# System Design Document for KahIt

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

11 oktober 2019
Version 0.2

## 1 Introduction

*KahIt* is a fun and entertaining quiz-app for everyone that can elevate a party to the next level. *KahIt* is an app designed to make parties more fun by making people play and interact with each other. When hosting a game friends can connect using nearby connection with up to 8 players. There is a mode where players can play as a team against other teams, or even a hot swap mode where player can play on one device only. *KahIt* makes it more fun to play against friends since it has items that boost players scores or reduce it. *KahIt* provide a verity of cosmetic items that do not affect players status but are used to show off.

### 1.1 Definitions, acronyms, and abbreviations

todo
Definitions etc. probably same as in RAD

# 2 System architecture

todo

*The most overall, top level description of your application. If your application uses multiple components (such as servers, databases, etc.), describe their responsibilities here and show how they are dependent on each other and how they communicate (which protocols etc.)*

*You will to describe the 'flow' of the application at a high level. What happens if the application is started (and later stopped) and what the normal flow of operation is. Relate this to the different components (if any) in your application.*

# 3 System design

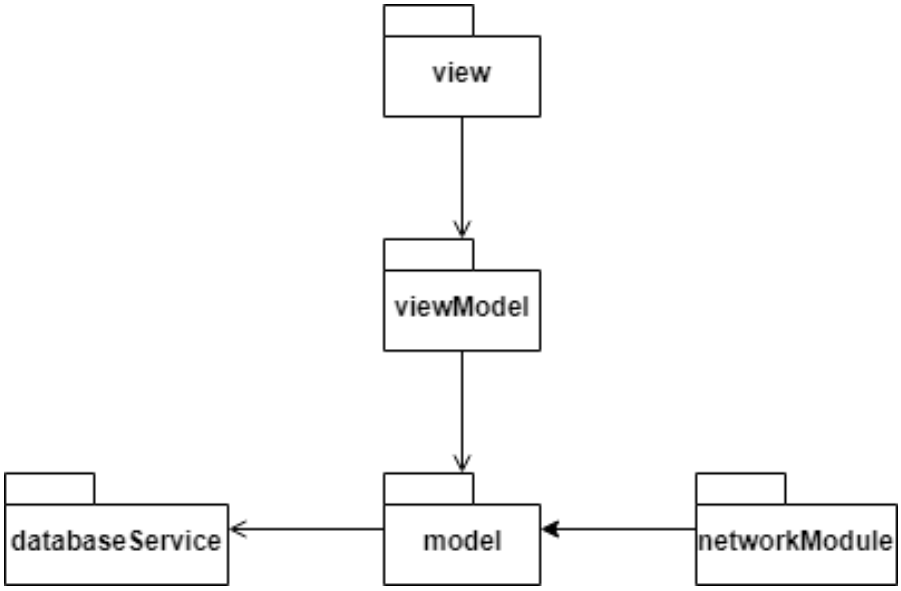



Figure 1: High-level package view

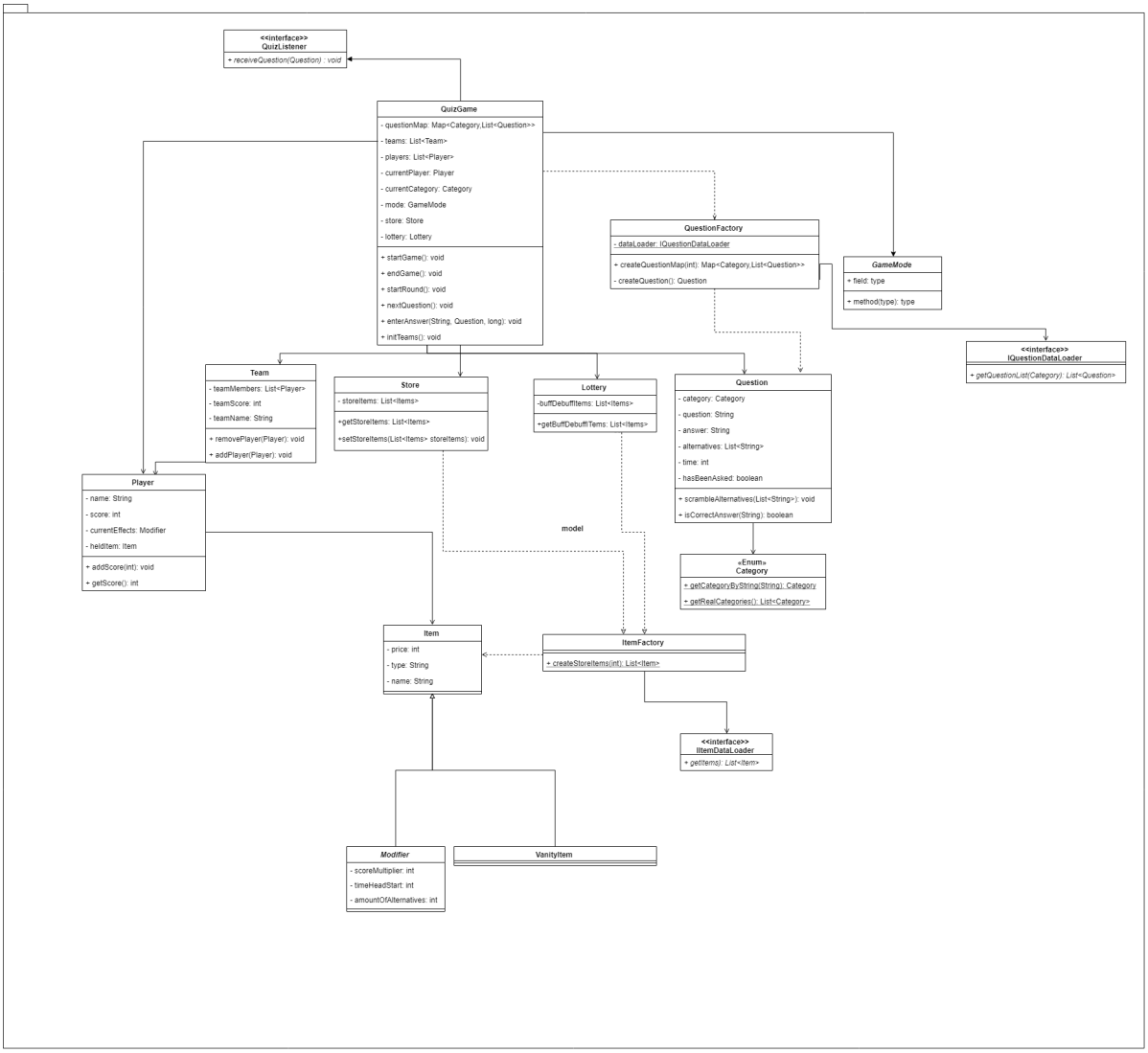



Figure 2: Design model

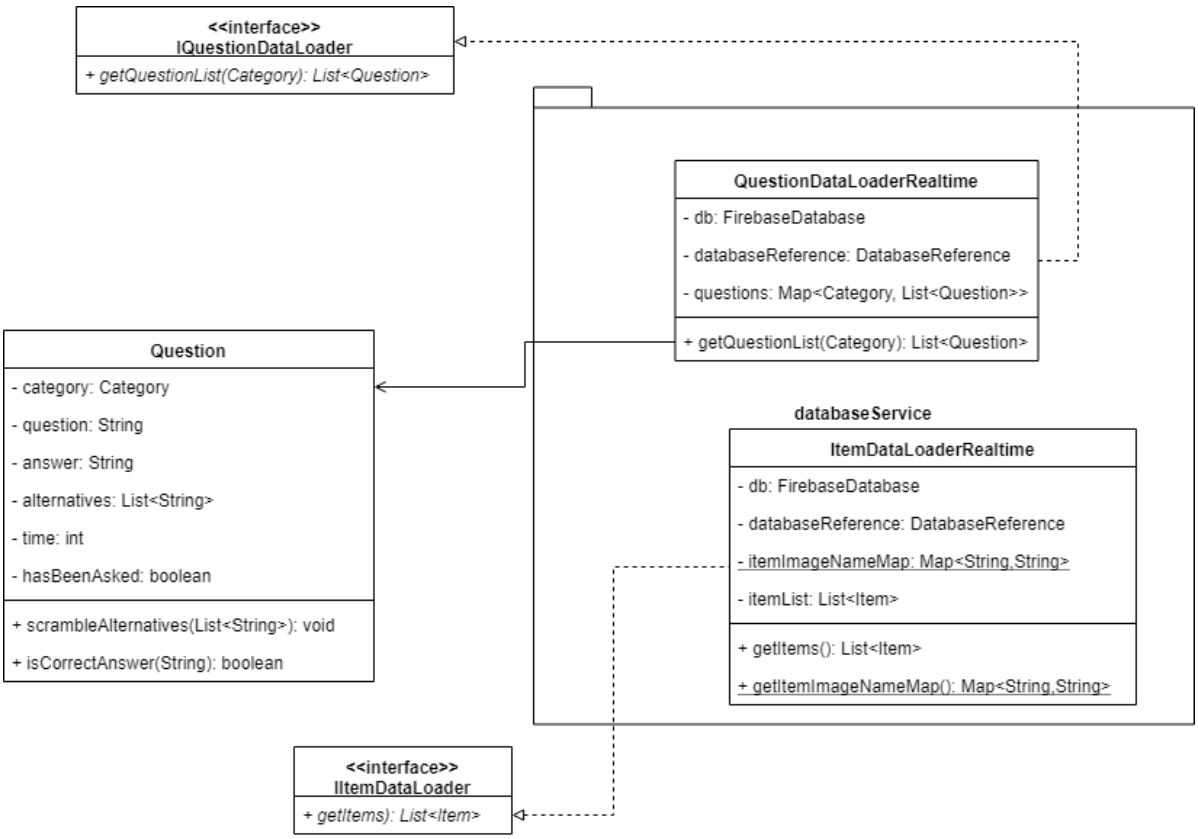



Figure 3: Database service package

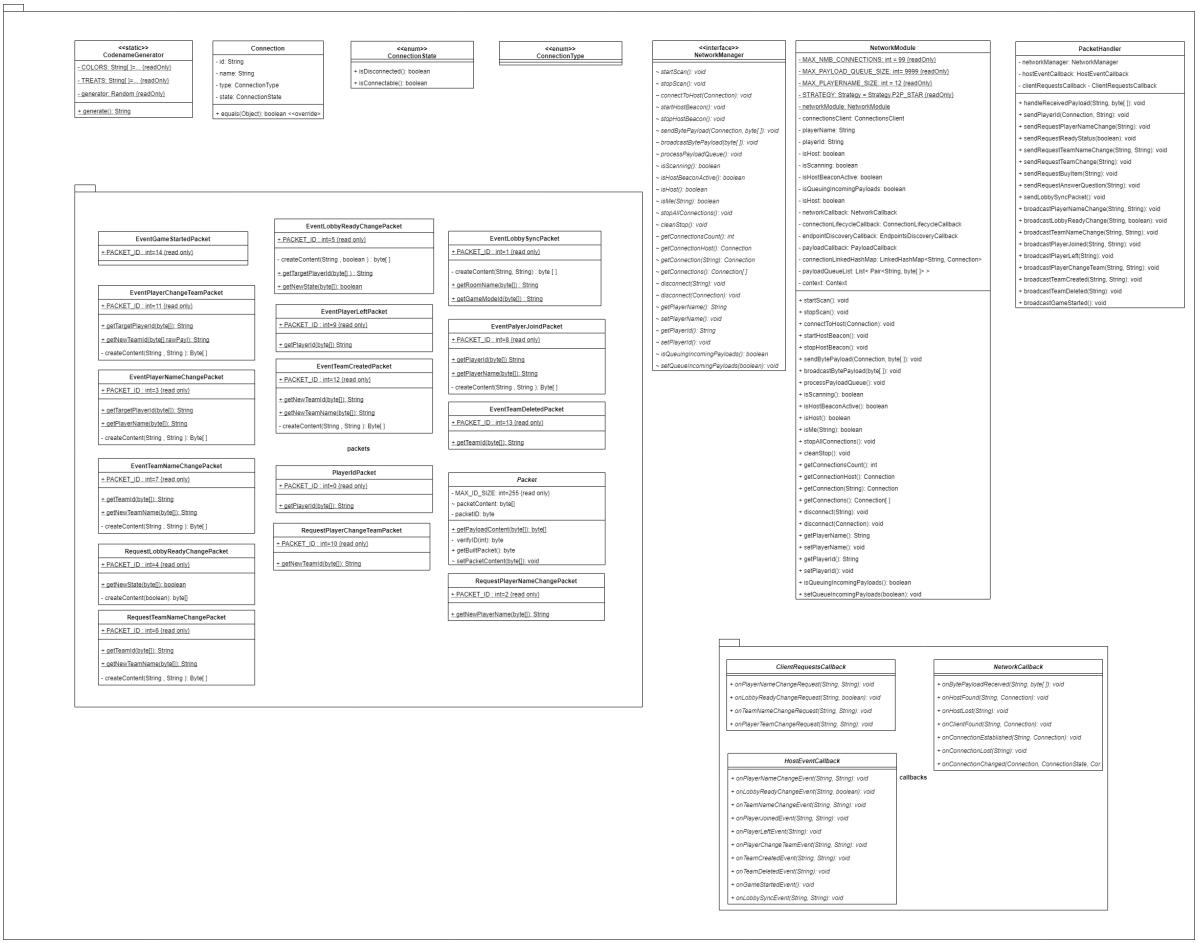
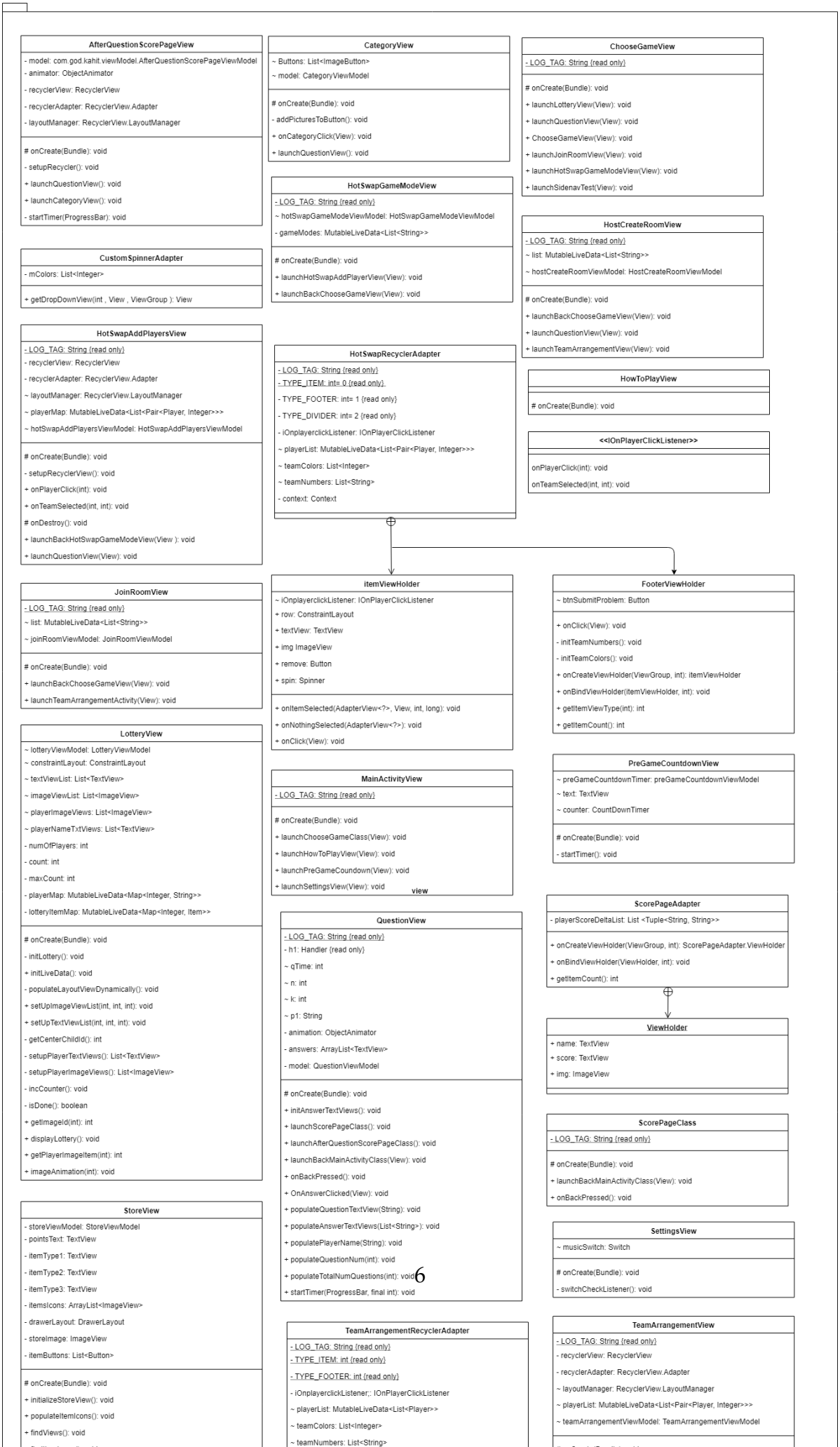



