|  |
| --- |
| Coursera Android specialization |
| Capstone Project |
| Mutibo |
|  |
|  |
| **10/26/2014** |

|  |
| --- |
|  |

## Functional specification

### Introduction

This variant of the mutibo project aims at creating a general-purpose quiz game for android leveraging the architecture and technologies presented in the coursera android specialization path.

The quizzes will follow the approach of sets consisting of 3 similar-1 different entries, and even though the final project will present only movie entries, an effort will be given to elaborate a design that is generic enough to accommodate different items with only small modifications.

### Functional description

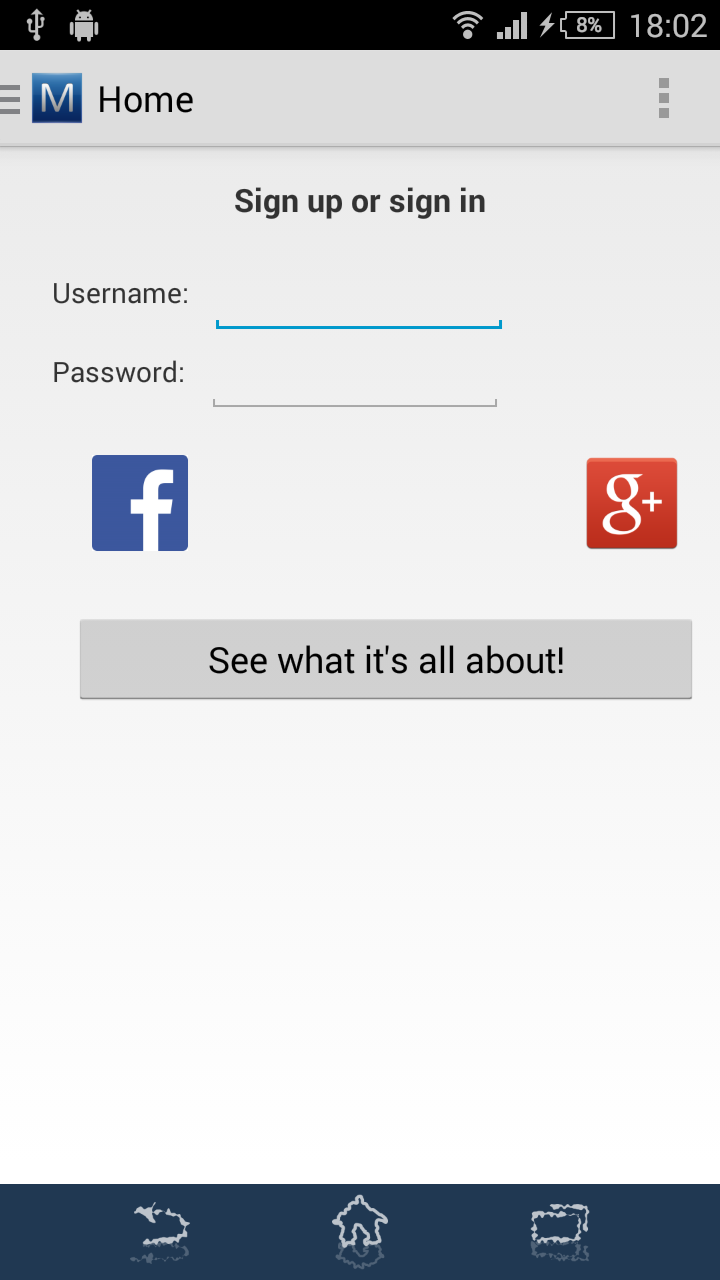
1. Users can be authenticated via username/password or (optionally) via facebook/google account.
2. If they choose not to authenticate they only have access to a demo game that only contains a few predefined sets
3. For authenticated users only, the initial possible actions will be 3 types of game: "*Solo*", "*Vendetta*", "Gang" as well as "*High scores*" and "Options". The actions should always be accessible in a navigation drawer. (<https://developer.android.com/design/patterns/navigation-drawer.html>)
4. "*Solo*" will start a game, where every question will pertain to a set. A set will be portrayed as four titles and (optionally in the final version) with images of the movie
5. The images of the movie will be shown in case the Amazon Mobile App SDK is integrated.
6. After the user gives an answer (either correct or incorrect), he can view a text with information on the similarity of the movies.
7. Each correct answer gives one point to the accumulated score.
8. In each game, the player has the right to give two wrong answers. The third incorrect guess will mark the end of the game.
9. The total score upon the end of a game will possibly qualify for a place in the list of high scores of the corresponding screen, where only the 10 highest scores are shown.
10. After reading the explanation of each set (which movie is irrelevant), the user can either "like", or "dislike" the specific set. Sets that have at least 10 dislikes and not more than half as much likes (e.g. 30 dislikes and <=15 likes) will be automatically removed from upcoming games.
11. "*Vendetta*" is game against a facebook friend (if facebook integration is implemented) or a mutibo user identified by a user name otherwise. The Vendetta game is a sudden death playoff, where the first player to make one mistake loses the game. The vendetta results are not stored in this version but in the future could be displayed as a new screen or optionally posted on facebook.
12. "*Gang*" is a game in cooperation with a facebook friend. The users could chat (using external app like facebook messenger) to agree on the correct answer. The first click on an answer by any of the two players is considered valid. The gang results are not stored in this version but in the future could be displayed as a new screen or optionally posted on facebook.
13. The first version will most likely show only a user’s facebook friends and interact with them. A further improvement on the friends functionality would be to have a list of in-app friends, that consist of any of the following:
    1. a) mutibo user names added manually,
    2. b) hand-picked facebook friends,
    3. c) hand-picked google+ contacts.   
       So one could for example have 5 available mutibo friends to play “Vendetta” or “Gang” with, 2 from facebook, 2 from google and 1 from mutibo.
14. Additional functionality to be considered for authorized users will be to skip a question, or ask a facebook friend for the answer of a question. A user will be given such bonuses only after a number (e.g. 10) of consecutive correct answers.
15. The application will provide the gestures that signal the immediate beginning of a new game regardless of the current state. The possible gestures are S for solo, V for Vendetta, and G for Gang. Any currently ongoing game will be abandoned with a proper warning.
16. The application will also use the animation android API to animate the question movies into the screen on each round.
17. The movie and image data will attempt to be pulled from [Amazon’s Product Advertising API](https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html). This is not a binding requirement though, in case there is not enough time there will be dummy movie data, just to demonstrate the game operations.
18. The app will potentially leverage Amazon Appstore or GameCircle API for leaderboards.
19. In case the movie data contain information about the difficulty of a set, the difficulty level of the game will vary according to the user level, the more correct answers, the harder it will get.

