Final report for KahIt

Working Group: Anas Alkoutli
Oussama Anadani
Mats Cedervall
Johan Ek
Jakob Ewerstrand

25 October 2019
Version 1.0

Requirements and Analysis Document for KahIt

Working Group: Anas Alkoutli
Oussama Anadani
Mats Cedervall
Johan Ek
Jakob Ewerstrand

25 October 2019

Contents

1 Introduction

KahIt is an application developed for the android operative system and is meant to be run on either a phone or a tablet. It is a quiz application that is closer to a party game. It can be played with friends in a multiplayer mode, with one player acting as host or it can be played locally on one device. When hosting a game friends can connect using *Nearby Connections* with up to 8 players. *KahIt* is designed to have items that can boost the player's score or reduce it. *KahIt* provides a verity of cosmetic items that do not affect players status but are shown for other players.

1.1 Definitions, acronyms and abbreviations

- Hotswap: A game mode were a multiplayer game is played on one device by rotating the active user and letting them preform one action per rotation, resulting in a multiplayer experience that only requires one device.

2 Requirements

2.1 Epics

As a: person who likes to have fun with friends I want: a way to challenge their knowledge in a fun and enjoyable way since the old way with cards are boring and limited.
As a: user I want: to play a quiz-like party game since I need something fun to do with my friends.
As a: Host I want: to have the setup time for a game to be quick and easy, so it does not become cumbersome.
As a: user I want: to be able to set up a game with my friends because it's more fun to play with them than with strangers.

2.2 User Stories

Story identifier: sFQuestionl
Story name: The first question
Status: Completed

As a: user I want to: answer a multiple choice question because answering questions is fun.

Functional Confirmation:

- A questions is provided to the user and the user can answer it.
- The user can choose between (4) alternatives of possible answers.
- The user is notified if their answer is correct or incorrect.

Non-functional Confirmation:

- The whole question is visible no matter the size of it.

Story identifier: sMQuestions
Story name: More than one question
Status: Completed

As a: user I want to: be able to answer more than one question because only one question is not enough for the game to be fun.

Functional Confirmation:

- After answering a question a new one should be provided.
- A group of question should of the same category.
- The new question should not be a question that has already been answered. Unless all questions of that category have been answered.

Non-functional Confirmation:

- There should not be a long load time between the different questions.

Story identifier: sHGame
Story name: Host a game
Status: Completed

As a: user I want to: be able to host a game so that my friends can join.

Functional Confirmation:

- The host does not have to provide any extra information except a possible password to setup a game.
- The host can remove players from the "lobby".

Non-functional Confirmation:

- Hosting a game should be low effort.
- Hosting a game should not take too much time.

Story identifier: sJGame
Story name: Joining a game
Status: Completed

As a: user I want to: be able to join a game that someone else is hosting because playing with someone is more fun than playing alone.

Functional Confirmation:

- The app searches for a "room" and when it finds a game it populates a list for the user to pick.
- The user is only connected if they pick one of the entry's in the list.
- The user can leave the "room" and terminate the connection at any point.
- When the user is prompted they are given some sort of identifying info about the game/room they are about to join.

Non-functional Confirmation:

- Joining a "room" should require very few steps.
- Connecting to a host should not take too long.

Story identifier: sHotSwap
Story name: Hot swap
Status: Completed

As a parent I want all of my children to be able to play on the same device because it's too expensive to get all of them their own phones.

Functional Confirmation:

- The game has a "Hot swap" mode where you can select the amount of players at the start and subsequently play on one device.
- The game rotates the active user allowing all player to participate in the game.
- The game adapts the different game modes to the single device limitation.
- All local players are given the same question.

Non-functional Confirmation:

- None

Story identifier: sStartGame
Story name: Start a multiplayer game
Status: Completed

As a: host I want: to be able to start the game because that is why I am hosting it in the first place.

Functional Confirmation:

- When the user starts the game it should start for all connected users.
- The game should not start until the host has chosen to start it and all users have chosen that they are ready.
- The game should show the same questions to all users.
- The game should not move on from a question until all users have provided an answer or the time has run out.
- If a user disconnects from the game, the game should continue unless the player that left was the host. If the host leaves the game all other users should be disconnected.

Non-functional Confirmation:

- All delays and slowdown should be avoided.

Story identifier: sUsingPowerUps
Story name: Using power ups.
Status: Completed in hotswap, Partially completed in multiplayer

As a: user I want: to Be able to use power-ups and bonuses on questions if i'm sure i know the answer since I like games that are tactical.

Functional Confirmation:

- A player will be able to buy power ups with points earned through answering correctly.
- A power will last for a limited period only ex. three question rounds.
- A player can only buy a certain power up once.

Non-functional Confirmation:

- None

Story identifier: sLeaderBoards
Story name: Leader boards
Status: Completed

As a: user I want: to continually get an update on whom is in the lead since It can be hard to keep the score in my head.

Functional Confirmation:

- After each question the players will be able to see a list of the players with their amount of points and their placement.
- The players in the leaderboard are sorted after the amount of points the players have.

Non-functional Confirmation:

- None

Story identifier: sLottery
Story name: Lottery
Status: Partially completed in hotswap, Uncompleted in multiplayer

As a: player I want: to have power ups and bonuses that reward good players since it makes the game more interesting and competitive.

Functional Confirmation:

- When answering correctly a player will have more chance to win power ups which will protect the players placement and take it even higher increasing the points earned per question.
- A working lottery that distributes winnings to all players.

Non-functional Confirmation:

- The lottery process is good looking and immersive.

Story identifier: sCosmetics
Story name: Cosmetics
Status: Partially completed in multiplayer since players can buy cosmetic items but they are not synced so that other players can them. Completed in hotswap

As a: User I want: to be able to buy cosmetics in game since it makes the game cooler.

Functional Confirmation:

- The user will be able to buy cosmetics such as icons or special colors to stick out when being high on the leader-board.
- The cosmetic items do not give any advantage in game but it adds character and customization to the game.

Non-functional Confirmation:

- None

Story identifier: sLotteryDraw
Story name: Lottery draw
Status: Uncompleted in multiplayer, Completed in hotswap

As a: player I want: to get some randomized items from the lottery.

Functional Confirmation:

- All the players will be able to get a random item each time the lottery ends.

Non-functional Confirmation:

- None

Story identifier: sName1
Story name: Name selector
Status: Completed

As a: user I want to: be able to set my own custom player name (just as you do when e.g. bowling) since it makes it even more fun.

Functional Confirmation:

- The user can choose a name before the game starts.
- The users name is displayed on various "score screens" etc.
- The username has a character limit, as not to mess up the layout.

Non-functional Confirmation:

- None

Story identifier: sMusic
Story name: Background music
Status: Completed

As a: user I want: to be able to have background music because it makes the game more interesting.

Functional Confirmation:

- Some kind of music starts playing when the user opens the app.
- Some kind of music starts playing when the user changes the category.

Non-functional Confirmation:

- The music is suitable for the application.

Story identifier: sToggleMusic
Story Turn off sounds
Status: Partially completed

As a: user I want: to have the option to turn off background music because sometimes it can be distracting and annoying.

Functional Confirmation:

- There exists a function that allows the user to turn off the music.
- The preferred setting is saved between app sessions.

Non-functional Confirmation:

- You should intuitively be able to find the place where you can turn off the music.

Story identifier: socialInteract
Story name: Social Interaction
Status: Uncompleted

As a: user I want to: The game should force players out of the couch in some way since a plain quiz can be to boring for a party.

Functional Confirmation:

- The game has at least one game mode where at least one player has to get up and do something e.g. "pose" in front of a camera.

Non-functional Confirmation:

- The game should be fun and players should laugh.

Story identifier: sTieCase
Story name: Tie case
Status: Uncompleted. The games goes on as long as the players wish therefore a tie case never occurs if the players choose to continue playing.

As a: competitive person I want: there too be an extra round if two players have the same score because THERE CAN ONLY BE ONE WINNER!!!!

Functional Confirmation:

- In a case of tie the game goes into sudden death mode where a player wins as soon as the player is ahead of the other player.

Non-functional Confirmation:

- None

Story identifier: sQuit1
Story name: Quit
Status: Uncompleted

As a: user I want to: be able to quit the game and get a winner since sometimes I want to stop playing and do something else.

Functional Confirmation:

- The game has an "escape hatch" available: during or after questions or after "rounds".
- Host and player can both use the function.
- A Score screen is displayed for the one who left.
- If two players are left and another player uses the function a score screen is displayed for both.
- If a player leaves other users are notified and asked whether or not they want to continue.

Non-functional Confirmation:

- None.

Story identifier: sDifficulty1
Story name: Difficulty selector
Status: Uncompleted

As a: child I want: to be able to set the difficulty of the quiz because some of the questions are to hard for me.

Functional Confirmation:

- The user can choose the difficulty of the quiz at the beginning.
- The questions given during the quiz are not harder then the given limit.

Non-functional Confirmation:

- None.

Story identifier: sPause
Story name: Pause
Status: Uncompleted

As a: user I want: to

Functional Confirmation:

- The game require some kind of input e.g. a button or someone touching the screen after each question before the game continues.
- User and Host can "press" continue and the game will go on to the next question.

Non-functional Confirmation:

- None.

Story identifier: sEliminationMode
Story name: Elimination mode
Status: uncompleted

As a: player I want: to eliminate other players since it makes players play more seriously.

Functional Confirmation:

- In a certain game mode it will be possible to eliminate other players by answering correctly and quickly or being on a streak.
- The player will not be able to choose who to eliminate but should answer correctly and quickly to not get eliminated.

Non-functional Confirmation:

- None.

Story identifier: sAbotage1
Story name: Sabotage
Status: Uncompleted in multiplayer. Partially completed in hotswap

As a: user I want: to Sabotage for other players so that I can catch up if I'm behind in score.

