VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
INFORMATION TECHNOLOGIES STUDY PROGRAM

Problem-Based Project

Mind Game Application

Done by:
Bohdan Borysei
Andrej Jegorov
Vidmantas Moisevičius
Aras Urbonas

Supervisor:
dr. Agnė Brilingaitė

Vilnius
2021

Contents

Abstract

The goal of this work is to develop a client-server type quiz application for android mobile devices. The main functionality of the system is centered around providing the user with an interactive interface to answer questions. The user is given the ability to choose between different quiz topics, housed in categories. The applications purpose is to provide more personalization and gamification to the quiz genre, in turn, by adding a "host" system in resemblance to quiz based television shows, providing an in-game shop, user analytics and the ability to have questions in various forms of media. An integrated custom topic creator is also provided to the player. Additionally, the implementation of categories, topics, questions and hosts is heavily focused on modularity, making new additions easier. The ability to import new questions is made via the use of a web server, which stores media files and acts as an intermediate layer between the Android app and the database. The relational database is used for storing questions in a structured way that allows grouping by topic and category. It also enables processing of user statistics, that are later shown to the user. Related applications have also been analyzed and their advantages and disadvantages are presented in comparison to the *"MindGame"* quiz application.

Keywords: Android Application, Interactive Quiz, User Analytics, Modular Structure, Media Support

Santrauka

MindGame programelė

"Abstract" section in Lithuanian is empty for now.

1 Preliminary design

This section discusses the preliminary design (*figure 1*) of the application concerning the user. The design shown in *figure 1* is by no means a direct recreation of the User Interface of the application but rather the representation of the interactivity that is supposed to be provided by the system to the user.

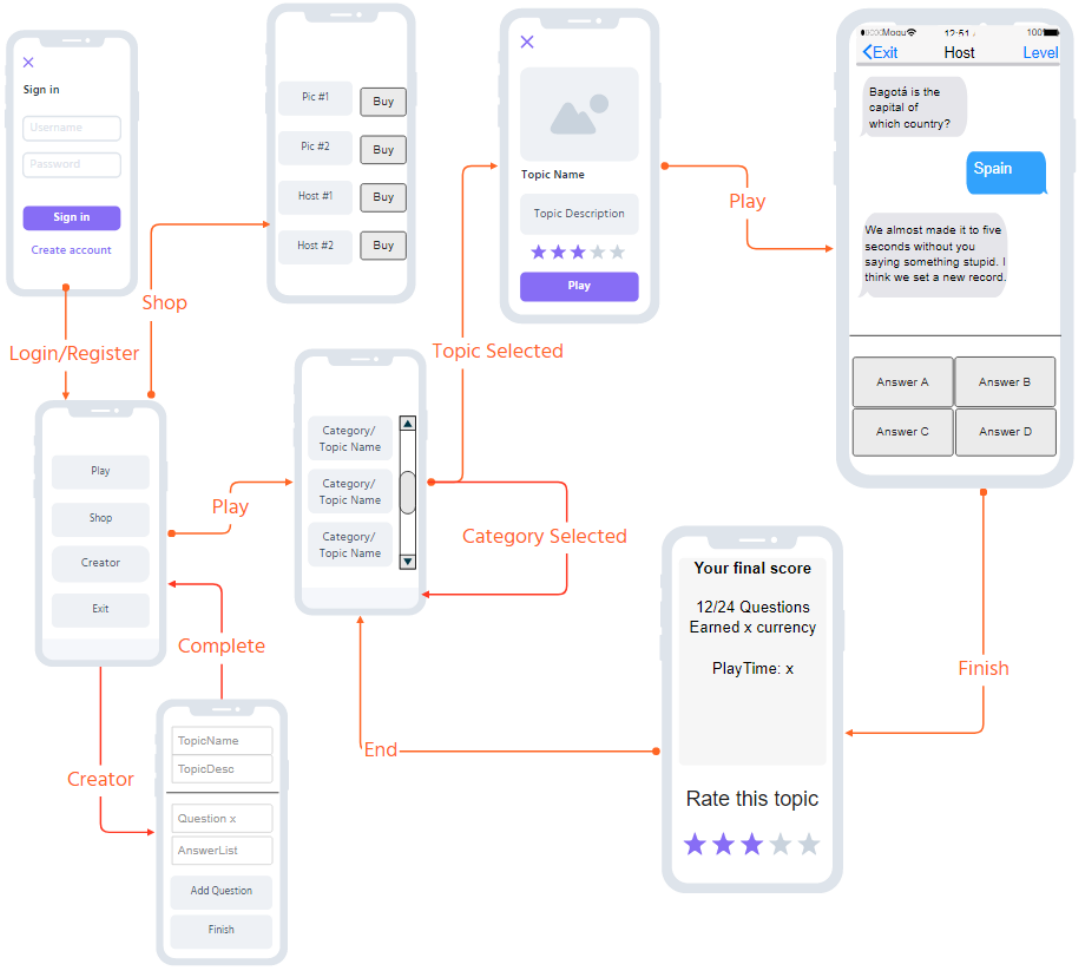

Figure 1. Preliminary design wireframe of the Mind Game application

1.1 Menu

Upon opening the application, the user shall be presented with a login screen. Following a successful login, the user shall be presented with several options including “PLAY”, “SHOP”, “CREATOR” and “EXIT” as presented in the bottom left of figure *figure 1*. The listed options will be explained in further detail in the following sections of the report with the exclusion of the self-explanatory “EXIT” option. If the user happens to not have a registered account upon launch of the application he may create one by clicking on the “Don’t have an account?” field and inputting the required user information.

1.2 Category/Topic selection

In order to play a quiz, the user must first select which topic they would like to play. Each selected topic has its own name, category, description, related image, author and rating (*figure 1*). Searching through the topics the user will first be required to select a specific category from a list that they want to play, upon selection of a category the user will be presented with all topics that are related to the selected category. Upon selecting of the desired topic, the user will be able to finally go to the game interface and play the quiz.

1.3 Game Interface

With an intention of providing a different experience than most quiz applications of similar notion the game interface of the application shall be rather unorthodox resembling a mixture of texting programs and a classic "A, B, C, D" type quizzes (*figure 2*). The questions shall be delivered by a bot host (*see section 1.4*) in a format similar to messages in a regular texting application. The user will be limited to 4 responses to each question, each of which represent an answer to the question given by the host. Upon a choice of a specific answer the user will be informed by the host whether they answered correctly or not and move on to the next question. This shall continue until all of the level's questions have been exhausted which in turn will prompt a result screen to pop up.