### Requirements mapping

The following table summarizes the project requirements and how they’re addressed in the functional and technical specifications of this document.

|  |  |
| --- | --- |
| **Requirement** | **Coverage** |
| 1. App supports multiple users via individual user accounts | Func. Description 1 & technical description 4 |
| 2. App contains at least one user facing function available only to authenticated users | Func. Description 1, 2 |
| 3. App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components:  Activity  BroadcastReceiver  Service  ContentProvider | Technical Description 2 |
| App interacts with at least one remotely-hosted Java Spring-based service | Technical Description 1 |
| App interacts over the network via HTTP | Technical Description 1,3,5 |
| App allows users to navigate between 3 or more user interface screens at runtime | Functional description 3 and User interface section |
| App uses at least one advanced capability or API from the following list (covered in the MoCCA Specialization): multimedia capture, multimedia playback, touch gestures, sensors, animation | Functional description 15, 16 |
| App supports at least one operation that is performed off the UI Thread in one or more background Threads of Thread pool. | Technical Description 10 |
| A Set is a unit of data that contains four movie titles, optional associated images for each movie, information identifying the one movie that is not like the other three, and accompanying text, explaining the relationship between the three related movies. | Introduction, Func. Description and DB schema. |
| A User should be able to log into the game using an authenticated user account. | Func. description 1 & UI home page |
| A single game presents a series of Sets and guesses, until the User has made three Incorrect Guesses. | Introduction and Func. Description 8 |
| After viewing a Set, a User will be able to rate a Set based on the explanation of the link between the movies. | Func. Description 6 and UI Game screen |
| If a Set receives a large number of poor ratings, it can be removed from the game. | Func. Description 10 |
| For each successfully completed Set, the user will get *Points*. | Func. Description 7 |
| All data (questions, answers, points, etc.) are stored to and retrieved from a web-based service accessible in the cloud. | Technical Description 3,9 & REST API |
| Users could be allowed to challenge a friend to a sudden death playoff. For example, friends could answer questions turn by turn, and the first person to make a mistake loses. | Func. Description 11 |
| Users could be given progressively difficult questions (e.g., based on other users' previous success with each Set). | Func. Description 19 & Tech. Description 7 |
| Users could be given special "power ups" such as the ability to pass a Set, or to get help from a friend them when they are stuck. | Func. Description 14 and UI Game Screen |
| Users could be allowed to challenge Facebook friends to do various things such as to beat their high score, to help answer a question they are stuck on, etc. | Func. Description 14 and UI Game Screen |
| The app could be optimized for the Amazon Appstore and could leverage Amazon’s GameCircle API to incorporate leaderboards. | Func. Description 18 |