Functional Confirmation:

- The game has different types of "debuffs" that a user can give away before each question.
- The user can choose freely whom he wants to target.
- The debuff only applies for that question.
- Debuffs can stack so if one players is targeted with multiple debuffs the effect should e.g. be double.

Non-functional Confirmation:

- None.

Story identifier: sCategory
Story name: Categories
Status: Completed

As a: user I want: to be able to see the "theme" or category before each question.

Functional Confirmation:

- The users will be able to see the round/questions theme/category during the game.

Non-functional Confirmation:

- None.

Story identifier: sCategory1
Story name: Choose Categories
Status: Completed

As a: user I want: to have the ability to choose the category on some questions.

Functional Confirmation:

- The users will be able to choose between different categories before some questions or rounds during the game.

Non-functional Confirmation:

- None.

Story identifier: sPeopleNotFacts
Story name: People not facts
Status: Uncompleted

As a: user I want to: be able to win without being the one with the most knowledge since i tend to be good with people not facts.

Functional Confirmation:

- In a certain game mode the player will be able to type in a question which could be personal.
- After typing a question the user will be able to choose one correct answer.
- During the time waiting the players will be able to enter a lottery for bonuses and power ups.
- This game mode can make a party more fun since the user who is typing the questions can know which of the players knows the user best.

Non-functional Confirmation:

- None.

Story identifier: sNotOnlyName
Story name: Not only name
Status: Uncompleted

As a: user I want: to be able to distinguish players based on not just the name since some of my friends have the same name.

Functional Confirmation:

- Every player will have a unique icon with the photo of the player.
- Players can also choose a player name which can be what ever name they like.

Non-functional Confirmation:

- None.

Story identifier: sNonTrivia
Story name: Not only trivia
Status: Uncompleted

As a: user I want: to play a game where i can interact with the players in the same room since only knowledge-based questions tend to be to boring for my parties.

Functional Confirmation:

- The game has a mode where you are forced to interact with other users in real life. e.g. a mode where you answer these types of questions "Which player would most likely cheat on a test?" and the most popular answer wins.

Non-functional Confirmation:

- None.

Story identifier: sCustomQuiz
Story name: custom game
Status: Uncompleted

As a: user I want: to create different quizzes since it adds more customization to the game.

Functional Confirmation:

- The user will be able to add their own quizzes to the game.
- The user will be able to host a game with the user's created quizzes
- Users will be able to answer questions created by other users.

Non-functional Confirmation:

- None.

Story identifier: sContPlaying
Story name: Continue playing
Status: Partially Completed

As a: user I want: to be able to continue playing even if one person leaves since it would be unpractical to start a new game if you are near the end.

Functional Confirmation:

- When a user leaves the game the remaining players should be allowed to decide if they will continue as normal or end the game. However it was the host that left the game the game ends.
- The disconnected players stats should still be displayed in the leader-board but appear grayed-out.
- The game should treat the disconnected player as if they don't exist (not include in any calculations) but keep their data on the of-chance that they were to reconnect.

Non-functional Confirmation:

- There should not be any hangups or slow downs after a user disconnects.

Story identifier: sGameModes
Story name: Game modes
Status: Uncompleted

As a: user I want: to Have a variety of questions/modes since it would make the game more fun.

Functional Confirmation:

- The user will be able to choose different game modes when hosting a game.
- Different game modes are played out differently where in some mode answering correctly is the better option while in another answering quickly is better.

Non-functional Confirmation:

- None.

Story identifier: sAddQuestion
Story name: Add question
Status: Uncompleted

As a: user I want: to Be able to write and add my own questions since my own question can be more relevant to the setting.

Functional Confirmation:

- The user will be able to add questions.
- The user will choose the answers for the question where one answer must be right.
- The user will be able to choose the category of the added question.

Non-functional Confirmation:

- None.

Story identifier: sDraw
Story name: Draw
Status: Draw

As a: artist I want: to be able to draw pictures while playing since I'm a good artist.

Functional Confirmation:

- In some cases the answer to a question will be to draw something.
- Points are distributed by popular vote.
- This is a feature when playing in teams where one player in team draws and the other player in the team has to guess what the first player is drawing.

Non-functional Confirmation:

- None.

Story identifier: sRejoinGame
Story name: Rejoin a game
Status: Uncompleted

As a: user I want: to rejoin the game if I or anyone else disconnects since having to wait for a whole new round is boring.

Functional Confirmation:

- If the player gets disconnected from a game the player will still be able to rejoin the game.
- The game which the player was disconnected from should still be running in order for the player to rejoin.
- The player will not lose points upon disconnecting but will not gain points for the missed questions.

Non-functional Confirmation:

- None.

Story identifier: sTeamMode
Story name: Team mode
Status: Uncompleted

As a: user I want: to be able to play in teams since it adds more interaction and communication with other players.

Functional Confirmation:

- In this game mode players will work together against other teams in order to win.
- Each team will have a certain amount of player where each player in a team can answer for a part of the the whole answer.
- For a team to get points must each player's part of the answer be correct.

Non-functional Confirmation:

- None.

Story identifier: sChooseHost
Story name: Choose game settings
Status: Completed

As a: host I want: to be able to decide on the setup of the game before I host because this will allow me to personalize the game more.

Functional Confirmation:

- a functional req

Non-functional Confirmation:

- None.

2.3 Definition of Done

All user stories and features that are implemented in the application have to fulfill the following requirements to be considered done.

- All code clears all of it's tests. And all classes in model are fully tested.
- All code has relevant comments explaining it's function and purpose.
- All public classes are documented with Java Doc.

- All comments and documentation is written in English.
- All classes and variables share a common naming convention.

2.4 User interface

MainPage has quite a big button that follows the Clear Entry Point design pattern. It makes the navigation easier for the user. Two additional buttons, Settings button navigates to settings page and AboutKahit button that shows a page contains an explanation of the app(1).

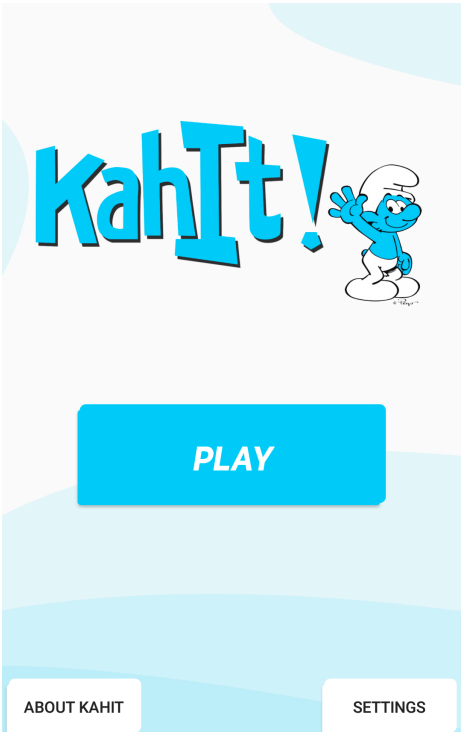

Figure 1: The main page of the application

When the play-button is clicked the app navigates to ChooseGameView. This view contains Three buttons: Host, Join and Hotswap(2).

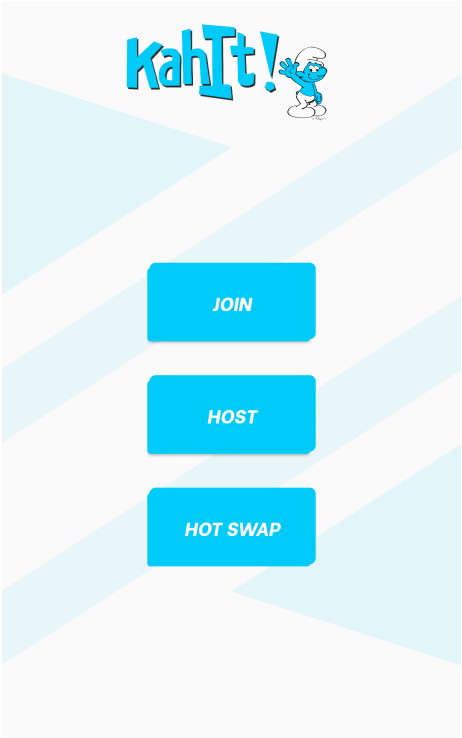

Figure 2: ChooseGameView presents the user with three choices.

- The host button navigates forward to CreateLobbyView where the user can create a lobby as a host and the lobby will be visible to other players nearby. Once all users are ready then the host should be able to start the game. On this page the Good Defaults design pattern is used, where the user already has a default value in every textField that needs to be filled. When the user is ready so they can click on Ready-button and wait for the host to start the game(3).

Figure 3: The multiplayer lobby

- Join button navigates to JoinLobbyPage where the user can join a game.

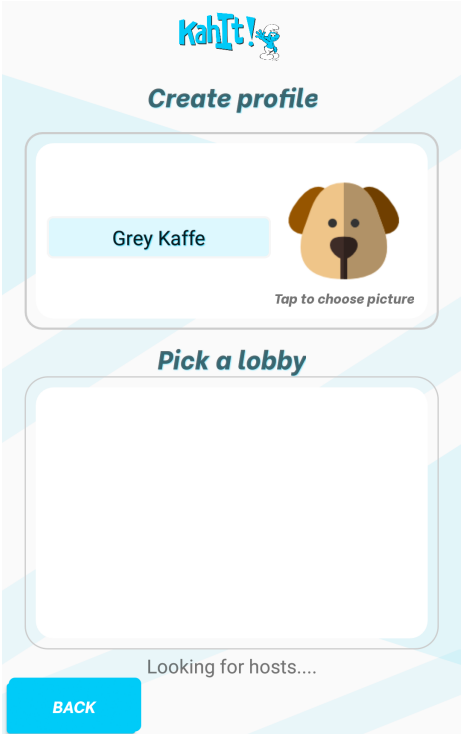

Figure 4: The lobby finder

- Hotswap button navigates to HotswapGameMode-page where the user should be able to choose a gameMode depending on how they want to play. Right now game modes are not implemented due to time restrictions.

