Mutibo Design Document

A Coursera Android Capstone project

Author	*Redacted for peer review*
Revision	v1.0
Last modified	2014-11-30
Number of pages	31

Table of Contents

1. Revision History

Revision	Date	Comments
v0.1	2014-10-19	Initial draft
v0.2	2014-11-02	Fixes a few copy&paste and spelling errors
v1.0	2014-11-25	Updates to reflect final design

2. Introduction

This document describes the design for the Mutibo project of the Android Capstone Project on Coursera. For more information about this project and its requirements please visit the project homepage at https://class.coursera.org/androidcapstone-001/wiki/Mutibo.

Please do not feel daunted by the size of this document. A large portion is taken up by API-references and screen mockups and are quickly read.

Chapter 3 describes the architecture of the server back-end. The fourth chapter delves into the design of the android application. In the fifth chapter we discuss the implementation time line and the various milestones we want to deliver. For your convenience the sixth chapter provides direct answers to the assessment rubric for the mid-term evaluation of this document.

The appendix lists the external resources that will be used to implement all the parts of this project.

3. Server architecture

3.1. Overview

The server is be a Java web-server that handles client-requests through a rest-like API. The service is implemented using Spring with a MongoDB database as a storage back-end.

As an extension to the requirements of the Mutibo project we introduce the concept of a “deck”. A deck is a grouping of Mutibo-sets that are managed as a single entity. This allows us to simplify the release cycle of new sets and manage updates to previously released sets.

3.2. Functionality

The server provides these functions :

1. User management and authentication
2. Management of game information
3. Data-synchronization with client applications
4. Game results and leaderboards
5. A limited lobby for multiplayer games

3.2.1 User management and authentication

Users can create accounts from the android application and need to authenticate before using any functionality of the server. The server defines two user roles : ADMIN and PLAYER. A user with the PLAYER-role is allowed to play games but can not modify movies or sets. A user with the ADMIN-role can define and edit movies and sets.

The android app supports authentication through Google+ and Facebook. The server also supports user/password authentication but at the moment this is only used for the management module.

The android app requests an access token from Google+ or Facebook and sends this to the server. The server validates this Google/Facebook specicfic access token and generates an internal token that is used by the client for the rest of the session. This encapsulation allows extra authentication methods to be added easily.

Any valid Google+ or Facebook-user can login, user-accounts are created on the fly. When a user logs in for the very first time he/she is given the opportunity to change the assigned nickname. The default nickname is constructed from the firstname of the user on the social-network.

3.2.2 Management of game information

The server exposes several API-functions to manage decks, sets and the movies contained therein.

These functions also encapsulate the use of external API's to retrieve information about the movies (e.g. posters). This allows us to transparently change these details during the course of the applications' lifetime without affecting the client. This also insulates the client visible portions of the application from failure of services not under our control.

A simple web-application (using jQuery and Handlebars) was developed to relatively easy management of decks, sets and movies during the course of the project.

3.2.3 Data-synchronization with client applications

To keep the number of required client-requests to a minimum most information will be transferred a whole deck at a time. The decks are relatively small so the tradeoff with respect to required bandwidth is justified.

Any client application is expected to cache information locally to further reduce bandwidth requirements and delays at application start-up. The android app developed for this course implements client side caching. The API is designed to make it easy to replace out-of-date information.

3.2.4 Game results and leaderboards

The results of each played game are reported back to the server with an optional user provided rating of the quality of the set. Rating information is visualized in the management interface. Results are sent back to server once in bulk at the end of a game session. When the server is not reachable at that time the data is stored locally and sent at a later time.

The game scores are used to build leaderboards which can be consulted by the client application.

3.2.5 A limited lobby for multiplayer games

Due to time constraints multiplayer is not implemented yet.

As a bonus feature a simple multiplayer lobby system will be implemented. This lobby will provide two ways of starting a multiplayer game.