### User Interface

The functionality of the application is partially demonstrated in the following screenshots:

#### Home page

The home page asks the user to login via an individual mutibo user name and password, via facebook or google account.

The first time a user opens the application, she can create a mutibo username and password by logging in via facebook or google.

There is an additional button to play a demo game without signing in.

### 

### D:\Work\Dropbox\Camera Uploads\2014-10-26 17.02.49.pngMain menu

The navigation drawer screen menu can open by tapping the mutibo icon on the top left or scrolling from the left edge of the screen.

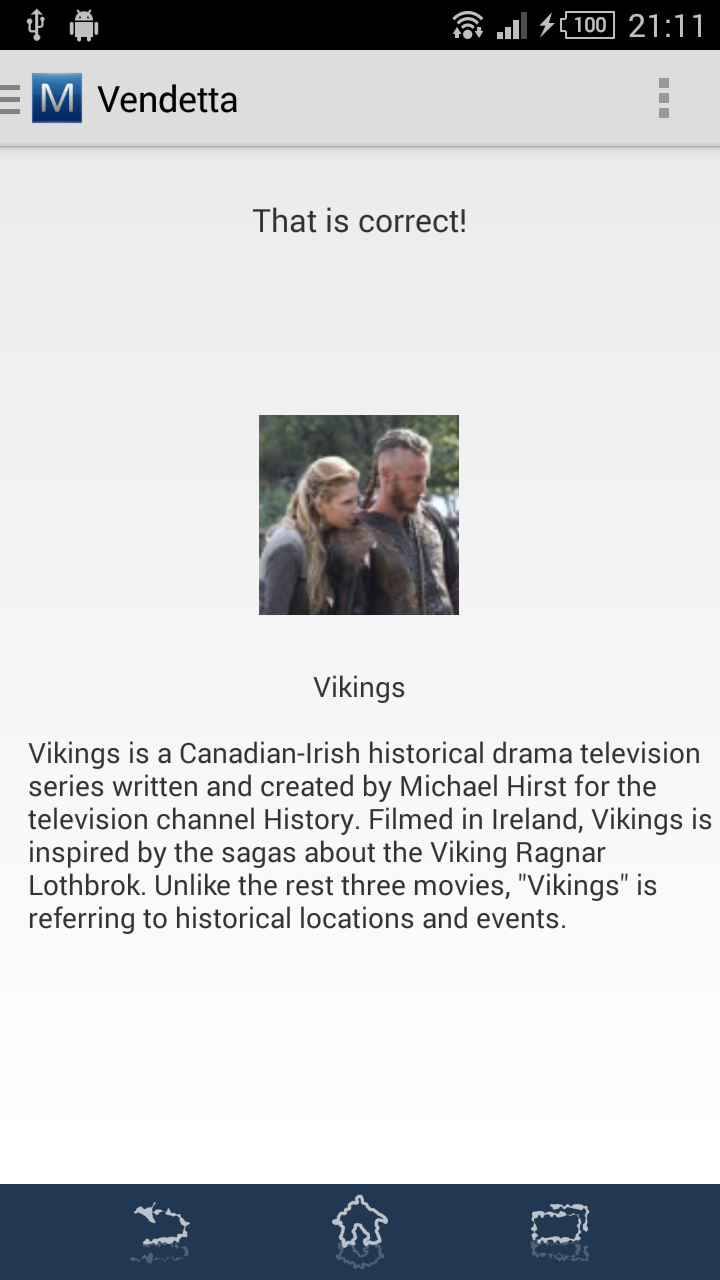
It contains the screen possibilities: Solo (new solo game), Vendetta (new Vendetta game), Gang (new Gang game), Hot scores (achievements screen), Options (settings of the app).