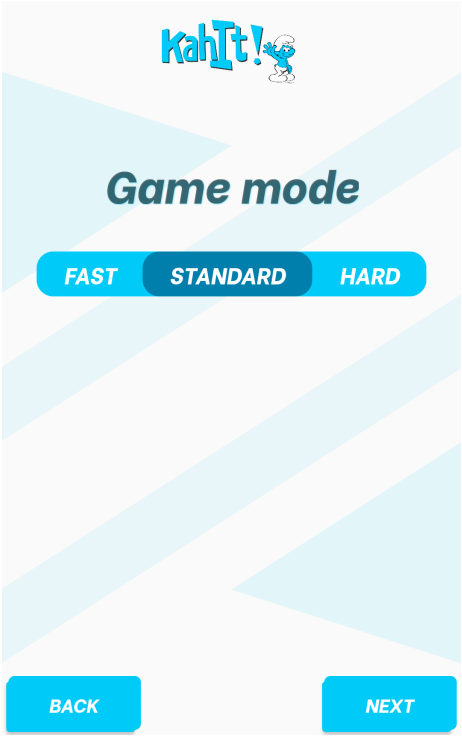

Figure 5: The view for selecting the gamemode for a hotswap game

Next-button takes the user to GameLobby for hotswap mode. In hotSwapLobby, 8 players maximum can be added to the game. Then the game can be started by Go-button.

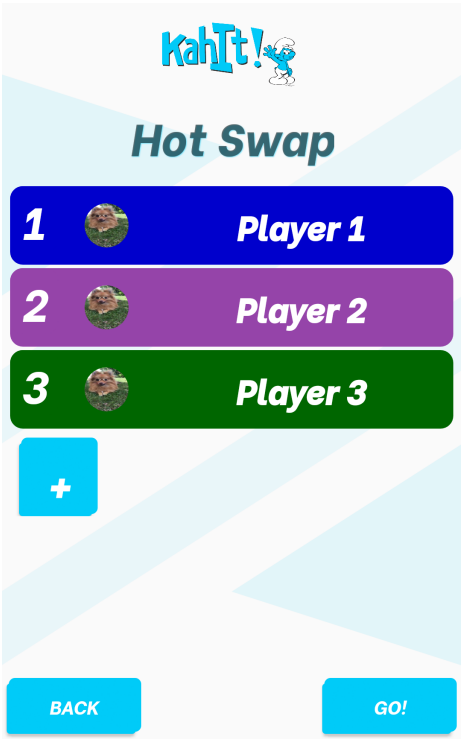

Figure 6: The screen to add players to the hotswap mode

Before starting the game, the user will pass through PreGameCountdown Page that gives them few seconds to get ready.

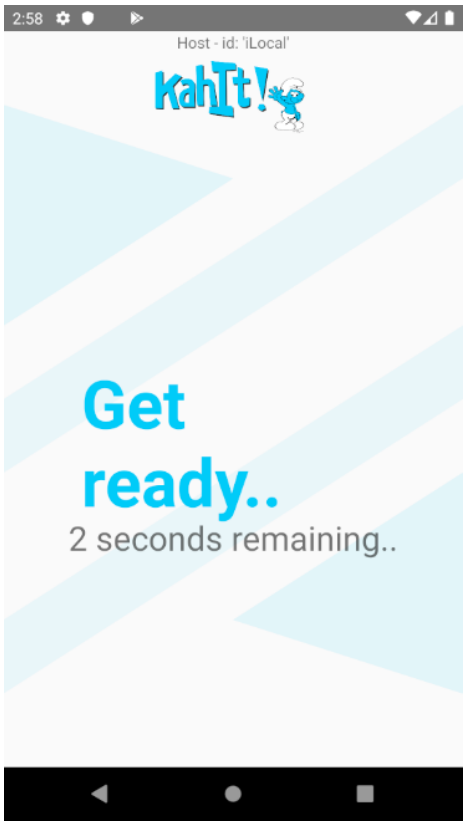

Figure 7: The pre game countdown view.

When the game has been started, there are 2 modes: Hotswap and multi-player. In both cases, the user will pass through the same Question page.

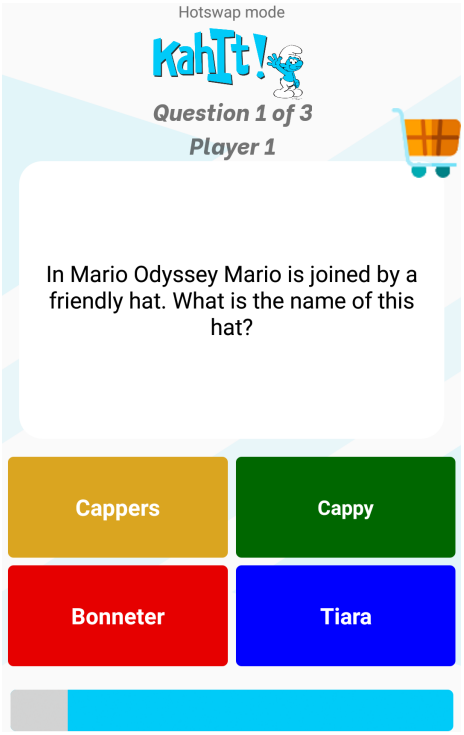

Figure 8: The view for answering questions

During the game, the Store-page is available all time to the users, where they can but items(buffs, debuffs or vanity items) depending on their points that they get when answering questions.

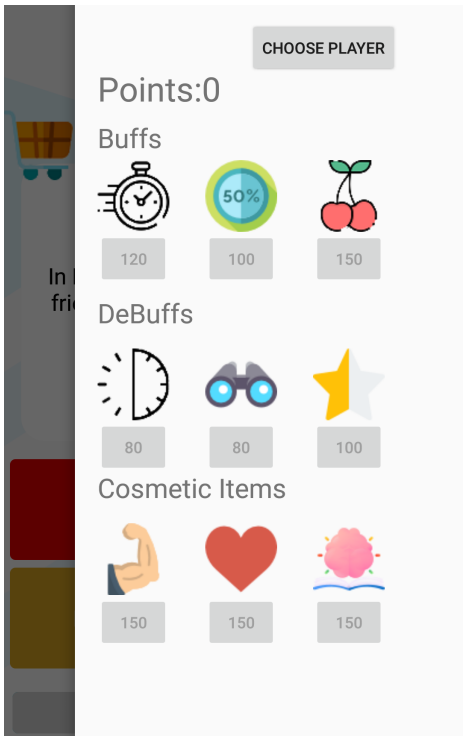

Figure 9: The store

After each question, Score-page will be shown up and shows scores for all users

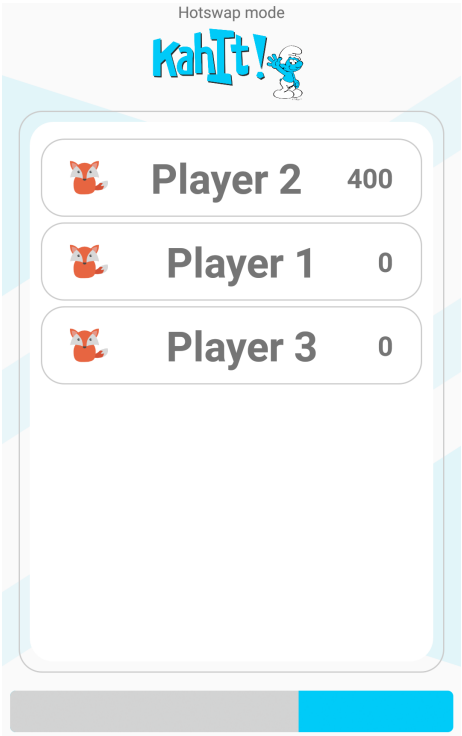

Figure 10: Score-page displaying the score of 3 players in a hotswap match

After each round(2 questions), the user will pass through Lottery-page that will randomize items to all users.