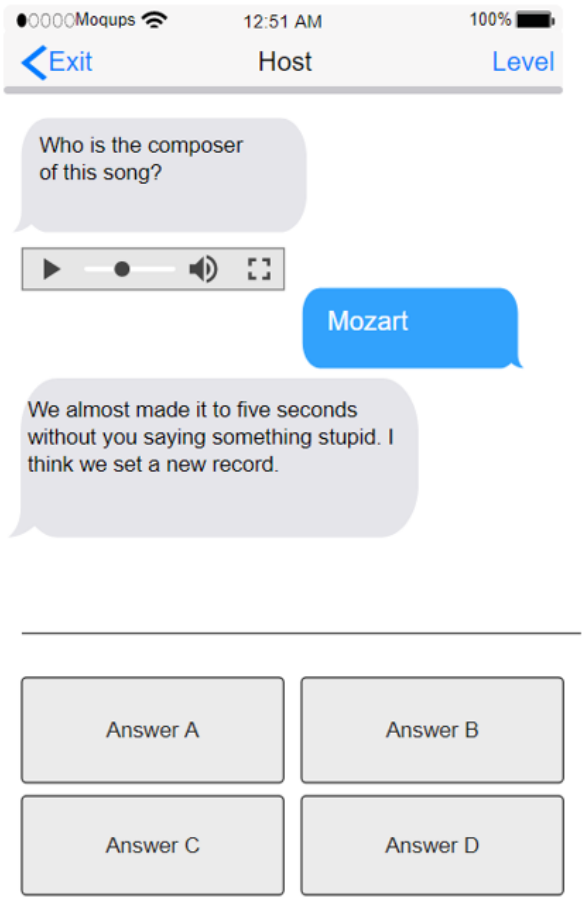

Figure 2. Game interface of the Mind Game application

1.4 Host

Considering the fact that one of the main goals of the application is to partly mimic the functionality that television game shows of similar quiz format possess, the application shall feature a host. The host's main purpose is to deliver the questions and react based on the players performance when answering the given questions. The host shall have different sets of responses by reacting on correct answers, incorrect answers and by posting arbitrary phrases which will be randomly selected during the duration of the quiz level playthrough (in addition to other types of messages). Thus, the application's host functions as an intermediary between the quiz itself and the user in an attempt to provide a more interesting and engaging experience (in a similar manner to the television game shows).

Different personalities for the host are planned, purchasable at the in-game shop (*see section 1.6*) with the starting personality for the "host" being sarcastic (interaction example provided in *figure 2*).

No complex capabilities such as learning from previous encounters are complete for the aforementioned reaction system of the host, all of the possible sets of correct answer, incorrect answer and random phrase responses are pre-written and specific choice of response in each of those three reaction categories is determined by chance.

1.5 Results

Upon the completion of the quiz the user will be presented with a congratulations screen which shall provide the number of correct answers scored in the quiz topic, the elapsed time as well as the ability to rate the completed played topic (*figure 1*). The ability for a user to rate levels will help immensely with sorting out proper and poorly made topics, which will be especially useful after the implementation of the topic creator (*see section 1.7*).

1.6 Shop

In order to provide the feeling of progression to the user an in-game shop will be provided. The items in the shop shall be purchasable with in game currency earned by completing quiz topics. The currently planned features for the shop include:

Profile Pictures - ability to purchase new types of profile pictures for user customization purposes with the coins earned from completing quizzes.

Host Personalities - provide different sets of phrases for correct answer, incorrect answer and random reactions used by the host as discussed in *section 1.4*.

1.7 Creator

The application also features an option for users to create their own quiz topics and allow other users to play them. The design of which is rather simple - the user will be presented with a screen prompting to enter the to be created topic's name and description, after that the user will be presented with a screen to add new questions to that topic. After it is complete and the user finishes the quiz creation process, the created topic shall be sent to and stored in the database as long as all details have been inputted correctly. It is important to note that all topics created by users shall be associated with the "Custom" topic category. Image and audio upload will not be available to an average user due to moderation reasons.

2 System architecture

2.1 High-level Overview

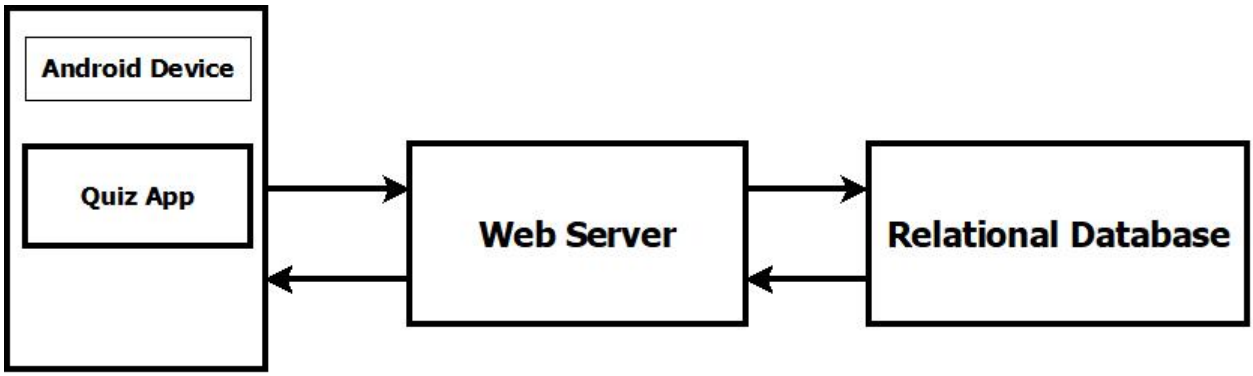

Figure 3. High-level overview of the Mind Game application

The client side Android app is being made for Android mobile devices using the Kotlin programming language.

The database is needed to store question sets, user statistics and other content. For this PostgreSQL - an open-source relational database management system is being used.

The web server is used to connect the database and the client side Android app. Currently PostgREST is used for the web server, which is running an API that serves HTTP requests by fetching data from the database and sends it back in JSON format.