Figure 4: Network module package


AfterQuestionScorePageView
- model: com.god.kahit.viewModel.AfterQuestionScorePageViewModel
- animator: ObjectAnimator
- recyclerView: RecyclerView
- recyclerAdapter: RecyclerView.Adapter
- layoutManager: RecyclerView.LayoutManager
# onCreate(Bundle): void
- setupRecycler(): void
+ launchQuestionView(): void
+ launchCategoryView(): void
- startTimer(ProgressBar): void
CategoryView
~ Buttons: List<ImageButton>
~ model: CategoryViewModel
# onCreate(Bundle): void
- addPicturesToButton(): void
+ onCategoryClick(View): void
+ launchQuestionView(): void
ChooseGameView
- LOG_TAG: String {read only}
# onCreate(Bundle): void
+ launchLotteryView(View): void
+ launchQuestionView(View): void
+ ChooseGameView(View): void
+ launchJoinRoomView(View): void
+ launchHotSwapGameModeView(View): void
+ launchSidenavTest(View): void
CustomSpinnerAdapter
- mColors: List<Integer>
+ getDropDownView(int , View , ViewGroup ): View
HotSwapGameModeView
- LOG_TAG: String {read only}
~ hotSwapGameModeViewModel: HotSwapGameModeViewModel
- gameModes: MutableLiveData<List<String>>
# onCreate(Bundle): void
+ launchHotSwapAddPlayerView(View): void
+ launchBackChooseGameView(View): void
HostCreateRoomView
- LOG_TAG: String {read only}
~ list: MutableLiveData<List<String>>
~ hostCreateRoomViewModel: HostCreateRoomViewModel
# onCreate(Bundle): void
+ launchBackChooseGameView(View): void
+ launchQuestionView(View): void
+ launchTeamArrangementView(View): void
HotSwapAddPlayersView
- LOG_TAG: String {read only}
- recyclerView: RecyclerView
- recyclerAdapter: RecyclerView.Adapter
~ layoutManager: RecyclerView.LayoutManager
~ playerMap: MutableLiveData<List<Pair<Player, Integer>>>
~ hotSwapAddPlayersViewModel: HotSwapAddPlayersViewModel
# onCreate(Bundle): void
- setupRecyclerView(): void
+ onPlayerClick(int): void
+ onTeamSelected(int, int): void
# onDestroy(): void
+ launchBackHotSwapGameModeView(View ): void
+ launchQuestionView(View): void
HotSwapRecyclerAdapter
- LOG_TAG: String {read only}
- TYPE_ITEM: int= 0 {read only}
- TYPE_FOOTER: int= 1 {read only}
- TYPE_DIVIDER: int= 2 {read only}
- iOnplayerclickListener: IOnPlayerClickListener
~ playerList: MutableLiveData<List<Pair<Player, Integer>>>
~ teamColors: List<Integer>
~ teamNumbers: List<String>
- context: Context
HowToPlayView
# onCreate(Bundle): void
<<IOnPlayerClickListener>>
onPlayerClick(int): void
onTeamSelected(int, int): void
JoinRoomView
- LOG_TAG: String {read only}
~ list: MutableLiveData<List<String>>
~ joinRoomViewModel: JoinRoomViewModel
# onCreate(Bundle): void
+ launchBackChooseGameView(View): void
+ launchTeamArrangementActivity(View): void
itemViewHolder
~ iOnplayerclickListener: IOnPlayerClickListener
+ row: ConstraintLayout
+ textView: TextView
+ img ImageView
+ remove: Button
+ spin: Spinner
+ onItemSelected(AdapterView<?>, View, int, long): void
+ onNothingSelected(AdapterView<?>): void
+ onClick(View): void
FooterViewHolder
~ btnSubmitProblem: Button
+ onClick(View): void
- initTeamNumbers(): void
- initTeamColors(): void
+ onCreateViewHolder(ViewGroup, int): itemViewHolder
+ onBindViewHolder(itemViewHolder, int): void
+ getItemViewType(int): int
+ getItemCount(): int
LotteryView