None of the game types can be started without logging in.

### D:\Work\Dropbox\Camera Uploads\2014-10-26 20.11.59.pngGame screen

The game screen is approximately the same in all game modes. In a new solo game each set is represented on screen with 4 movie titles accompanied by their respective images. The user has to tap on one title or image to answer.

For advanced users there will be a possibility to ask a facebook friend or skip a question, under the conditions described earlier.



After giving an answer, either successful or unsuccessful, the user can see details about the correct answer and an explanation. Tapping anywhere on the screen will lead to the next set of movies.

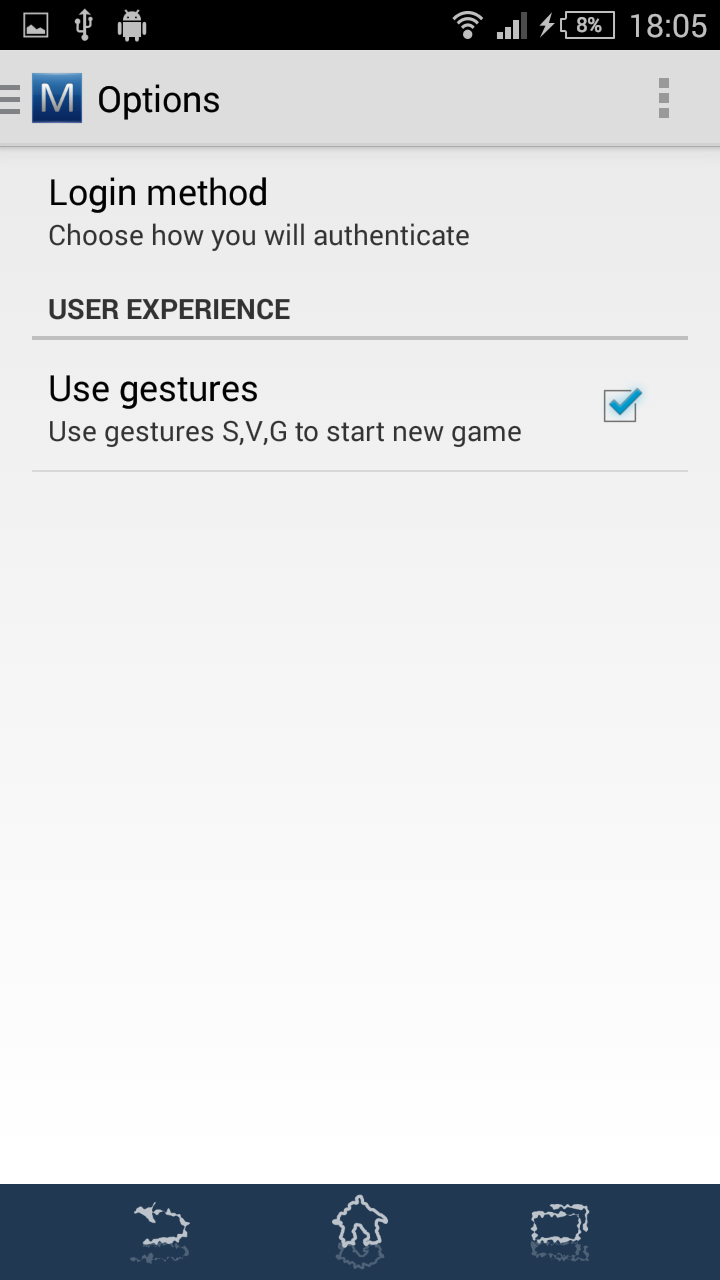
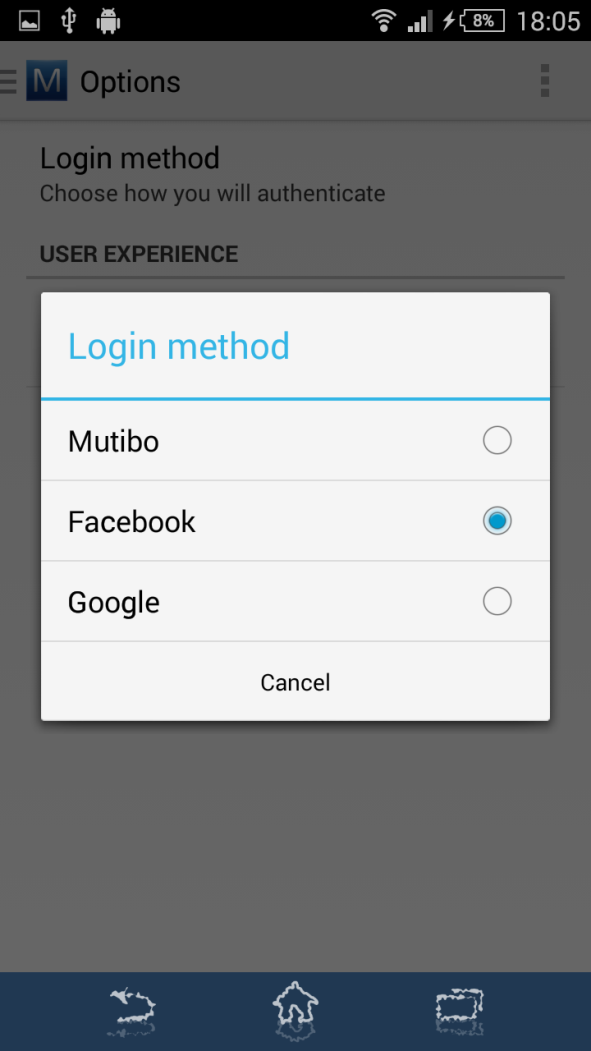