2.2 Front-end

2.2.1 Android Activities

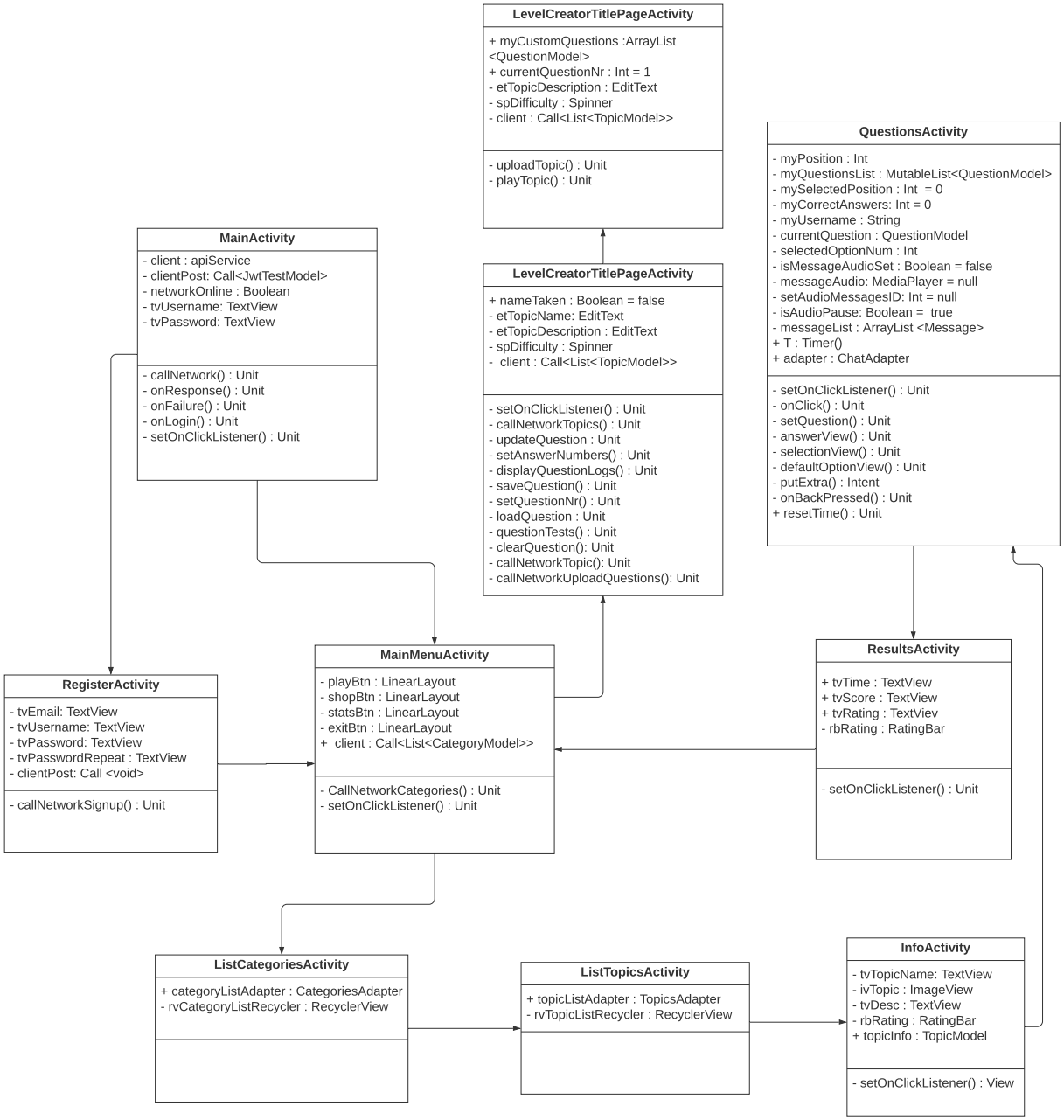

Figure 4. MindGame UML Activity Class Diagram

The preceding UML diagram represents the required variables and functions in front-end android activities of the application described individually in *section 1*. Please note that this class diagram regards only the functions and variables that are declared and utilized in these activity classes specifically, other models, adapters, global objects and functions that appear in non-activity based classes are not represented in this diagram.

2.3 Back-end

The back-end mainly consists of 2 parts: web server and database. Both of them are deployed on OpenNebula on separate virtual machines with only one difference - web server is running in Docker container and database is running natively on Debian.

2.3.1 Overview

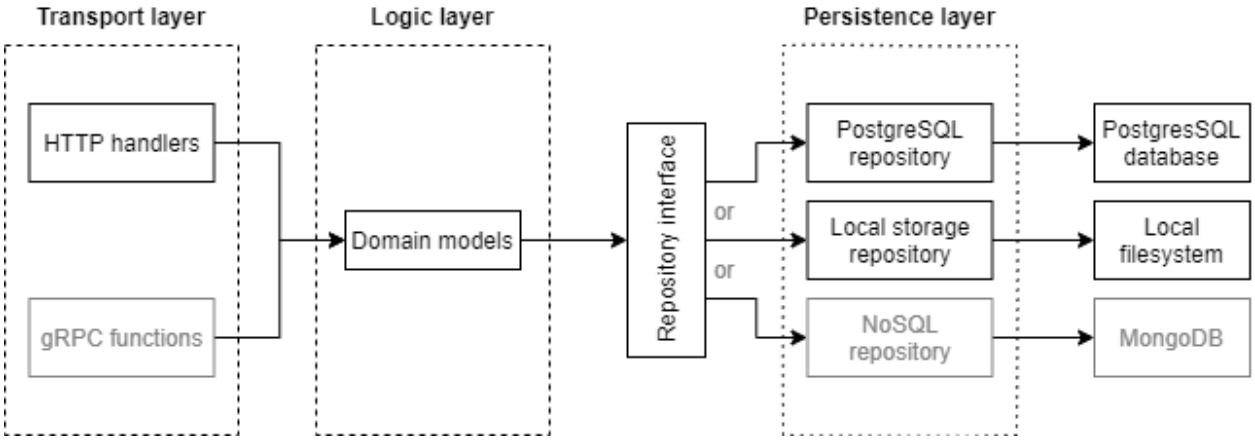

Figure 5. Back-end Overview Diagram

In this diagram (*figure 5*) black text and rectangles show already implemented components while gray ones mean possible additional implementation which are shown here for an example.

2.3.2 Database

For this project PostgreSQL 13 is used as a relational database. Its schema was derived from ER model (*figure 6*) and all of the DDL queries are available on GitLab https://git.mif.vu.lt/anekdotas/mindgameproject.

In order to meet the fifth point in the description of the ER model, SQL function was created along with additional constraints. Together they validate correctness of game session data passed by user by checking relationships between answers, questions and game sessions in such a way that answers must belong to the question to be chosen, questions must be selected in the game session to be answered, and only user that has opened the game session may choose answers for questions in it.

2.3.3 Web server

The web server is written in Go 1.17. It uses Repository pattern to create persistence layer between domain layer and data tier.

Transport layer The communication with clients is done using HTTPS protocol and self-signed TLS certificates. Internally it is implemented as a separate module with HTTP handlers and authentication middleware. Such implementation allows future extensions of the application's transport layer with different protocols and logic that handles them (e.g. gRPC, sockets, etc.). Existing HTTP handlers properly parse input data into internal domain model, initially validate them and pass down to the logic layer.