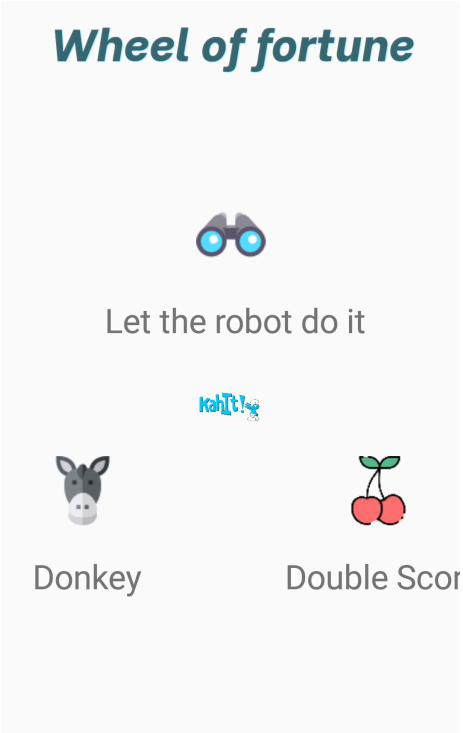

Figure 11: The lottery displaying the winnings of 3 players

3 Domain model

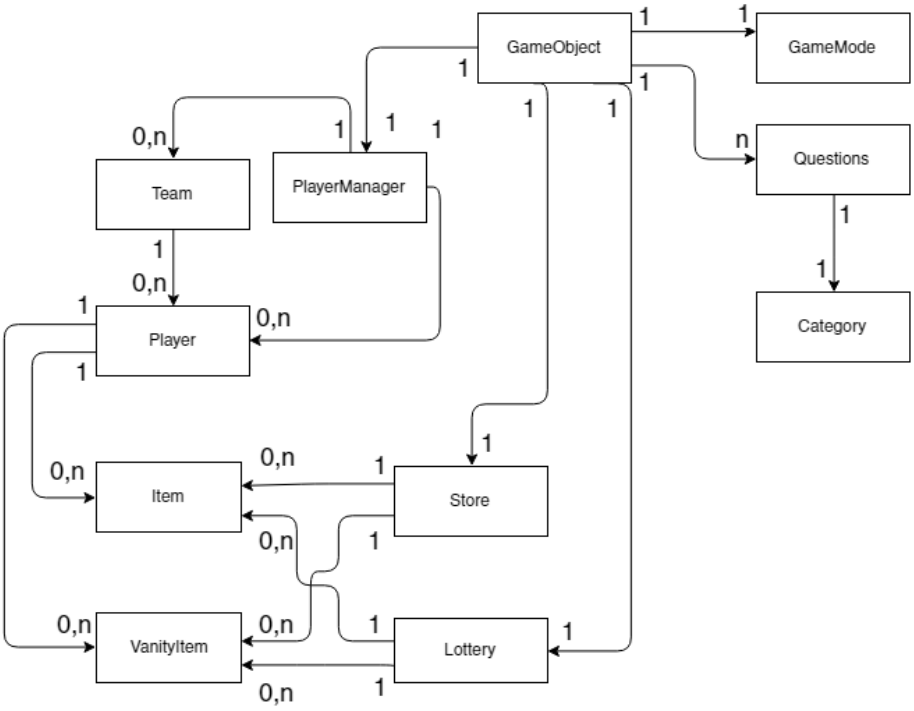

Figure 12: Domain Model

3.1 Class responsibilities

3.1.1 QuizGame

The class QuizGame acts as the aggregate object of the model, in other words the QuizGame is the access point through which outer parts of the application communicates with the model.

3.2 PlayerManager

PlayerManager is responsible for handling operations on players and teams and holding them.

3.2.1 Questions

The Question class is responsible for holding all information that is relevant to a question. E.g actual question and possible answers. The class is also responsible for checking if a given answer is correct.

3.2.2 Category

Category is an enumerator that is responsible for representing the different categories that a question can be a part of. The enumerator is also responsible for a number relevant behaviour such as returning a category based on a text string or an index.

3.2.3 Store

The Store class is responsible for holding items and making it possible for players to buy items that can affect their score during gameplay. This class holds values such as which items that are available in the store. This class also holds the necessary logic to complete a transaction.

3.2.4 Lottery

The Lottery class is responsible for managing the lottery prize giveaway system.

3.2.5 Modifier

The modifier class is responsible for the items that can alter the stats of a player. A modifier holds all the values which determines if the stats of a player will be affected positively or negatively. A modifier can affect the stats by the score of a question or the time taken to answer.

3.2.6 VanityItem

The VanityItem class is responsible for the cosmetic items in the game. This class holds values such as the price and the duration of an item as in how many rounds this item can last.

3.2.7 Player

The Player class is responsible for holding all player specific information. Attributes such as current score, player name, id, modifier stats and owned vanity items.

3.2.8 Team

Team class is responsible for keeping track of players in a specific team, team combined score and team name.

Explanation of responsibilities of classes in diagram.

4 References

https://developers.google.com/nearby/connections/overview

List all references to external tools, platforms, libraries, papers, etc.

System Design Document for KahIt

Working Group: Anas Alkoutli
Oussama Anadani
Mats Cedervall
Johan Ek
Jakob Ewerstrand

25 October 2019
Version 1.0

Contents

1 Introduction

KahIt is an application developed for the android operative system and is meant to be run on either phones or tablets. It is a quiz application that is closer to a party game. It can be played with friends in a multiplayer mode, by one player hosting or locally on one device. When hosting a game friends can connect using *Nearby Connections* with up to 8 players. *KahIt* is designed to have items that can boost the player's score or reduce it. *KahIt* provides a verity of cosmetic items that do not affect players status but are shown for other players players.

1.1 Definitions, acronyms, and abbreviations

- Hotswap: A game mode were a multiplayer game is played on one device by rotating the active user and letting them preform one action per rotation, resulting in a multiplayer experience that only requires one device.
- MVVM : Model view view-model
- XML : Extensible markup language.
- STAN : A dependency analisys tool.
- JUnit : A java unit testing framwork.
- Bus : A library that enhances loose coupling since it uses a publisher/subscriber pattern. Bus can also be mentioned as event bus in this document.
- Map : A component in Java that acts similarly to a list since it holds multiple values. A map uses a key to get the wanted value instead of an index

2 System architecture

This section will present the system architecture of the application. It includes what components the application uses, how it uses them and how they work together

2.1 Android basics

Kahit is an andriod application which has been built and tested using Android Studio IDE. It uses SDK28 which is Android Pie but has support down to SDK24. SDK28 was chosen because it was easy to build the application with and test it on our own devices.

This application uses Gradle which is an open-source build automation tool. Gradle makes it easier to download and configure dependencies or other libraries.

2.2 Application components

Kahit is built with MVVM structure, which enhances separation of concern. This structure is similar too the MVC design pattern but a controller is replaced with a view model. The controller holds a dependency on the model in a standard MVC structure but in MVVM structure a view model depends on a repository which depends on the model. This section will present the components of the application and provide an overview on how they depend on each other.

2.2.1 Model

This component holds the data of the application. This is also called the domain layer since it represents the domain model and it should be invisible to the user and separated from other components such as the view-model and the view In other words this component holds the data needed for the application to work but not how it should behave

2.2.2 View-model

This component falls under the presentation layer of the application. As the name suggests it as a component between the view and the model therefore it contains all the logic needed for the view to work but also gets all the data needed from the model.

The View-model acts as a converter that gets data from the model and sends it to the view in way it can handle the data. It can also work the other way around where the model can be manipulated by it if an event is triggered in the view for example.

2.2.3 View

Similarly to the View-model falls the View component under the presentation layer. This component is the only component the user directly interacts with. The view's only purpose is to present data to the user but does not hold or handle data within itself. The View can manage the user inputs such as clicks and gestures which can trigger events that the View-model can act upon

2.2.4 Application events

Application events holds all the necessary events for the application to function. An event acts as a switch where the application takes action as soon as it is flipped. An example of this would be waiting for all players to be ready. When all players are ready an event will be sent on the event bus which notifies the components that are listening for this event that all players are ready and the game can start.

2.2.5 Repository

The repository package functions as a connection point between the network, the domain layer which is the model and the presentation layer which is the view and the model. This package has a singleton class the can be accessed from the view model to get information from the model send information to it. In this case the view model does not directly depend on the model and the repository makes easy to maintain the connection with the network manager.

2.2.6 Music services

This package handles all the audio related issues in the application. The user can choose to run the application with or without music. The application provides various tracks for the different categories.

2.2.7 Database services

The database services connect the program to a Firebase realtime database and is used to fetch the data required to instance the questions and items that are used by the rest of the program. This component is used by the different factory classes in the program as well as a number of the views.

2.2.8 Network manager

The Network manager component contains all of the details surrounding the game network communication. It cares for the setup and operation of the Nearby Connections API integration. It provides methods for the host to transmit model state changes to its clients. It also supplies several callback interfaces to the repository,

used to update the model and the general state of the app according to what is suitable to the state of the host.

When a host makes any action that results in changes within the model, then the host also broadcasts the change through the Nearby Connections API. Meanwhile clients are only allowed to request changes to be made by the host and always update their own model based on what the hosts broadcasts.

2.3 Application flow

A normal flow of the application can be described by explaining a simple task such as answering a question. When the game is started by the host an application event is sent to the network manager which sends it to all players. The application loads the question from the database and sends it as an event. The question view model loads the event and extracts the need information for the view.

After the question is displayed and the player has answered the time and answer are sent to the player's model through the repository to calculate and update the player's score. Since the player has answered a question, the repository sends an event to the host that the player is ready for the next question. Through the whole application flow will the music services be playing different tracks for different situations such as category related tracks or waiting tracks while waiting for the game to start.

3 System design

3.1 Model view view-model: MVVM

Instead of implementing a standard MVC pattern the application uses a Model view view-model (MVVM) which is one of the standard solutions to facilitate communication between the different parts of an Android application. In MVVM each of the views (xml files) of the application has a view class that handles the input of the user and appearance of the view, and a view-model class that holds all data for the view and performs operations on the data and updates the view. To facilitate the communication between the view and view-model, the view hold a reference to it's view-model and calls method in it. While the view-model uses live-data to update the view. Live-data is variables that are synchronised between classes, if the data is updated in the view-model a listener method is called in all classes the listen to the data in this case the view. The view-model fetches it's data by either using method calls or by getting it pushed by listeners or events through a event-bus.

3.2 Package structure

Figure 1 shows the structure of the packages in the application.

- View package provides classes that expose basic user interface classes that handle screen layout and interaction with the user.

- The ViewModel is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel allows data to survive configuration changes such as screen rotations. Architecture Components in Android provides ViewModel helper class for the UI controller that is responsible for preparing data for the UI(view). ViewModel objects are automatically retained during configuration changes so that data they hold is immediately available to the next activity or fragment instance(View). For example, if you need to display a list of users in your app, make sure to assign responsibility to acquire and keep the list of users to a ViewModel, instead of an activity or fragment(View parts)

- Same figure shows that Repository-package is the main core of the whole application. Basically, repository package is the business Layer. it holds everything related to the application's Domain. It uses a lot of business objects and creates a lot of new objects, too. It knows that all these objects need to be persisted or retrieved from some storage. For instance, the repository has like these shelves where it puts all the new objects on. And everytime it needs to work with an object it just gets it from the shelf, does the work then puts it back. The repository doesn't know how these shelves work and how they're actually storing or retrieving the objects. It does only know that it just puts the business objects there and it gets them back exactly how they were.