#### Hot Scores screen

As a more attractively named screen than “high scores”, it provides a list of the user’s achievements in solo mode. For the other modes it will be considered to post them on facebook for the final project. This serves as the list screen described in the project requirements.

**Options screen**

The settings screen provides some basic app parameters, like the preferred login method and whether the user would like to use gestures to start a new game. The login method opens a new list preference as seen on the second screenshot.



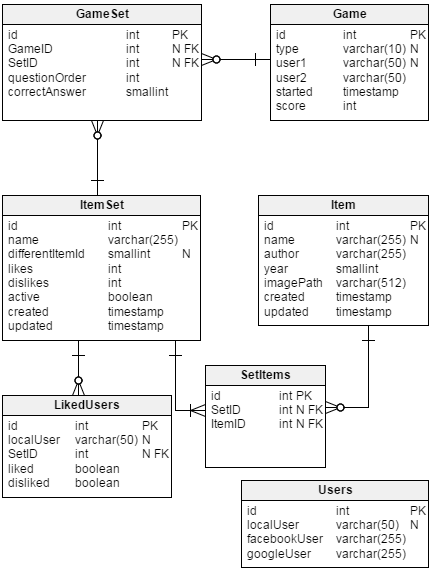
### Technical description

Below are some of the basic decisions on the technical aspects of the project.