Persistence layer This layer implements Repository pattern and has an interface for it. We have an PostgreSQL implementation of the repository which is used by the other parts of the application to store, retrieve or update data. As well as transport layer, this one can be extended with other implementations of repository interface without any changes to those parts of the system that actually use the interface itself.

Logic layer This is the main layer in the application that connects all other layers together. Logic methods may be called by any implementation of the transport layer in the same way. This is achieved by implementing modular architecture of the web server. In addition, the repository interface is called by logic but not a specific implementation. This is really helpful for future updates.

Testing Unit testing can be done in harmless way because of the usage of interfaces and modular architecture. This means that almost any component can be tested in isolation - just like it meant to be in unit tests.

2.4 Conceptual Modelling

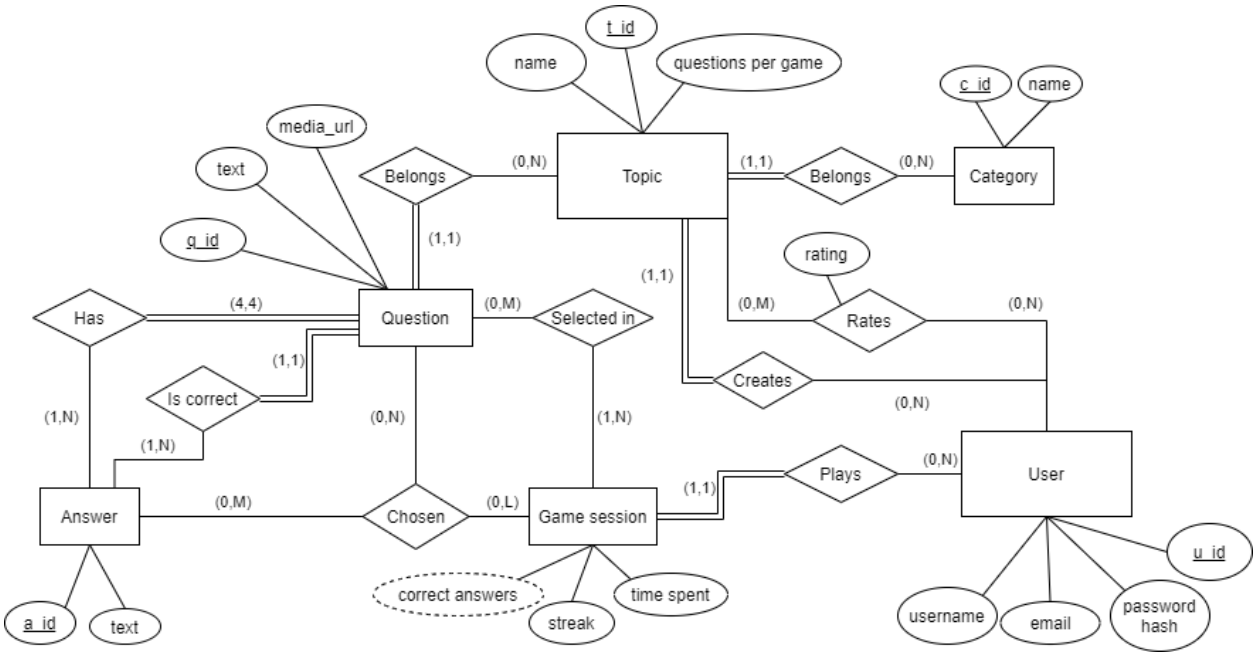

Figure 6. Entity-Relationship diagram

This diagram (*see figure 6*) represents all the relationships in our mini-world and our database structure completely based on it. There are 6 entities: **category**, **topic**, **question**, **answer**, **user** and **game session**. In addition to them, there are also quite a few relationship between:

1. Each category may have multiple topics in it, however, each topic should belong only to one category.
2. Many questions may belong to a specific topic, however, every single question cannot belong to more than one topic.
3. Every question must have 4 answers, exactly one of which is correct. In turn, answers may be related to different questions as well as be correct.
4. Various questions may be selected in many game sessions.
5. Any answer that belongs to a specific question may be chosen in many game sessions in which the question was selected.
6. User may play any amount of game sessions but each of those should belong only to one specific user.
7. User can create many topics, however, each topic can have only one creator (user).
8. Any topic might be rated by multiple users, who also may rate many different topics.

'Question' entity has attributes which represent data that will be sent to the client-side app. For text-based questions only the *'text'*, *'correct answer'* and *'incorrect answers'* attributes will be populated. In case questions require additional media (pictures, audio, video files), *'media url'* will be populated with a location of corresponding file. Thus media files can be stored independently of the database itself.

Any questions always belongs only to one topic. A **'topic'** can have many questions or possibly, none at all. It has attributes which help the player choose between them.

The **'user'** entity represents an authorized unique player. It has attributes which allow the player to securely log into their account.

The **'game session'** entity represents quizzes played by a user. It has attributes which allows collecting statistics for each time played in order to let users track their progress.

- The web server and the PostgreSQL database moving to different VMs to increase scalability and satisfy one of the non-functional requirements regarding security.
- Using the retrofit HTTP client in the Android client side app for receiving text data in JSON format from the web server.
- Moving from postgREST to another type web server, while maintaining the same interfaces with the client side application. This will be done only if the web server itself needs to do processing that cannot easily be done in the postgreSQL database.

3 Functional Requirements

The functional requirements in this section are prioritized and will be described in the form of user stories.

3.1 High Priority

- As a “Mind Game” app player, I want to select a set of questions by choosing a topic, so I can select topics based on my knowledge.
- As a “Mind Game” app player, I want to see my statistics: number of topics completed, percentage of correct answers, time spent on question. This way I can track my progress.
- As a “Mind Game” app player, I want my progress to be saved every time I finish a set of questions, so that I do not have to answer the same questions repetitively.
- As a developer of the “Mind Game” app, I would like to easily add new sets of questions without the need of editing the client-side (mobile) application code.
- As a “Mind Game” app player, I want to share sets of text questions created by me, so that other players can get a better experience.

3.2 Medium Priority

- As a “Mind Game” app player, I want to experience picture guessing and audio questions. This will make my experience more varied, joyful and engaging.
- As a “Mind Game” app player, I want to see how other users rated question sets, so I can better decide what question set I want to select.
- As a “Mind Game” app player, I can continue playing on different devices and have my progress synced between them.
- As a “Mind Game” app player, I want to be able to see other players’ statistics, so I can have fun competing with others.
- As a “Mind Game” app player, I want to rate the question sets I complete, so I can influence the ratings that other players see.

3.3 Low Priority

- As a “Mind Game” app player, who is a non-native English speaker, I would like the application to have the ability to switch to Lithuanian or Russian languages and have ability to select separate sets of questions for my language.
- As a “Mind Game” app player I want to get have short videos along with my questions in order to make my experience more diverse.