- Model package holds all the data and the business that will be provided in the app. It depends on non of the packages because its the logical data that could be changed over the time and provided to different work environments.

- Database package provides all data to model-package which means it only depends on the model.

- View package depends on ViewModel to provide all the changes that happens when the user interacts with UI(screen). In additional, ViewModel is a path to the other packages and to the Repository package that in it's turn is a path to get the data form the model.

- NetworkModel package holds the all classes that provide network business that in turn has no dependency on other packages.

- BackgroundMusicService holds the music class that provides music as a service in the background of the application.

- ApplicationEvents contains plain Java Objects that are used as events on the GreenRobotEventBus.

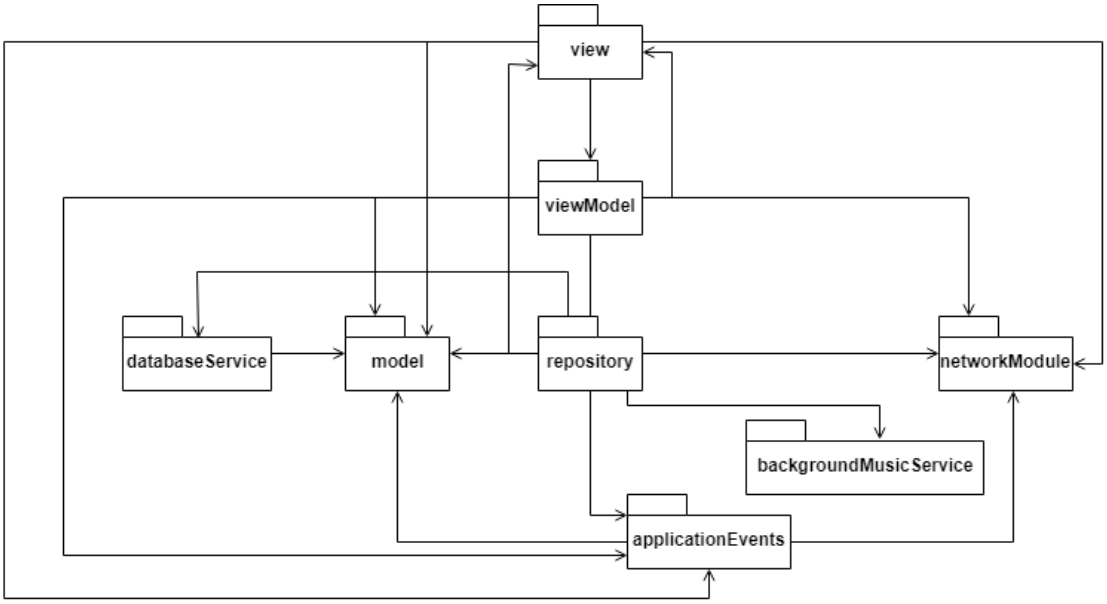

Figure 1: High-level package view

3.3 Model - Design model vs Domain model

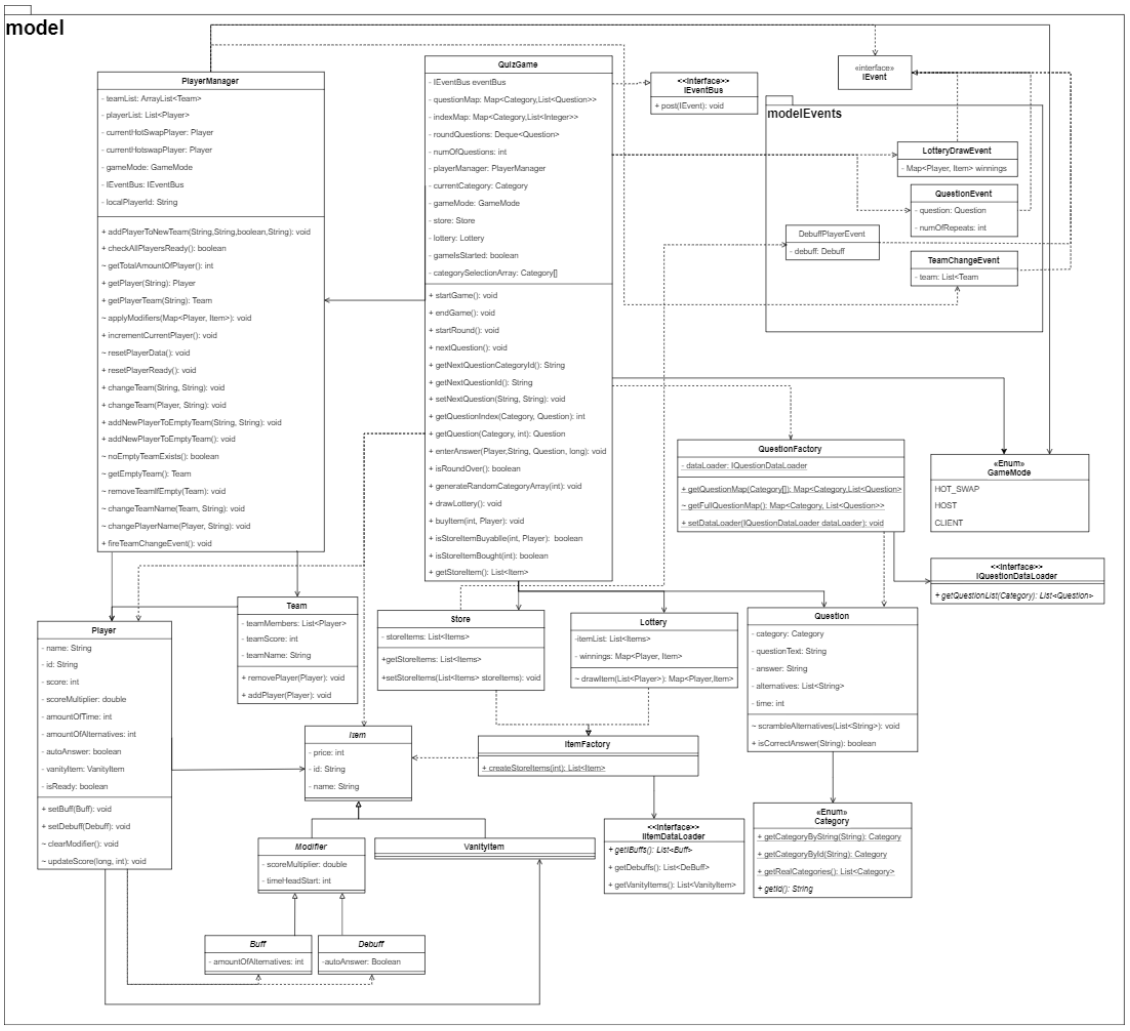

Figure 2: Design model

3.4 Database Service

The package DatabaseService is responsible for the connection to the Firebase Realtime database and the fetching of data from it. To accomplish this responsibility it has two data loader classes that connect to the database and fetch the question and item data and then creates their respective objects. The package also has data holder classes, these classes mimic the structure of the database and are used to ”cast” a part of the data structure (such as the data for a question) into a object. These classes also have methods that create questions or items out of the data that they hold. The reason

for why these ”middle man” objects are used is that the data in the database does not completely match the data in questions or items, for an example the database hold the name of the images associated with each of the items but the items do not hold their image names. So if the data from the database were to be cast directly into a item the image name would be lost.

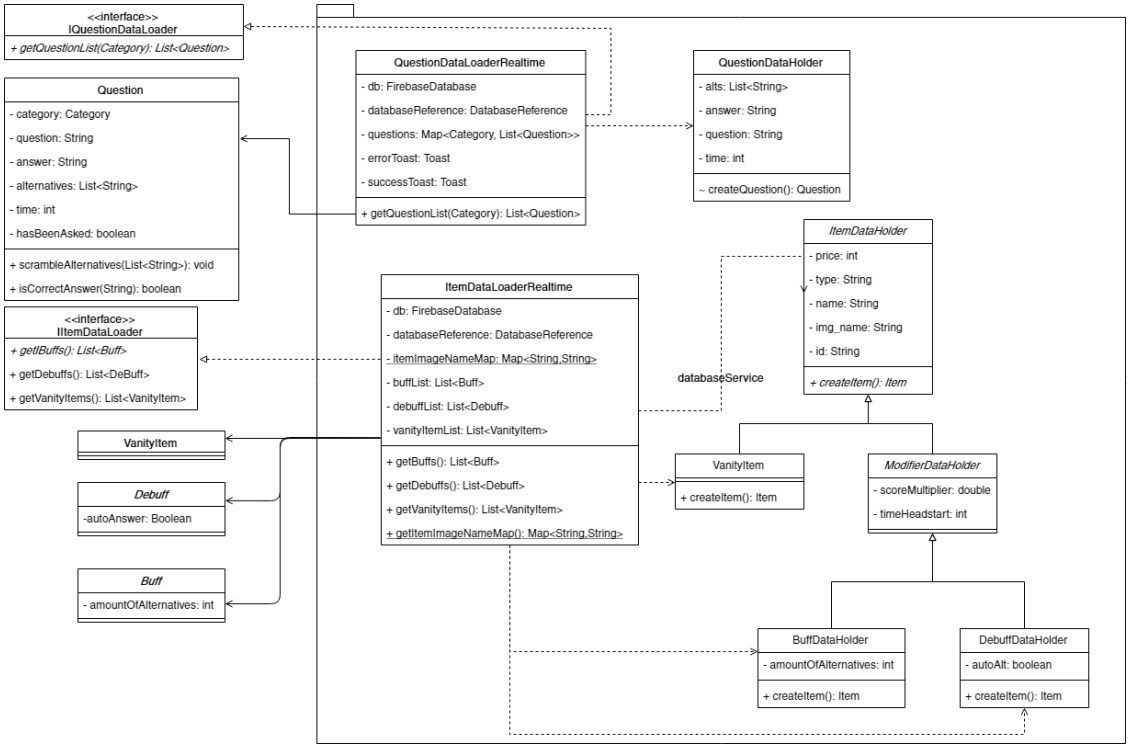

Figure 3: Database service package

3.5 Network manager

The NetworkManager package comprises of mainly two classes, the NetworkModule and the PacketHandler. The NetworkModule cares for the low level Nearby Connection API integration, handling the details relating to the setup, transmission and reception of byte-array payloads to and from other devices. The PacketHandler holds the high level package send and receive logic, responsible for the construction and deconstruction of network packets.