~ lotteryViewModel: LotteryViewModel
~ constraintLayout: ConstraintLayout
~ textViewList: List<TextView>
~ imageViewList: List<ImageView>
~ playerImageViews: List<ImageView>
~ playerNameTxtViews: List<TextView>
- numOfPlayers: int
- count: int
- maxCount: int
- playerMap: MutableLiveData<Map<Integer, String>>
- lotteryItemMap: MutableLiveData<Map<Integer, Item>>
# onCreate(Bundle): void
- initLottery(): void
+ initLiveData(): void
- populateLayoutViewDynamically(): void
+ setUpImageViewList(int, int, int): void
+ setUpTextViewList(int, int, int): void
- getCenterChildId(): int
- setupPlayerTextViews(): List<TextView>
- setupPlayerImageViews(): List<ImageView>
- incCounter(): void
- isDone(): boolean
+ getImageId(int): int
+ displayLottery(): void
+ getPlayerImageItem(int): int
+ imageAnimation(int): void
MainActivityView
- LOG_TAG: String {read only}
# onCreate(Bundle): void
+ launchChooseGameClass(View): void
+ launchHowToPlayView(View): void
+ launchPreGameCoundown(View): void
+ launchSettingsView(View): void
view
PreGameCountdownView
~ preGameCountdownTimer: preGameCountdownViewModel
~ text: TextView
~ counter: CountDownTimer
# onCreate(Bundle): void
- startTimer(): void
QuestionView
- LOG_TAG: String {read only}
- h1: Handler {read only}
~ qTime: int
~ n: int
~ k: int
~ p1: String
- animation: ObjectAnimator
- answers: ArrayList<TextView>
- model: QuestionViewModel
# onCreate(Bundle): void
+ initAnswerTextViews(): void
+ launchScorePageClass(): void
+ launchAfterQuestionScorePageClass(): void
+ launchBackMainActivityClass(View): void
+ onBackPressed(): void
+ OnAnswerClicked(View): void
+ populateQuestionTextView(String): void
+ populateAnswerTextViews(List<String>): void
+ populatePlayerName(String): void
+ populateQuestionNum(int): void
+ populateTotalNumQuestions(int): void
+ startTimer(ProgressBar, final int): void
ScorePageAdapter
- playerScoreDeltaList: List <Tuple<String, String>>
+ onCreateViewHolder(ViewGroup, int): ScorePageAdapter.ViewHolder
+ onBindViewHolder(ViewHolder, int): void
+ getItemCount(): int
ViewHolder
+ name: TextView
+ score: TextView
+ img: ImageView
ScorePageClass
- LOG_TAG: String {read only}
# onCreate(Bundle): void
+ launchBackMainActivityClass(View): void
+ onBackPressed(): void
StoreView
- storeViewModel: StoreViewModel
- pointsText: TextView
- itemType1: TextView
- itemType2: TextView
- itemType3: TextView
- itemsIcons: ArrayList<ImageView>
- drawerLayout: DrawerLayout
- storeImage: ImageView
- itemButtons: List<Button>
# onCreate(Bundle): void
+ initializeStoreView(): void
+ populateItemIcons(): void
+ findViews(): void
SettingsView
~ musicSwitch: Switch
# onCreate(Bundle): void
- switchCheckListener(): void
TeamArrangementRecyclerAdapter
- LOG_TAG: String {read only}
- TYPE_ITEM: int {read only}
- TYPE_FOOTER: int {read only}
- iOnplayerclickListener;: IOnPlayerClickListener
~ playerList: MutableLiveData<List<Player>>
~ teamColors: List<Integer>
~ teamNumbers: List<String>
TeamArrangementView
- LOG_TAG: String {read only}
- recyclerView: RecyclerView
- recyclerAdapter: RecyclerView.Adapter
~ layoutManager: RecyclerView.LayoutManager
~ playerList: MutableLiveData<List<Pair<Player, Integer>>>
~ teamArrangementViewModel: TeamArrangementViewModel