4 Non-functional Requirements

4.1 Compatibility

- The mobile app must be able to run on Android 7.0 or greater.
- The system must run in VU infrastructure.

4.2 Reliability

- The system should not have any bugs or issues that prevent the player from having an enjoyable, non-frustrating experience.

4.3 Security

- The database and the server should be in separate virtual machines due to security reasons.
- Avoid real names, cell phone number and other data that could personally identify a player.

4.4 Performance

- Topic loading should be optimized in such a way that on a 3G clients' internet connection, a topic should not take more than 20 seconds to load.

5 Developed Algorithm

5.1 Algorithm Overview

In this section, media saving algorithm is described. **Function SaveMediaFile**
INPUT:
`question_id` - question for which media URL should be update - `integer`
`filename` - media file location - `string`
`src` - content of the media file - `bytes array`
`storage_dir` - directory where file should be stored - `string`
OUTPUT: none
PSEUDO CODE:

```
1     if length(src) = 0 then
2       return
3     end if
4
5     extension := ""
6     splitted := split(filename, ".")
7     if length(splitted) > 1 then
8       extension := splitted[length(splitted) - 1]
9       extension := concat(".", extension)
10    end if
11
12    random_str := random(16) // uniquely randomize 16 characters
13    filepath := concat(storage_dir, "/", random_str, extension)
14
15    dst_file := OpenFile(filepath, O_RDWR|O_CREATE|O_EXCL, 0600)
16    dst_file.write(src)
17
18    update_media_url(question_id, filepath)
```

Function UpdateMediaURL
INPUT:
`question_id` - question for which media URL should be update - `integer`
`media_path` - media file location - `string`
OUTPUT: none
PSEUDO CODE:

```
1     query := "UPDATE questions SET media_url = $1 WHERE id = $2"
2     execute(query, media_path, question_id)
```

5.2 Example Data

Set 1 of Example Data

Question_id: 1

filename: example1.jpg

src: bFtX2E2I8nIW/CfmwNt9W6KmKSbWF1YagLm/v3W+Gev/2Q==

storage_dir: /home/example/media

Set 2 of Example Data

Question_id: 2

filename: example2.png

src: [empty]

storage_dir: /app/files

Set 3 of Example Data

Question_id: 3

filename: example3

src: bFtX2aE2I8nIW/CfmwNt9W6hZms0MUVTHagLm/v3W+Gev/2Q==

storage_dir: /root/media

5.3 Real World Example

Using first set of example data:

- On **line 1** in the IF statement THEN section will be skipped because src's length more than 0.
- On **line 6** splitted will be equal to an array with such elements: ["example1" "jpg"].
- On **line 7** in the IF statement THEN section will be executed and on line 8 and 9 extension will be equal "jpg".
- Let's assume that on **line 12** random_str will be equal to some string "auQ37jUgJSO-_.72".
- On **line 13** filepath will be equal to "/home/example/media/auQ37jUgJSO-_.72.jpg".
- On **line 15** and **16** content of the src will be written to a file and will be available under /home/example/media/auQ37jUgJSO-_.72.jpg path on hard drive.
- On **line 18** update_media_url will be called with parameters question_id equal to 1 and filepath equal to /home/example/media/auQ37jUgJSO-_.72.jpg.

Using second set of data:

- On **line 1** in the IF statement THEN section will be executed because src's length is equal to 0 and as a result algorithm will end.

Using third set of data:

- On **line 1** in the IF statement THEN section will be skipped because src's length more than 0.
- On **line 6** splitted will be equal to an array with such elements: ["example3"].
- On **line 7** in the IF statement THEN section will be skipped because splitted's length is equal to 1.
- Let's assume that on **line 12** random_str will be equal to some string "hgd78ASY8fas_q{-".
- On **line 13** filepath will be equal to "/root/media/hgd78ASY8fas_q{".
- On **line 15** and **16** content of the src will be written to a file and will be available under /root/media/hgd78ASY8fas_q{- path on hard drive.
- On **line 18** update_media_url will be called with parameters question_id equal to 1 and filepath equal to /root/media/hgd78ASY8fas_q{-.

5.4 Data Storage

id	topic_id	text	media_url	correct_answer	answers
7	1	What is Vydmantai?	http://193.219.91.103:7537/media/3238849391.jpg	1	{"A small town in Klaipėda county","A hotel in Utena county", "Unofficial name for PBL team ANEKDOTAS","Bad people"}

Figure 7. Question's data stored in data base

JSON:

```
{
  "id": 7,
  "text": "What is Vydmantai?",
  "mediaUrl": "http://193.219.91.103:7537/media/3238849391.jpg",
  "correctAnswer": 1,
  "answers": [
    "A small town in Klaipeda county",
    "A hotel in Utena county",
    "Unofficial name for PBL team ANEKDOTAS",
    "Bad people"
  ]
}
```

6 Testing

6.1 Unit tests

6.1.1 Front-end

UT001

Title: Incorrect password confirmation.

Test of: validateRegistrationInput().

Instrumented: No.

Description: Unit test returns false upon password and repeatedPassword not matching.

Precondition: password and repeatedPassword match.

Assumption: Username input is of correct format and is not a duplicate, email input is of correct format and is not a duplicate, password contains 6-20 characters and does not contain unsupported characters.

Test Data: username: “Josh”, emailAddress: “Josh@gmail.com” password: “asd123”, repeatedPassword: “asdf123”.

Test Steps:

1. Navigate to anekdotas.mindgameapplication (test) directory.
2. Select RegistrationUnitTest class file.
3. Run ‘incorrect password confirmation returns false’ method (or all associated unit tests by running the class).

Expected Result: Test returns false.

UT002

Title: Password limit test.

Test of: validateRegistrationInput().

Instrumented: No

Description: Unit test returns true upon password length being 6-20 characters.

Precondition: password contains 6-20 characters.

Assumption: Username input is of correct format and is not a duplicate, email input is of correct format and is not a duplicate, password does not contain unsupported characters, password and repeatedPassword match.

Test Data: username: “Josh”, emailAddress: “Josh@gmail.com” password: “asd123”, repeatedPassword: “asdf123”.

Test Steps:

1. Navigate to anekdotas.mindgameapplication (test) directory.
2. Select RegistrationUnitTest class file.
3. Run ‘valid password limit test returns true’ method (or all associated unit tests by running the class).

Expected Result: Test returns true.

UT003

Title: Email format test.

Test of: validateRegistrationInput().

Instrumented: No.

Description: Unit test returns false upon incorrect emailAdress format.

Precondition: emailAdress does not contain email prefix, “@” symbol, email domain.

Assumption: Username input is of correct format and is not a duplicate, password does not contain unsupported characters, password and repeatedPassword match.