The network protocol consists of a single source of truth, the host, and several clients that may request changes to be made by the host. If the host complies with the request it updates its own model accordingly and broadcasts the resulting model change to all of the connected clients. Thus the host is always in control of the flow of the game,

strictly limiting the cheating capabilities of a malicious client.

The protocol relies on unique identifiers for each of the connection endpoints. The unique id is used by Nearby Connections to distinguish all of the different endpoints, to ensure that a packet is sent to the intended device. Furthermore, the id is also used by the client to determine if a received packet is affecting their local player. If that is the case, then the client updates the view to inform the user that their action has been confirmed by the server and is now part of the new game state.

Due to a limitation within the Nearby Connections API, the local device may never know their own unique identifier. To circumvent this issue, upon every established connection the peers exchanges each others id.

Efforts has been made to provide the repository with a clean interface to work with. Given the intricacy of the communication required to convey all the possible network and model changes, certain complexity still remains. This requires large interfaces for supplying sufficient callback methods to cover all the use cases, resulting in high complexity in the repository. That due to time restrictions could not be refactored.

Potential solutions to reduce the complexity of the repository includes splitting it into two parts, a modelRepository and a new NetRepository. The main improvement comes from extracting the network component to a separate repository. The refactor would improve the readability and separation of concern inside the repository package, but also increase the complexity due to a larger amount of classes.

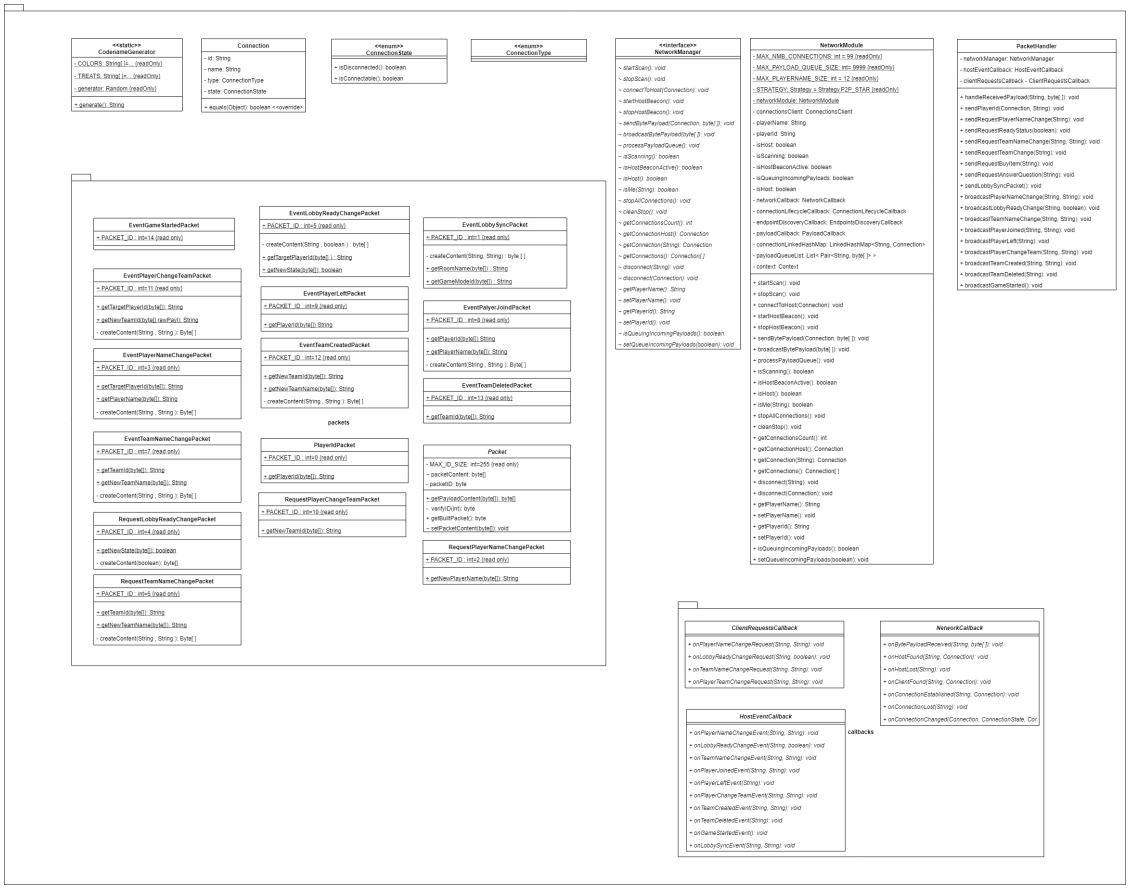

Figure 4: Network module package

3.6 View

The View in the used MVVM-pattern represents the UI of the application just like the more common MVC-pattern. Therefore it should be devoid of any logic. In the application each View consists of a given Activity or Fragment and the different views should only route the user's action to it's specific ViewModel. Each View only observes the given ViewModel for that specific View(within it's lifecycle) using LiveData. LiveData provided many benefits compared to normal observers such that it is lifecycle-aware, it can only be observed in the given context of that Activity's or Fragments components. LiveData therefore prevents memory leaks and it prevents crashes due to activities being destroyed.

The applications different Views inflates it's components from a specific xml-file. The different child-views specific Id's are then used to setup onClick- or onTouchListeners that are then used to notify the ViewModel using normal dependency. This naturally

makes the View dependent on the ViewModel but not the other way around. In some cases the components are dynamically generated instead of specifying them in an xml-file simply because it proved to be easier given that the View varies a lot depending on the amount of players. The LotteryView uses such a solution.

In some instances views do call on the repository directly. This is not optimal but it is mainly related to the networks current implementation and would have been addressed if time was given.

3.7 ViewModel

The ViewModel's responsibility is to manage and prepare data for the View while also handling the communication with the model. In the application the ViewModel is only created with an activity in it's scope and it will persist after that view. When a new View instance is created the viewModel will reconnect with it's owner(view). The use of viewModels in the application reduces the amount of logic in the model that is needed in order to prepare the data for the View the ViewModel can now performs that task.

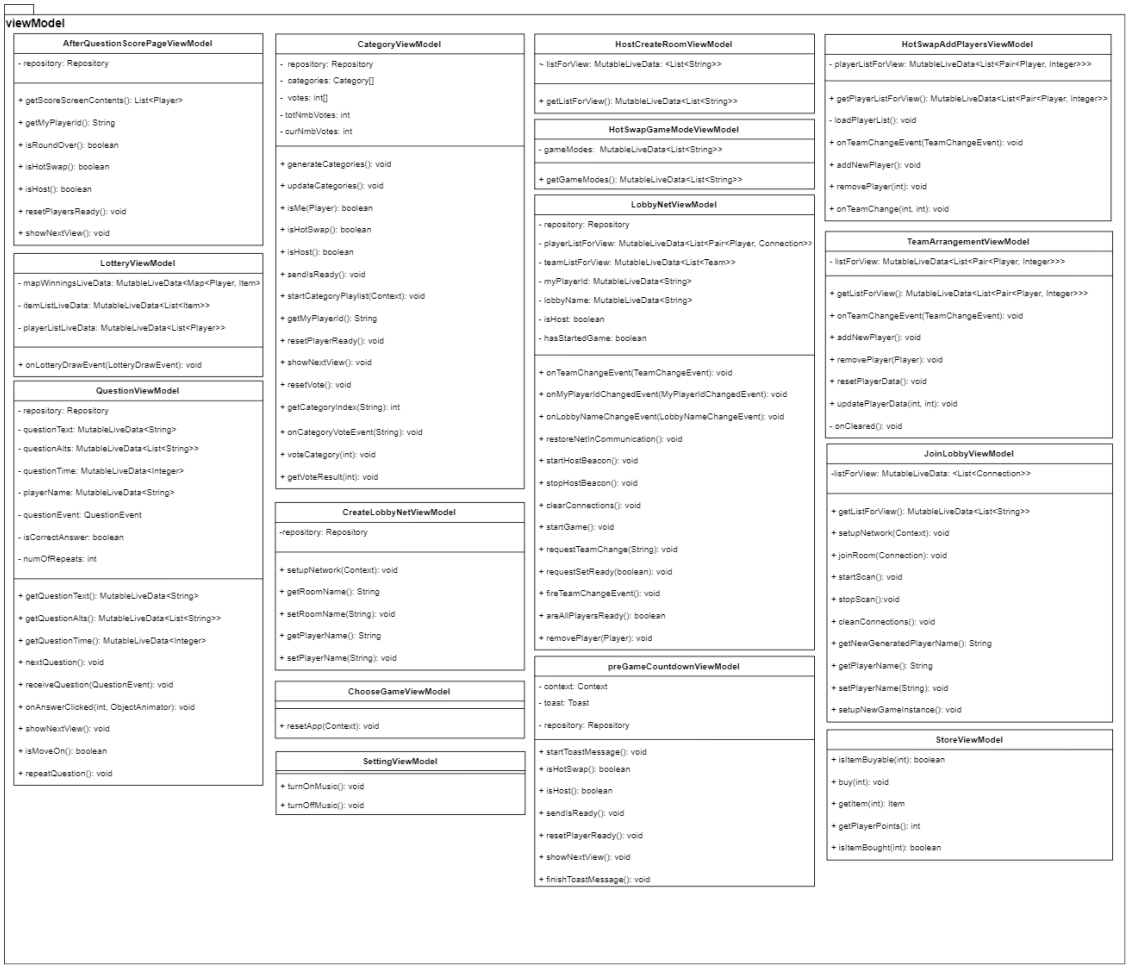

Figure 5: ViewModel package

Create an UML class diagram for every package. One of the packages will contain the model of your application. This will be the design model of your application, describe in detail the relation between your domain model and your design model. There should be a clear and logical relation between the two. Make sure that these models stay in 'sync' during the development of your application.