1. The project will consist of a server Spring based application that exposes REST operations via HTTP and an Android client application that the user can use to play the game on her phone.
2. The client application will mainly consist of
   1. A basic android activity that exposes the user interface in the form of fragments
   2. A bound android service that will asynchronously exchange information with the server about new sets etc.
3. The client application will use a Sync Adapter (<http://developer.android.com/training/sync-adapters/index.html>) for communication with the server that will be realized over HTTP. Particularly the Client will leverage a Retrofit based REST client as shown in the mobile cloud MOOC.
4. The server will use Spring Security and Spring OAuth 2.0 to support multiple users via a custom UserDetailsService. Users will authenticate via the Spring OAuth 2.0 token endpoint and provide bearer tokens in an authorization header to prove user identity.
5. The server will contain one or more controllers that serve the REST HTTP requests as demonstrated in the mobilecloud MOOC.
6. The database access will be performed using JPA entities and Repositories as described and demonstrated in the mobilecloud MOOC.
7. The server will be stateless when it comes to the active user’s game. The “game id” has to be provided by the client via REST, but the game data (already answered questions etc.) will be kept on the server side in order to:
   1. Avoid repeated questions
   2. Aid in eventually increasing the game difficulty
   3. Provide game history in the future, especially for the collaboration modes (vendetta, gang)
8. The user preferences shown on the corresponding UI screen will be stored only on the client mobile app using the built-in android preferences functionality. The preferences UI will use the Android *PreferenceFragment* to display the available options.
9. No storage of movies/sets will be performed on the client. The client will have to request every time something is needed (or for that matter also in advance). The request will be always against the REST API described in a next section.
10. The request will be performed in a thread retrieved from a Thread pool to avoid blocking user operation in the UI thread.

#### Database

The set and game data is maintained in a relational database. As a first phase this database is an HSQLDB persisted on disk. As the system scope is not to provide a production ready solution, no more attention will be given to the question what DB to use, as well as NoSQL options will not be considered for the purposes of the capstone project.

The area of focus is rather the DB schema, which has to be designed according to the project requirements. The proposed schema can be seen in the following diagram. The basic entities are:

- **Item**: an item of a question. In this case a movie. It contains basic information that can correspond in our case to a movie, like the author (director), year, the path to an external image file on the server, as well as creation and update timestamps.

- **Set**: a set of Items that can be used as a question. The columns contain a name (optional), an index of the different question that can receive values 0..3, the number of users that liked and disliked this set, if it's active (by default true) as well as timestamps.

- **SetItems**: realizes the many-to-many (M..N) relationship between Sets and Items.

- **Game**: every game started by the user is stored here. The type is one of the 3 types offered. Two user names, as currently there is a hard limitation of up to two users playing a game simultaneously. Also we keep the time it was started and the current (or final if finished) score. It's not yet clear if we'll need to store the status of the game (finished, ongoing etc.) and end timestamp. They will be added if needed.

- **GameSet**: realizes the many-to-many (M..N) relationship between Sets and Games. Stores additionally the order in which a specific set appeared in a game as a question, and whether the given answer was correct. The latter field is of integer type to anticipate future versions of the game where more than one attempts to answer a question might be permitted.

- **Users**: Storage of user information, particularly the mapping of a mutibo user (localUser) to her corresponding facebook and google accounts. This structure has to be reconsidered according to the interface with these systems.

- **LikedUsers**: list of the users who liked or disliked a set. It serves as a history for like/dislike display purposes to the user herself.

This model is subject to modifications for the final hand-in version of the project.

#### REST interface

Below is a summary of the basic REST operations that facilitate the communication between the Spring based HTTP server and the android application:

|  |  |  |
| --- | --- | --- |
| **Operation** | **HTTP Method** | **Description** |
| /oauth/token | POST | The access point for the OAuth 2.0 Password Grant flow. |
| /item | POST | Adds a new item (movie) to the database |
| /item/{id} | GET | Returns the information of the specific item in JSON format |
| /itemset/{id}/like | POST | Sets an item as liked by the user that made the request (liking an item unsets any previous unlike of the same user for the same item) |
| /itemset/{id}/unlike | POST | Sets an item as unliked by the user that made the request (unliking an item unsets any previous like of the same user for the same item) |
| /itemset/{id}/like | GET | Returns whether the current user has liked or unliked the item indicated by {id}. Returns a composite object with both flags like/unlike. |
| /itemset/next?gameid={gameid} | GET | Returns the next item for the current user and her active game. |
| /itemset/demo | GET | Returns the list of sets that comprise the demo game offered. |
| /game/start/{type} | POST | Starts a new game for the current user, of the given type. Returns the game ID to be used by the client for subsequent calls. |
| /game/finish?gameid={gameid} | POST | Finishes the current game. |
| /game/answer gameid={gameid}&setid=  {setid}&answer={answer} | POST | Sets the user answer for a particular set. |

In all operations the root path “/mutibo” is implied. All operations, particularly the authentication related ones are likely to be enhanced.