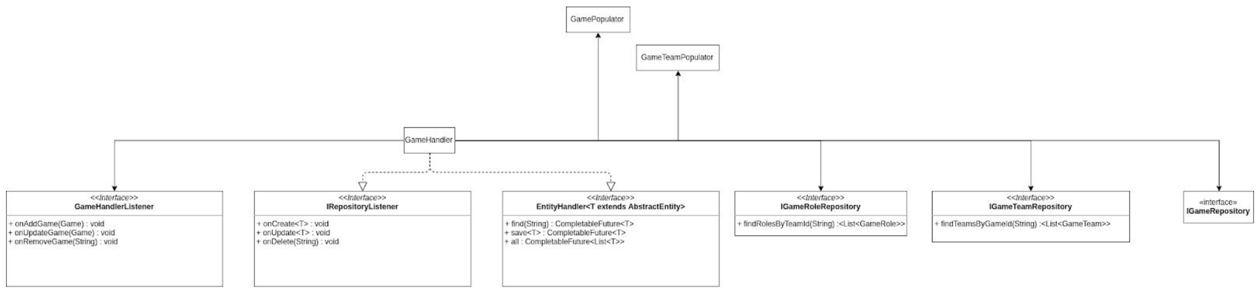
AddFriends + Friends:


FriendsFragment
AddFriendActivity
<<Interface>>
IFutureInteractable
+ enableViewInteraction: void
+ disableViewInteraction(): void
+ displayLoadingFailed(): void
<<interface>>
IAddFriendActivity
+enableAddFriendList() : void
+disableActivityInteraction() : void
+enableActivityInteraction (): void
+showErrorMessage (): void
+finishAddFriendActivity(): void
activity
<<interface>>
IFriendFragment
+enableFriendList(): void
+disableFragmentInteraction() : void
+enableFragmentInteraction(): void
+enableFragmentInteraction(): void
FindOnePresenter
FriendsPresenter
AddFriendsPresenter
Extends
AdapterPresenter
Extends
presenter
addfriends
Profile activity
AbstractSearchAdapter
Extends
FriendsAdapter
AddFriendsAdapter
Use
Use
Extends
User
Boardbooksingleton
UserHandlerListener
UserHandler
Model

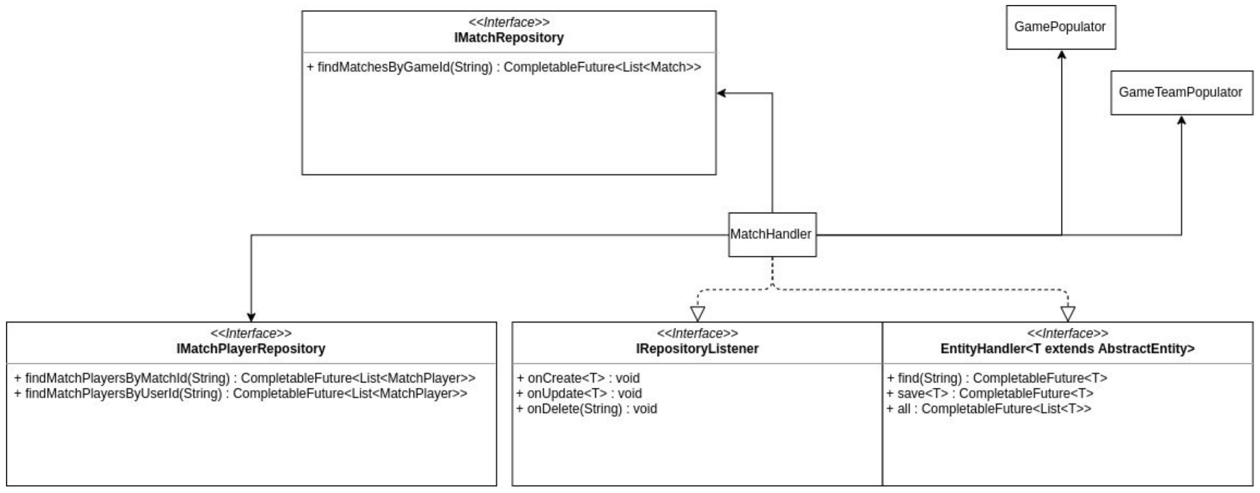
Game:



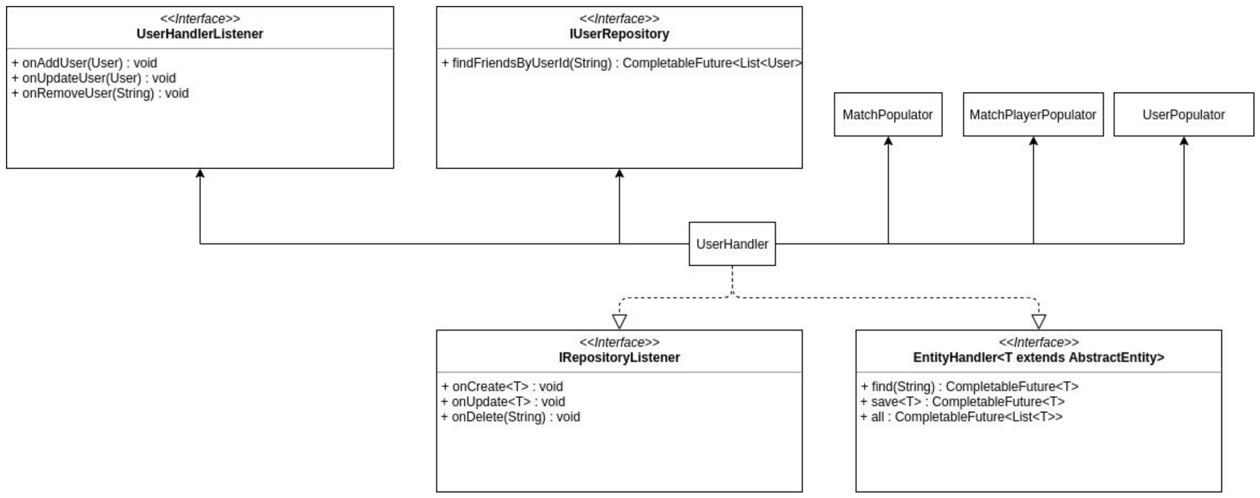
GameHandler:


GamePopulator
GameTeamPopulator
GameHandler
<<Interface>>
GameHandlerListener
+ onAddGame(Game) : void
+ onUpdateGame(Game) : void
+ onRemoveGame(String) : void
<<Interface>>
IRepositoryListener
+ onCreate<T> : void
+ onUpdate<T> : void
+ onDelete(String) : void
<<Interface>>
EntityHandler<T extends AbstractEntity>
+ find(String) : CompletableFuture<T>
+ save<T> : CompletableFuture<T>
+ all : CompletableFuture<List<T>>
<<Interface>>
IGameRoleRepository
+ findRolesByTeamId(String) :<List<GameRole>>
<<Interface>>
IGameTeamRepository
+ findTeamsByGameId(String) :<List<GameTeam>>
«interface»
IGameRepository

MatchHandler:

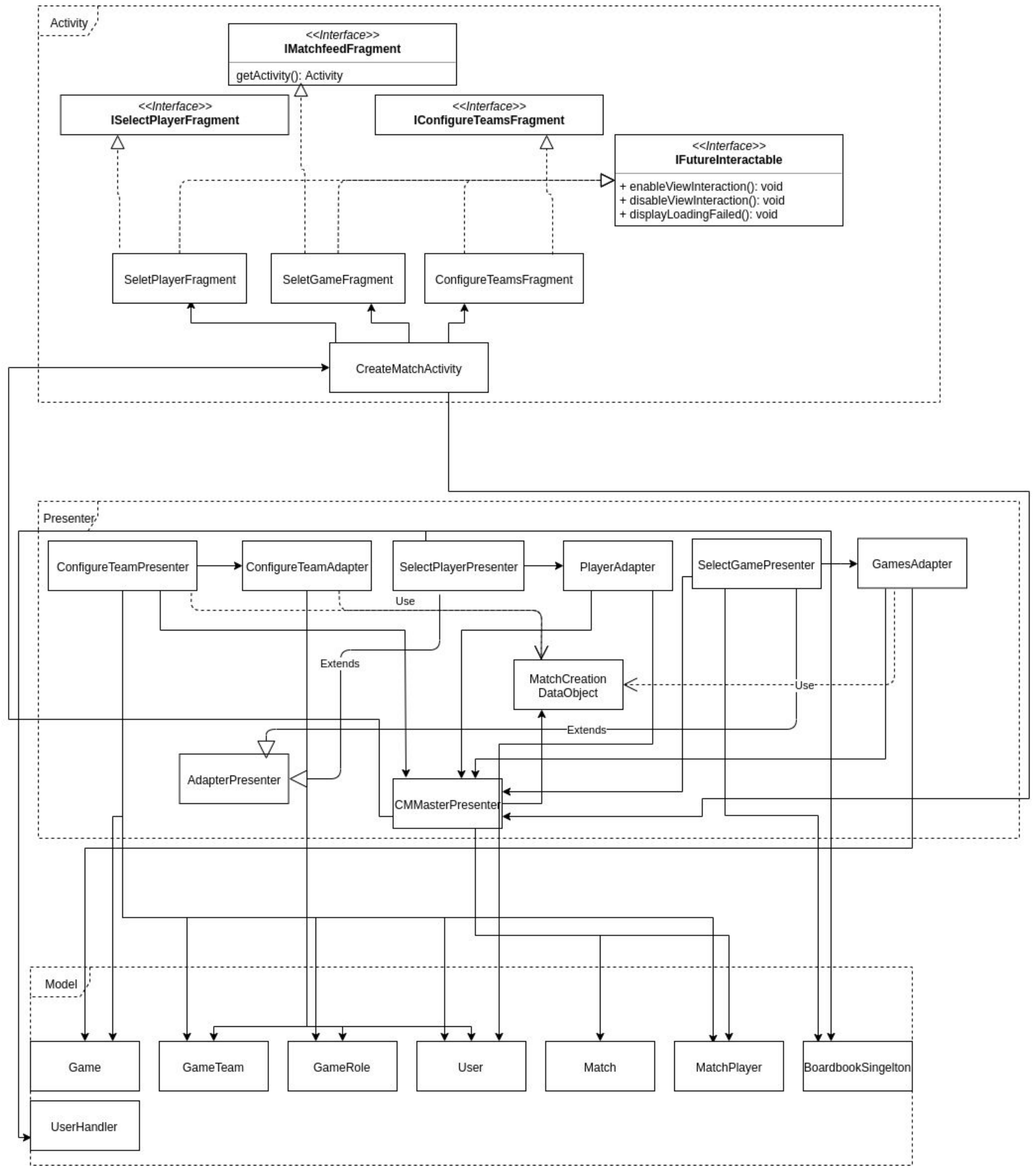

<<Interface>>
IMatchRepository
+ findMatchesByGameId(String) : CompletableFuture<List<Match>>
GamePopulator
GameTeamPopulator
MatchHandler
<<Interface>>
IMatchPlayerRepository
+ findMatchPlayersByMatchId(String) : CompletableFuture<List<MatchPlayer>>
+ findMatchPlayersByUserId(String) : CompletableFuture<List<MatchPlayer>>
<<Interface>>
IRepositoryListener
+ onCreate<T> : void
+ onUpdate<T> : void
+ onDelete(String) : void
<<Interface>>
EntityHandler<T extends AbstractEntity>
+ find(String) : CompletableFuture<T>
+ save<T> : CompletableFuture<T>
+ all : CompletableFuture<List<T>>

UserHandler:

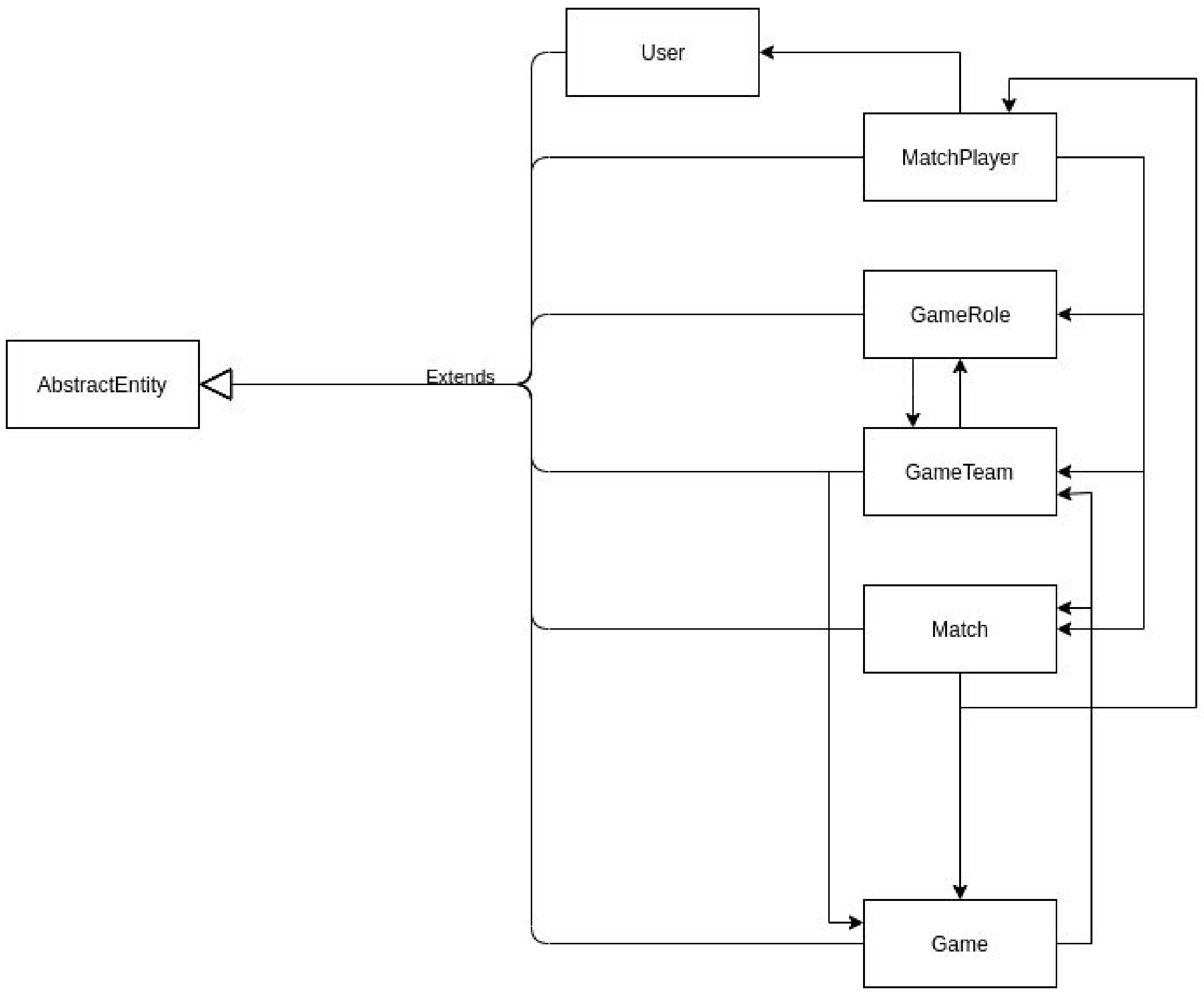


<<Interface>>
UserHandlerListener
+ onAddUser(User) : void
+ onUpdateUser(User) : void
+ onRemoveUser(String) : void
<<Interface>>
IUserRepository
+ findFriendsByUserId(String) : CompletableFuture<List<User>
MatchPopulator
MatchPlayerPopulator
UserPopulator
UserHandler
<<Interface>>
IRepositoryListener
+ onCreate<T> : void
+ onUpdate<T> : void
+ onDelete(String) : void
<<Interface>>
EntityHandler<T extends AbstractEntity>
+ find(String) : CompletableFuture<T>
+ save<T> : CompletableFuture<T>
+ all : CompletableFuture<List<T>>

Creating Matches:



Entity:


User
MatchPlayer
GameRole
AbstractEntity
Extends
GameTeam
Match
Game

4 Persistent data management

Boardbook uses firebase to store all its data. Therefore, all necessary data is requested when the application starts and no data is stored on the users device.

The model is completely abstracted from the database by using the Repository pattern. This ensures that the database could completely be swapped out for another storage solution. There are currently two implementations of the repositories, the firebase version and the mock version primarily used for testing.

5 Quality

Code Quality

Firebase can be slow on initial load. The speed has been improved by limiting cascading loads and caching loaded data and relationships, but it is still slower than we would like it to be.

Sometimes people are shown as added to a match when they are actually not added. This is a visual bug and we can’t find a solution to this problem before the deadline.

Testing

For testing we use unit tests. They are located in the test directory in the project. These are run with travis ci to ensure that the code will build and that the code is tested.

5.1 Access control and security

Boardbook enables users to create their own account with an email and a password. The application uses the authentication service of Firebase in order to save and compare this data when it is necessary. Therefore, Boardbook does not handle any of its own login authentication.

6 References

Firebase: https://firebase.google.com/
Android Studio: https://developer.android.com/studio

Android: **https://developer.android.com/**