Describe which (if any) design patterns you have used. The above describes the static design of your application. It may sometimes be necessary to describe the dynamic *design of your application as well. You can use an UML* sequence diagram *to show the different parts of your application communicate an in what order.*

4 Persistent data management

4.1 Question database - Firebase Realtime database

The questions and their related data is stored in a Firebase Realtime database. The database is structured in the following way: All categories are children of the object question and all questions are children of their respective category. Each question has the children question, answer, time, alts and id(see figure 6). Question, answer and id hold string values, the question, answer and id of the question respectively. Time holds a long value, the time that the question will be displayed to the user. And alts holds a list of string objects, the alternative answers to the question.

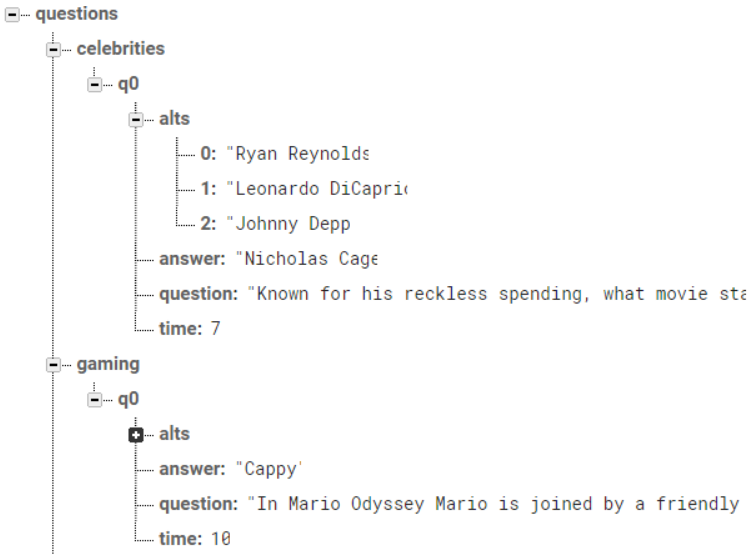

Figure 6: The structure of the question database

When the application launches a event listener is attached that fetches the questions from the database when the program launches and when the data in the database is updated. A backup of the fetched data is also stored on the device so that the application is still functional if it where to lose it's connection to the database.

4.2 Item database

Just like the questions the data needed to instance the different items in the game are also stored inside a Firebase Realtime database. The only real difference between this and the question database is the structure or each of the items. The top level node in the database is item which has the children buff, debuff and vanityItem. These children hold the data for their respective type of items. Each of these items have the children: price which is a int values and type, name and img_name that hold string values. The children of buff and debuff hold the int value time headstart and the

double value scoreMultiplier. The children of buff also have the int values amountOfAlternatives (se figure 7 and the children of debuff hold the boolean value autoAlt. All of theses values except img_name is used in the creation of the different types of items, while img_name is stored in a map with the name of the item as the key and the image name as the value, this is used to connect items to their respective images in the different views.

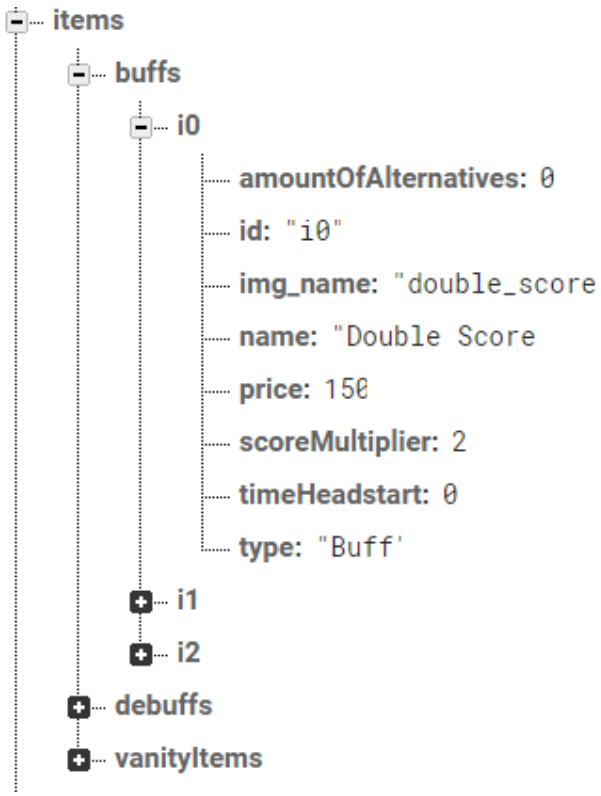

Figure 7: The structure of one buff in the item database

5 Quality

5.1 Testing and Continues integration

The application is tested using both JUnit tests and Android Instrumentality tests. Android Instrumentality tests where used for classes that required context to be able to be tested, such as the data loaders. For the view model classes Mockito was also used to mock certain components to simplify the testing process and remove any unwanted side-effects. The classes that did not require context or need any mocking where tested using JUnit.

All classes in the model except QuizGame were tested with 100% line coverage while QuizGame was tested with 90% line coverage (see figure 8), while only most of the classes in database service and some of the methods in networkManager were tested. The rest of the application remains untested at the present. This is because of a lack of time and the fact that only a small amount of the test were written as their class counterparts were developed. In the case of any future development this should be one of the first things that should be rectified.

modelEvents	100% (4/4)	100% (9/9)	100% (18/18)
Buff	100% (1/1)	100% (2/2)	100% (4/4)
Category	100% (14/14)	100% (44/44)	100% (84/84)
Debuff	100% (1/1)	100% (2/2)	100% (4/4)
GameMode	100% (1/1)	100% (2/2)	100% (4/4)
Item	100% (1/1)	100% (4/4)	100% (8/8)
ItemFactory	100% (1/1)	100% (8/8)	100% (50/50)
Lottery	100% (1/1)	100% (4/4)	100% (12/12)
Modifier	100% (1/1)	100% (3/3)	100% (6/6)
Player	100% (1/1)	100% (23/23)	100% (52/52)
PlayerManager	100% (1/1)	100% (34/34)	100% (168/...
Question	100% (1/1)	100% (8/8)	100% (16/16)
QuestionFactory	100% (1/1)	100% (5/5)	100% (17/17)
QuizGame	100% (1/1)	92% (35/38)	89% (127/1...
Store	100% (1/1)	100% (8/8)	100% (28/28)
Team	100% (1/1)	100% (7/7)	100% (14/14)
VanityItem	100% (1/1)	100% (1/1)	100% (2/2)

Figure 8: The code coverage of the model

Continues integration was also used in the form of Travis. The application was built towards android api 24 and 28. Travis was also used to generate JavaDoc and a git inspector report that was pushed to the projects github page. However there seemed to be some inconsistencies with the git inspector report so it should not be taken as gospel.

5.2 Known issues

Currently the application has the following known issues:

- The player can currently not choose their profile picture.
- Teams can not be renamed.
- There is currently no end to the game.

- Network gameplay does not support lottery functionality.
- Network gameplay does not support store functionality.
- Hotswap lobby allows the add button to be swiped but not completely removed. It should not move at all.
- A game can't be setup into different game modes, the selection is simply a dummy implementation.
- Clients are currently not visibly being notified about them waiting for the server in various situations during gameplay.
- Strings are hard coded instead of using android string resources. Makes future translation work much more tedious.
- In hotswap lobby, the add-player button does not disappear when maximum players are reached.
- Firebase connection is not shut off when the app is sent to background.
- In network game players can not see each others category vote.
- In hotswap mode only one player may vote for a category.
- Weird dependencies between repository and several views caused by how NewViewEvent is populated. Not fixed due to time constraints.

5.3 STAN

The dependency diagram that was generated using STAN only displays the classes that did not use any android classes(9). This is because we could not get android studio to build a jar file with these present. Because of this the diagram may be a bit misleading. Never the less it does display the dependencies between the classes in model and the classes in the network module(10).