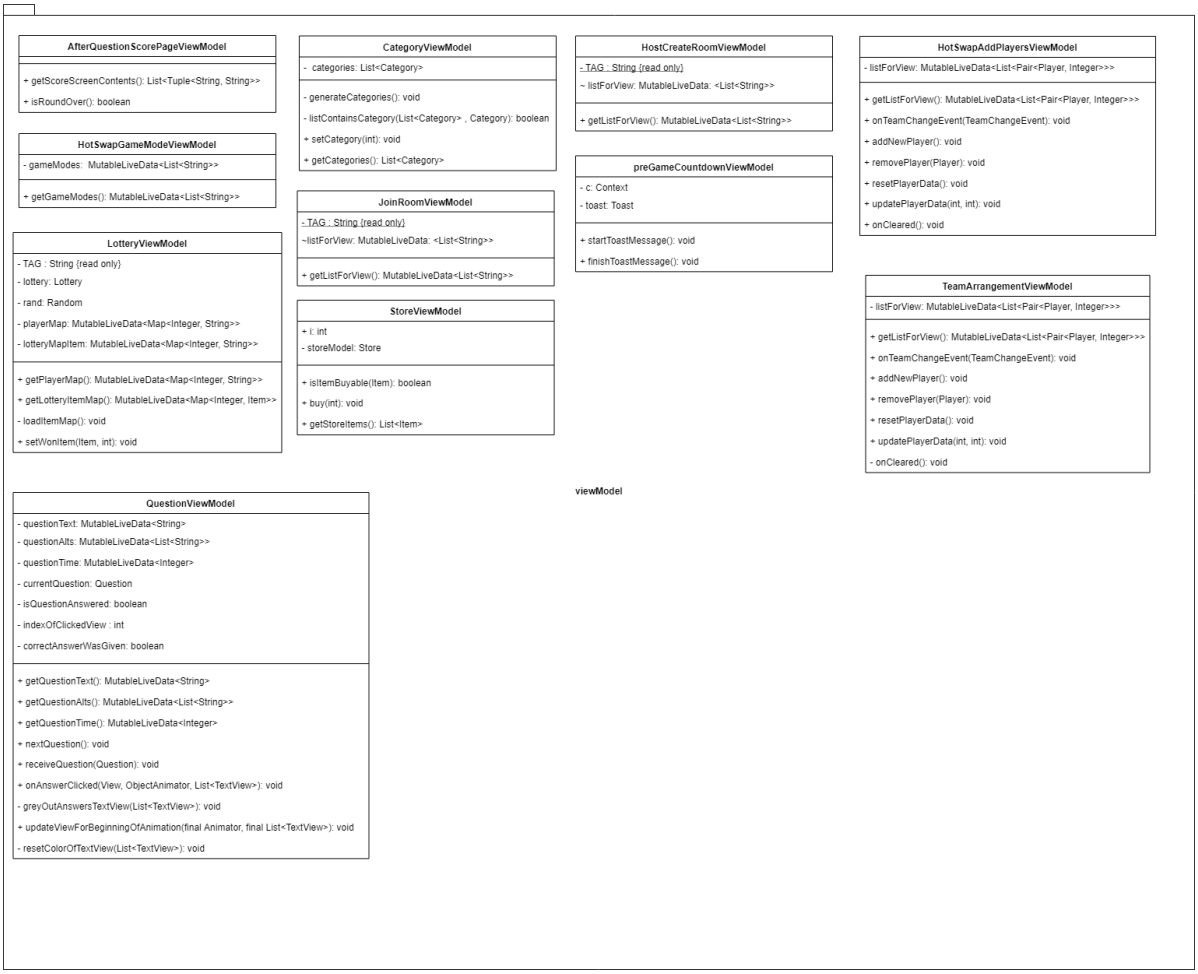



Figure 6: ViewModel package

todo

*Draw an UML package diagram for the top level for all components that you have identified above (which can be just one if you develop a standalone application). Describe the interfaces and dependencies between the packages. Describe how you have implemented the MVC design pattern.*

*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 and alts (see figure 7). Question and answer hold string values, the question and the answer 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.