Test Data: username: “Josh”, emailAddress: “asd123” password: “asd123”, repeatedPassword: “asdf123”.

Test Steps:

1. Navigate to anekdotas.mindgameapplication (test) directory.
2. Select RegistrationUnitTest class file.
3. Run ‘valid email format test returns false’ method (or all associated unit tests by running the class).

Expected Result: Test returns false.

Note: All of the aforementioned tests use assertThat() method, asserting what the expected results would be either false (it failed) or true (the input was valid).

UT004

Title: Validate edit text.

Test of: MainActivity.

Instrumented: Yes.

Description: Instrumented unit test checks if MainActivity input fields work properly.

Precondition: Text input fields are accessible in the application for the user, tester has an emulator of supported API version set up (necessary for android instrumented tests).

Assumption: Tester has an Android emulator running.

Test Data: stringToBeTyped: “Hello”.

Test Steps:

1. Navigate to anekdotas.mindgameapplication (androidTest) directory.
2. Select MainActivityTest class file.
3. Run validateEditText method. The test does the following check.
   (a) Looks for an editText field with id “username” in MainActivity.
   (b) Types “Hello” into the editText field.
   (c) Verifies the editText field contains “Hello”.

Expected Result: Test returns true on edit success.

Note: Android unit tests are often categorized into instrumented and non-instrumented tests. Instrumented tests in android development consist of tests that rely on android-specific parts (such as activity context or UI objects). Thus contrary to non-instrumented tests they require an emulator in order to to run.

6.1.2 Back-end

UT005

Title: Create question - insertion executed.

Description: Tests if `INSERT INTO` query is executed on question creation attempt.

Precondition: Go 1.17+ installed.

Assumption: none.

Test Data: none.

Test Steps:

1. Go to `src/backend` folder.
2. Execute `go test -v -run TestRepo_CreateQuestion/insertion_executed ./internal/repository/db` .

Expected Result: Test returns success.

UT006

Title: Get questions by topic - select executed.

Description: Tests if `SELECT` query is executed on question retrieval attempt.

Precondition: Go 1.17+ installed.

Assumption: none.

Test Data: none.

Test Steps:

1. Go to `src/backend` folder.
2. Execute `go test -v -run TestRepo_CreateQuestion/select_executed ./internal/repository/db`.

Expected Result: Test returns success.

UT007

Title: Update media URL - update executed.

Description: Tests if `UPDATE` query is executed on media URL update attempt.

Precondition: Go 1.17+ installed.

Assumption: none.

Test Data: none.

Test Steps:

1. Go to `src/backend` folder.
2. Execute `go test -v -run TestRepo_CreateQuestion/update_executed ./internal/repository/db`.

Expected Result: Test returns success.

6.2 Integration Tests

IT001

Title: Integration test of login functionality between web server and client side Android app when provided with correct login data.

Description: A registered user should be able to successfully login in the mind-game Android application.

Precondition: The user must already be registered.

Assumption: A user's device is connected to the internet.

Test Steps:

1. Open the mind-game Android app.
2. In the 'Username' field, enter the username of the registered user.
3. In the 'Password' field, enter the password of the corresponding user.
4. Click 'Play'.

Expected Result: A screen displaying the main menu of the application should be shown, along with a greeting toast that contains the previously entered username.

Diagram describing dataflow and interactions:

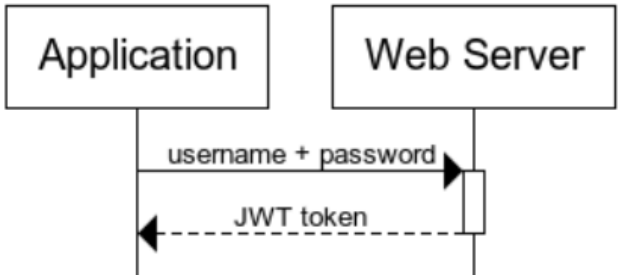

Figure 8. Successful Login Diagram

IT002

Title: Integration test of login functionality between web server and client side Android app when provided with incorrect login data.

Description: If username is incorrect, the user should not be able to proceed.

Precondition: Account with given username does not exist.

Assumption: A user's device is connected to the internet.

Test Steps:

1. Open the mind-game Android app.

2. In the 'Username' field, enter username that is not associated with any account.
3. In the 'Password' field, enter any string of characters. (for example "123").
4. Click 'Play'.

Expected Result: User should not be able to proceed and an error message should be shown.

Diagram describing dataflow and interactions:

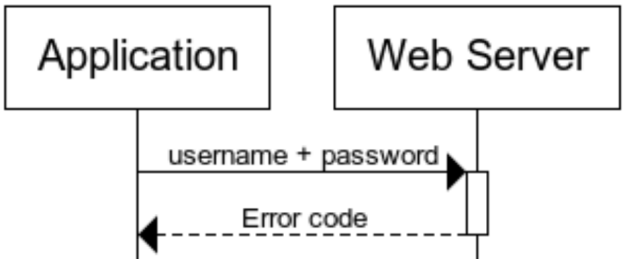

Figure 9. Login Failure Diagram

IT003

Title: Integration test between database and web server for question text and media end points.

Description: Questions, possible answers and image media should be retrieved from the back-end and presented in web browser.

Precondition: Database must have a specific table for topic "Games" containing 7 questions, with 4 possible answers for each question and associated media URL in question 7. Web server must contain image media.

Assumption: Database VM, Web Server VM have access to the internet.

Test Steps:

1. From a web browser call end point for specific topic "Games":
   IP:PORT /categories/1/topics/Games/questions
2. Check that the data matches the data specified in the precondition.
3. Check the media URL, make sure it is accessible from the browser.

Expected Result: Questions, possible answers are shown in JSON format, media URL contains image - observed data is consistent with the precondition.