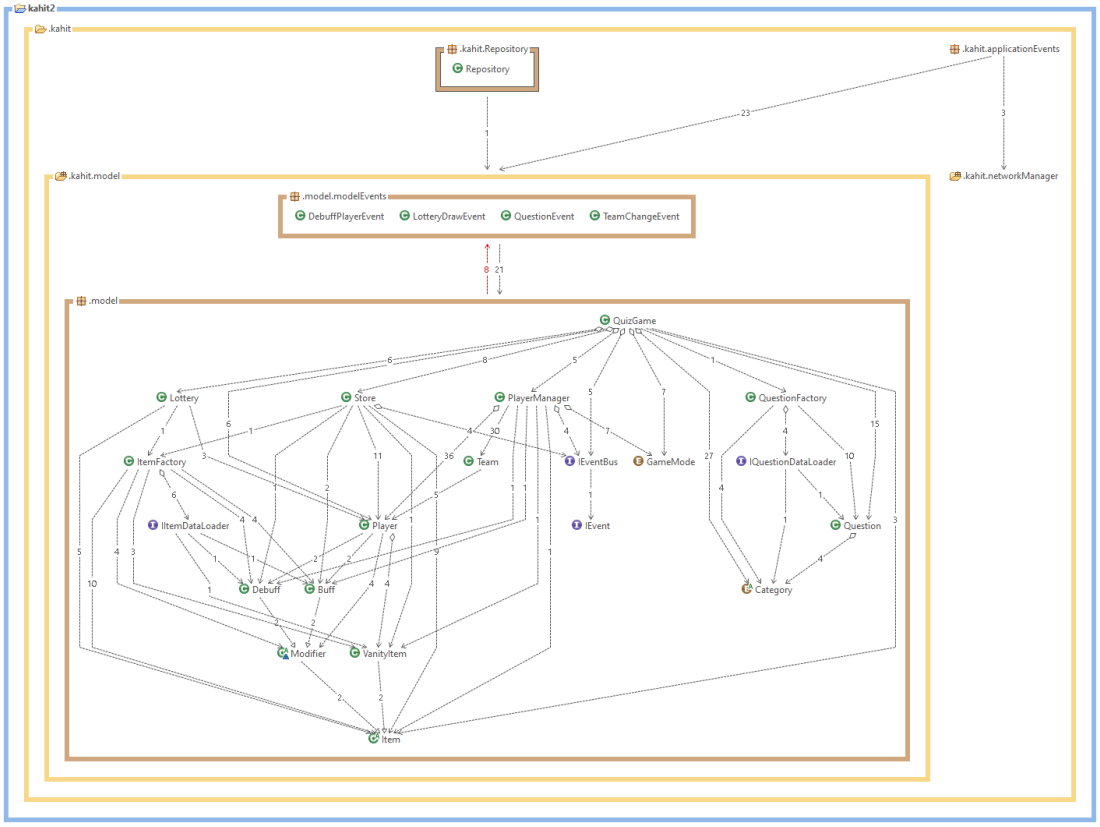

Figure 9: The STAN report of the model.

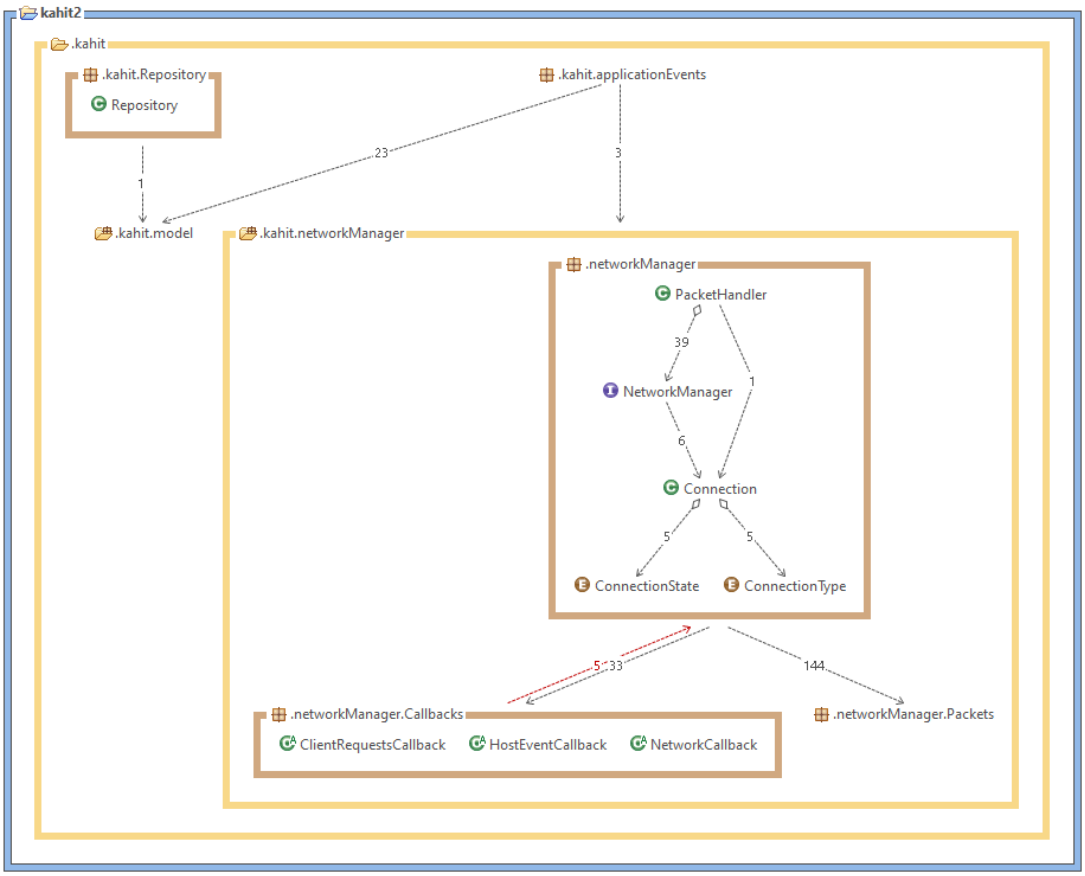

Figure 10: The STAN report of parts of the network module

5.4 Access control and security

NA

6 References

Firebase Documentation: `https://firebase.google.com/docs/database`

Nearby Connection: `https://developers.google.com/nearby/connections/overview`

Green Robot EventBus: `http://greenrobot.org/eventbus/`

Gradle automation tool: `https://docs.gradle.org/current/userguide/what_is_gradle.html`

MVVM structure: https://www.toptal.com/android/android-apps-mvvm-with-clean-architecture

TDA367 Peer-review Grupp 18

Working Group: (Grupp 19)
Anas Alkoutli
Oussama Anadani
Mats Cedervall
Johan Ek
Jakob Ewerstrand

oktober 2019

Contents

1 Introduktion

Vi har valt att dela upp vår feedback i två listor; en bra och en mindre bra. Förhoppningsvis leder detta till en enklare läsning för er.

2 Feedback

2.1 Bra

- Strukturen med många paket är lätt för ögat och det underlättar att söka rätt på klasser.
- (Design pattern - modular tabs) Applikationen använder sig utav "Modular tabs" i form av fragments för ad, browse samt profil vyerna.
- (Design pattern - same page error messages) vid felaktig inloggning så återfinns designmönstret.
- (Design pattern - input prompt) Finns i varje textfält som kan fyllas i.
- (Design pattern - Observer) IOnItemClicked är ett interface som används som ett "onClick" observer pattern. En snygg lösning för att lösa kommunikationen mellan viewmodeln och viewn.
- (Design principle - MVC) Androids MVVM är ett passande mönster i sammanhanget. MVVM uppfyller samma princip som MVC med en separat "Controller" i form av en viewModel.
- (Design principle - Separation of concern) varje ansvarsområde är uppdelat i paket.
- Följer android plattformens designsnormer. NavGraph, fragments etc

2.2 Mindre bra

- LiveData i database (som ligger i model) leder till att modellen inte kan flyttas till en annan miljö utan att behöva modifiera den. Dock har ni kanske redan ändrat detta, framkommer inte i UML:en.
- Create knappen syns inte när man ska skapa en ad.
- Det finns en image URL i Product bör den vara där?
- I domänmodellen finns det ingen pil mellan User och Comments men i designmodellen finns det.
- I domänmodellen står Ad-klassen i plural.
- I domänmodellen måste det inte stå "0,n" istället för bara "n"?

- I designmodellen så borde klasserna använda sig av versaler på första bokstaven.
- I designmodellen finns det två cirkulära beroenden. Först en mellan User, Ad och Comment och sedan ännu en mellan Review, User och Ad.
- Enum med små bokstäver i Category det borde vara enbart versaler.
- Alla repositories bör kanske ligga i ett gemensamt package-repository?
- Verkar som de flesta av testerna inte fungerar i nuläget.
- Price i Account?
- IOnItemClicked finns det tre stycken av varav en skiljer sig i avseende på antalet parametrar i sin enda metod. Frågan är om det hade varit möjligt att ha ett gemensamt IOnItemClick interface?
- Access-modifiers, utnyttja package-private
- Attribut inuti te.x. CommentsFragment kan istället tas som lokala variabler.
- Kallar på en statisk-klass attributes genom en instans vilket är bad practise, CreateAd - onActivityResult.
- I den nya domänmodellen finns det ingen koppling mellan Comment/Review och User. Vilket känns lite konstigt då det inte finns något självklart sätt att hämta alla kommentarer/resencioner som en användare har skrivit.
- CreateAd-fragmentet skapar ett ad-objekt vilket bryter mot MVVM. Det gör denna view onödigt tjock, borde ske i respektive viewmodel.
- Databasen ska ligga i ett eget paket.
- För många oanvända imports

3 Informell Punktlista

- *Do the design and implementation follow design principles?*
  - Ett antal, se avsnitt 2.1.
- *Does the project use a consistent coding style?*
  - Ja alla namn använder liknande stil. Projektet använder som av en återkommande struktur.
- *Is the code reusable?*
  - LiveData i modellen, inte bra. Finns ett duplicerat Interface, till synes enbart för att dem ligger i två olika packages.
- *Is it easy to maintain?*
  - Genom att koden är väl separerad utefter ansvarsområde, ger det koden en låg coupling och hög cohesion. Något som är erkänt bra för maintainability.
- *Can we easily add/remove functionality?*
  - Då separation of concern används flitigt, förenklar detta modifikationer.
- *Are design patterns used?*
  - Flera stycken, se avsnitt 2.1.
- *Is the code documented?*
  - Varierat, många klasser väldokumenterade medan vissa inte alls.
- *Are proper names used?*
  - Gemener i enum. Inconsistency i interface-namn.
- *Is the code well tested?*
  - Model är vältestad, medan applikations-delen saknar tester helt och hållet.
- *Are there any security problems, are there any performance issues?*
  - Kan ändra på konto-lösenord utan att bekräfta med det gamla, tillåter utomstående att ändra ditt lösenord.
- *Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?*
  - Namngivningen är lättförståelig dock saknas Javadoc för en del klasser/meotder. Klasserna är också av rimlig storlek.
- *Can the design or code be improved? Are there better solutions?*
  - Circular dependency, se ovan.