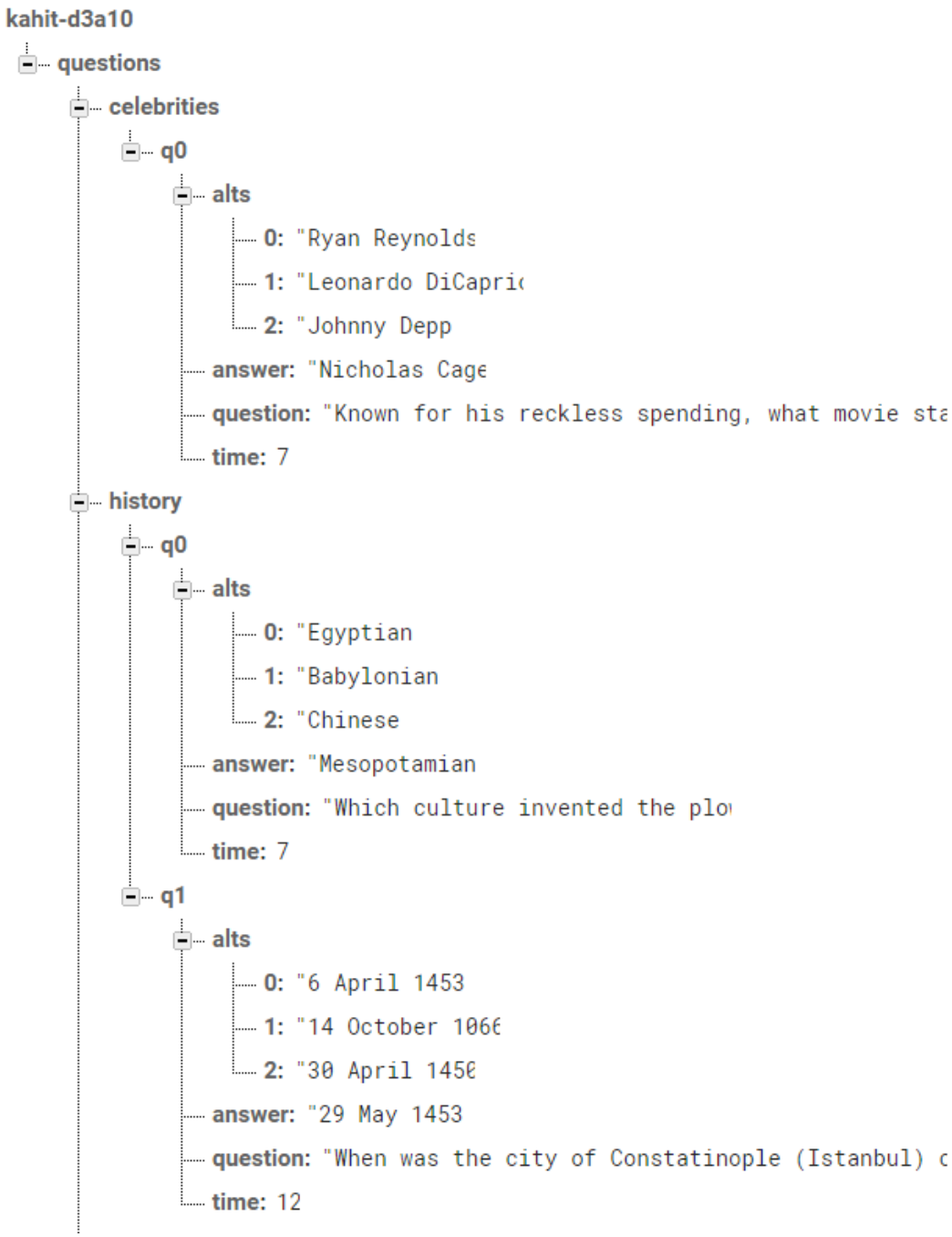



Figure 7: The structure of the database

When the application launches all questions are fetched from the database and an event listener is attached. If the database is updated the event listener fetches the new data so that list of questions in the model stays up to date. 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. Each item (which is child of items) has the children: price, scoreMultiplier, timeHeadstart and amountOfAlternatives that hold int values and type, name and img_name that hold string values (see figure8). All of theses values exept img_name is used in the creation of items, while img_name is to connect items to their respective images in the view.

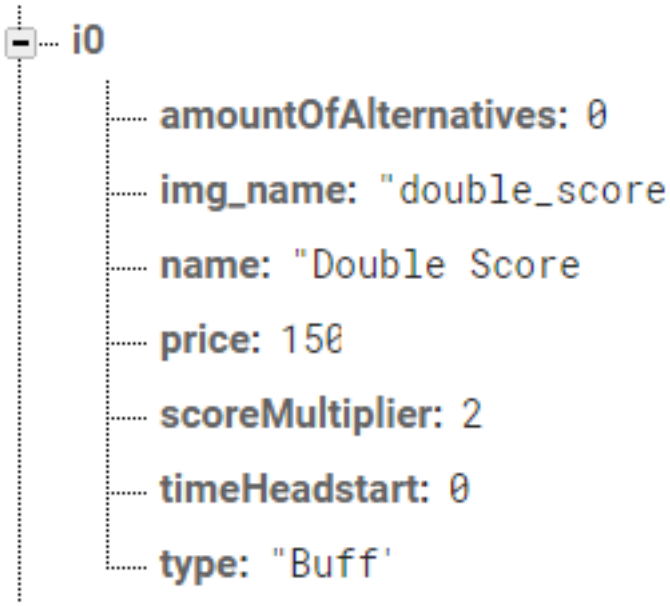



Figure 8: The structure of one item in the database

## 5 Quality

- Describe how you test your application and where to find these tests. If applicable, give a link to your continuous integration.
- List all known issues.
- Run analytical tools on your software and show the results. Use for example:
  - Dependencies: STAN or similar.
  - Quality tool reports, like PMD.

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.

### 5.1 Access control and security

todo

*If you applications has some kind of access control, for example a login, of has different user roles (admin, standard, etc.), then explain how you application manages this.*

## 6 References

todo

*List all references to external tools, platforms, libraries, papers, etc. The purpose is that the reader can find additional information quickly and use this to understand how your application works.*