Diagram describing dataflow and interactions:

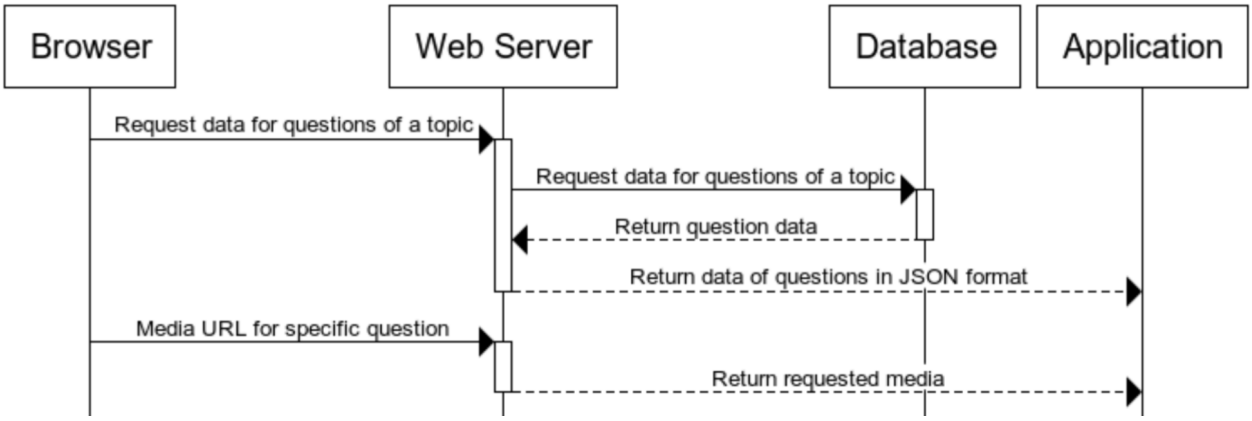

Figure 10. Integration Test for Retrieving Question Data

6.3 System Tests

SYS001

Title: System test for registration of new users.

Description: User should be able to enter new account data and create an account in the Android app.

Precondition: Username used for registration should not be already taken.

Assumption: Database VM, Web Server VM and client-side Android device all have access to the internet.

Test Steps:

1. Open the mind-game Android app.
2. Click on “Don’t have an account?”.
3. Enter username and password of the new account that is to be created.
4. Go back to the initial login screen.
5. Enter the username and password that was used to create the account.

Expected Result: Main menu should be shown along with a greeting containing the username.

Note: Black box test - no diagrams are provided.

6.4 GUI Tests

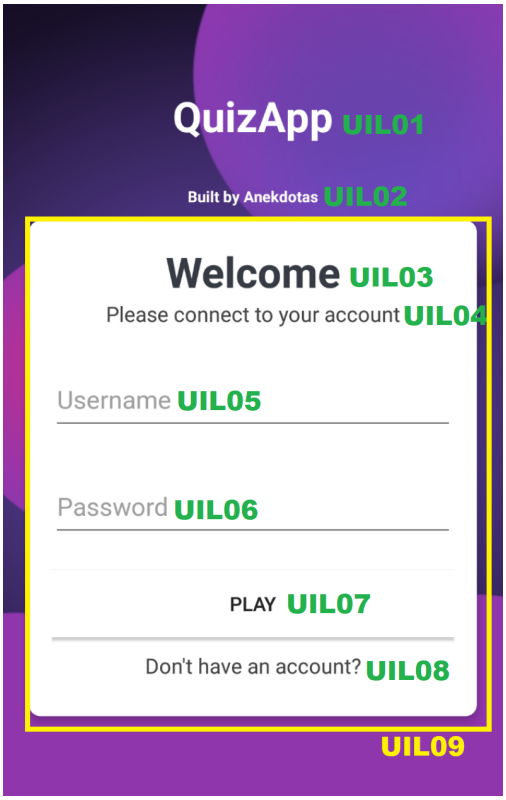

Figure 11. Login Screen

Login Screen

UIL01 - Login Screen (*Figure 11*)

- Validate whether the heading is correct.
- Check the heading position and font.
- Check the readability.

UIL02 - Login Screen (*Figure 11*)

- Check the label position and font.
- Validate whether the heading is correct.
- Check the readability.

UIL03 - Login Screen (*Figure 11*)

- Check the label position and font.
- Check the readability.

UIL04 - Login Screen (*Figure 11*)

- Check the heading position and font.
- Check the readability.

UIL05 - Login Screen (*Figure 11*)

- Check the hint readability.
- Check the position and alignment of the text box.
- Check field labels, validate the acceptance of both valid and invalid characters.

UIL06 - Login Screen (*Figure 11*)

- Check the hint readability.
- Check field labels, validate the acceptance of both valid and invalid characters.
- Check the position and alignment of the text box.

UIL07 - Login Screen (*Figure 11*)

- Test form submission.
- Check button position and clarity.

UIL08 - Login Screen (*Figure 11*)

- Check button position and clarity.

UIL09 - Login Screen (*Figure 11*)

- Ensure it wraps UIL03-08
- Validate a gap between itself and the screen frame.

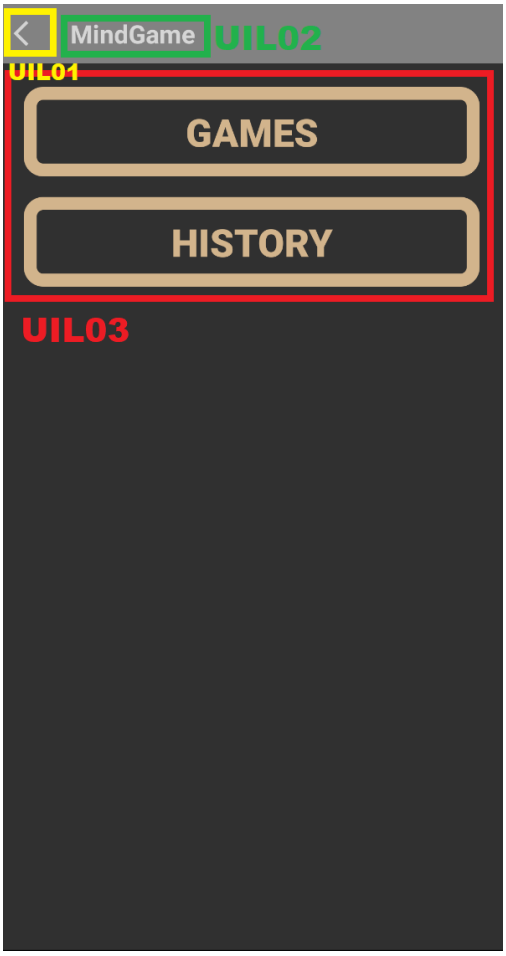

Figure 12. Topic Listing Screen

Topic Selection Screen

UIL01 - Topic Selection Screen (*Figure 12*)

- Test redirection to the Category Listing screen.

UIL02 - Topic Selection Screen (*Figure 12*)

- Validate whether the heading is correct.
- Check the heading position and font.
- Check the readability.

UIL03 - Topic Selection Screen (*Figure 12*)

- Test redirection to the selected category's list of topics.
- Validate a gap between button and the screen frame.
- Validate a gap between buttons.
- Confirm if the first button can be clicked.
- Check if there is a gap between the topic name and the outline of the topic name button.

6.5 Acceptance Tests

AT001

Title: Acceptance test of Android app for retrieving category list.

Description: The user should be presented with a list of available categories.

Precondition: The user must have completed the login procedure and entered the main menu.

Assumption: A user's device is connected to the internet.