1. When the user was authenticated through a social media service, like Facebook, the user will be able to challenge his friends of the social media to a Mutibo match. If the challenged user is also a known Mutibo-player the challenge will be send through Google Cloud Messaging for Android. If the challenged player is not a Mutibo player a message will be sent on the social network with information on how to obtain Mutibo.
2. A user can elect to challenge a random online user. A multiplayer game will be started against another user who has also challenged a random online user. If no waiting players are available the user will be asked to wait until a player becomes available.

Communication between players engaged in a multiplayer Mutibo battle will be handled by the server.

3.3. Database design

A NoSQL document database, specifically MongoDB, was chosen to implement the server back-end. This has influenced the design of the data structures. Easy synchronization with clients was another major consideration. Please refer to Illustration 1: Database Schema on page 5 for a schematic overview.

A DECK is a grouping of SETs that are processed simultaneously for certain actions. This includes marking a DECK for release and transferring information to the client. A hash of all the deck-information sent to the client is stored and can be used to quickly determine if a locally cached set must be refreshed.

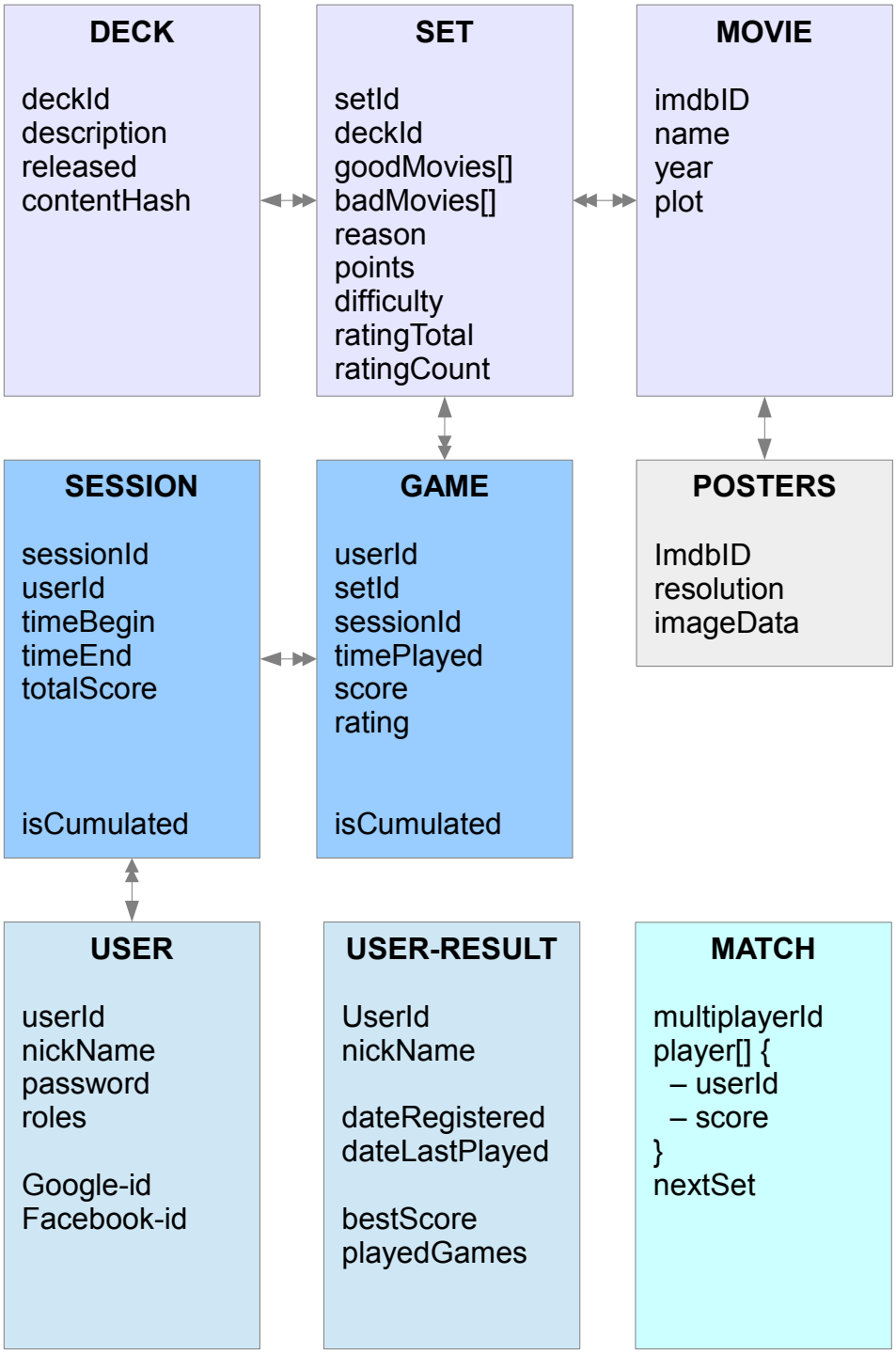

Illustration 1: Database Schema

A SET contains links to the movies and the information required to play one game. The user-ratings are tallied up regularly and cached in the set. The ratings aren't displayed by the client application so they are not included in the synchronization hash.

The identification of the movie on IMDB is used as the primary key of a MOVIE as this seems to be commonly used in multiple online movie database. All information required about a movie is retrieved from external services (or entered by an admin-user) and then served to clients. Posters for movies are stored in a separate collection.

The GAME collection contains information pertaining to the results of a single played set and a SESSION has the results of one play-through of a game. The “isCumulated” field is used by a server background procedure to update the sumation-fields in SET and USER.

A USER has a fixed user-id to make it simple to change a nickname of a player functionality is not surfaced at moment. Only the information needed to authenticate a user is store in this collection. Statistical information is stored seperatly in USER-RESULT.