Test Steps:

1. Get to the 'main menu' screen in the Android app.
2. Click the 'play' button in the main menu screen.

Expected Result: A screen showing the list of categories should be shown.

AT002

Title: Acceptance test of Android app for retrieving topic list.

Description: The user should be presented with a list of available topics.

Precondition: The user must have selected a category.

Assumption: A user's device is connected to the internet.

Test Steps:

1. Get to the category list screen in the Android app.
2. Click on one of the categories from the given list.

Expected Result: A screen showing the list of topics for the selected category should be shown.

AT003

Title: Acceptance test of Android app for retrieving quiz questions.

Description: The user should be presented with a question along with possible answers for said question.

Precondition: The user must have selected a topic.

Assumption: A user's device is connected to the internet.

Test Steps:

1. Get to topic list screen in Android app.
2. Click on one of the topics from the given list.

Expected Result: The first question for the selected topic is shown, along with the possible answers.

7 Competitive Analysis

The following section analyzes advantageous, disadvantageous features and the general player feeling of similar quiz applications with the goal of determining the competitive ability of the application in development. To achieve this, three other quiz applications have been selected and played through. The quiz programs analyzed were chosen randomly but contain a substantial (over 1 million downloads) user base and a generally favorable review score. An attempt to find applications of same, or extremely similar, question interface design (*see section 1.3*) was also made but was unsuccessful. Additionally, since the analysis was done entirely by our team, preference bias is a possibility.

7.1 Analysis

7.1.1 "Super Quiz - General Knowledge Russian" by WalkMe Mobile Solutions

Advantages:

- **Concise category selection** - each different topic is divided into categories of Geography, Entertainment, History, Art and Literature, Nature and Science, Sport, making it easy to select which topic a user would prefer to play, which provides more enjoyment to the user.
- **Daily login bonuses** - the application contains in game currency and daily currency bonus just for turning on the application in an attempt to keep player retention over a longer period of time.
- **Achievement system** - gaining achievements is not hard at all - player receives revards for completing achievements while playing the game. For instance, after player answers correctly 10 times in a row, after the end of a game (winnig or losing by answering incorrectly), pop-up appears on the screen and informs about successfully completing one or more achievements. Information about name, actions needed to complete and revard of the acievement is represented on that pop-up.

Disadvantages:

- **Cliché question structure** - structure common among most quiz applications. The quiz game interface provides nothing special of interest apart from the questions themselves and the possible answers making the application similar to hundreds of quizzes of such kind.
- **Too many advertisements** - advertisements are everywhere, from continuing a quiz if a user answered incorrectly, to earning in-game currency by watching them, to simply being forced to watch them in a middle of the quiz.

General thoughts:

Though common, achievement and daily in-game currency bonus systems are an options that allow to keep steady user retention, could potentially be something worth implementing.

7.1.2 "Quiz Planet" by LOTUM one GmbH

Advantages:

- **Nice Visuals** - the application has a consistent theme among all the screens, pleasant color scheme and correctly uses basic rules of composition.
- **Opponent system** - there are two people that get the same quiz questions and after finishing they see results of each other and scores they both got. This creates more engagement for the application as the players try to answer as correctly as possible in order to defeat their opponent, the sense of competition drives the user to continue playing.

Disadvantages:

- **Cliché question structure** - structure common among most quiz applications. The quiz game interface provides nothing special of interest apart from the questions themselves and the possible answers making the application similar to hundreds of quizzes of such kind.
- **Lacks user friendliness** - the interface is rather confusing at times, especially where to tap after quiz ends.
- **Too many advertisements** - the player has to watch an ad every 2-3 games which have only at maximum 3 questions. Eventually, it happens really often and becomes annoying to the player. There is no benefit from watching the ad for the players, they are forced to do that.

General thoughts:

This application has really good idea of adding competitive aspect and creating attractive design for it. However, it is a regular quiz with textual questions and 4 possible answers which is similar to most of other quiz application. In addition, its fifty-fifty UX design helps players to start a game by showing one big “Start” button but is confusing and not user-friendly in other places like ending the game or quitting. And one of the most important disadvantages is a willingness of the developers to overly monetize their game by adding a lot of advertisements.

7.1.3 "Quiz - offline games" by The Angry Kraken

Advantages:

- **User statistics are integrated with the quiz itself** - After answering each question the user is able to see how many people have answered that question correctly. This provides a competitive element to the application.
- **Charming visuals** - the interface, while rather simple, has a fairly aesthetically pleasing design.

Disadvantages:

- **Unadjusted difficulty** - the questions are often far too simple, and would fit best for a children, yet the application does not state its target demographic, which leads to a conclusion it aims to provide quizzes for every demographic.
- **Limitations for playthroughs** - Users are limited on how many quizzes they can do by an energy meter which can be replenished by either purchasing more or waiting, this hinders player enjoyment.
- **Cliché question structure** - structure common among most quiz applications. The quiz game interface provides nothing special of interest apart from the questions themselves and the possible answers making the application similar to hundreds of quizzes of such kind.
- **Too many advertisements** - the application contains many, long advertisements that can not be skipped and are shown randomly.

General thoughts:

The application implements user statistics tracking in an interesting way, which the Mind Game application described in the report is aimed to achieve as well. Though in the reviewed application the accuracy of the tracked statistics is rather dubious considering the fact that a user can play the quiz offline and the statistics are still shown without internet connection.

7.2 Results of Competitive Analysis

All of the applications tested have a lot of similar features, with the most important one to mention being the question structure, which only includes a question and possible answers with little to nothing else of interest, relying solely on the entertainment value of the questions, this has become a staple in the quiz application genre. Considering that our application aims to stand out in an already overpopulated field of mobile phone quiz programs the game interface that offers a different take on the mind game genre (*see section 1.3*) is extremely beneficial from the user engagement perspective. Additionally, since the application is being made with academic rather than business purposes in mind the advertisements that plague many quiz applications of similar nature are not going to be added (at the very least for the early versions of the system) with the aim of providing a better experience for the player. It is also important to mention that the analysis yielded some results regarding features that could be potentially implemented into our application, which include but are not limit to: achievements and daily rewards.

8 Related work

The choice of Kotlin programming language for this project is reinforced by scientific research. A 2020 article in "*Information and Software Technology*" titled "*Effectiveness of Kotlin vs. Java in Android App Development Tasks*"[1] has concluded the use of Kotlin in Android application development, leads to more compact code that is less prone to errors in comparison to the equivalent in Java.

References

[1] Luca Ardito, Riccardo Coppola, Giovanni Malnati, and Marco Torchiano. Effectiveness of kotlin vs. java in android app development tasks. *Information and Software Technology*, 127:106374, 2020. Last accessed 4 November 2021.