The MATCH collection contains information about currently running multiplayer matches.

3.4. API reference

3.4.1 User management and authentication

/login/login-password?username={*username*}&password={*password*}		POST
Action	Authenticate a user. Returns an access-token in the X-Auth-Token header if successful or an error code at failure.	
Permission	Anonymous	
Parameters	username	The nickname of the player that wants to login
	password	The password of the player that wants to login
Response-body	status	“OK” : user login successful “FAIL” : login failed
	X-Auth-Token	Authentication token to be used in subsequent requests

/login/login-google?googleToken={*googleToken*}&username={user*name*}		POST
Action	Authenticate a user through Google+. Returns an access-token in the X-Auth-Token header if successful or an error code at failure.	
Permission	Anonymous	
Parameters	username	First name on GooglePlus of the user
	googleToken	Token obtained from Google after local authentication
Response-body	status	“OK” : user login successful “NEW” : previously unknown user logged in “FAIL” : login failed
	X-Auth-Token	Authentication token to be used in subsequent requests

/login/login-facebook? fbToken={*fbToken*}&userId={*userId*}&username={user*name*}		POST
Action	Authenticate a user through Facebook. Returns an access-token in the X-Auth-Token header if successful or an error code at failure.	
Permission	Anonymous	
Parameters	fbToken	Token obtained from Facebook after local authentication.
	userId	Facebook-id of the user
	username	First name of the user on Facebook.
Response-body	status	“OK” : user login successful “NEW” : previously unknown user logged in “FAIL” : login failed
	X-Auth-Token	Authentication token to be used in subsequent requests

/user/change-name?current={*current*}&new={*new*}		POST
Action	Change the nickname of the current user. Mostly used when a new user logs in.	
Permission	USER	
Parameters	nickname	The nickname of the new player
	password_hash	(optional) The password of the new player
HTTP-Response	200 - OK 409 - Confict	Operation successful New nickname is not valid (already in use?)

/user/current		DELETE
Action	Delete the current user from the database	
Permission	USER	
Parameters	-	-
HTTP-Response	200 - OK	

3.4.2 Movies

/movie		GET
Action	Retrieves a list of all known movies	
Permission	ADMIN	
Parameters	-	-
Response-body	*list*	List of movie-records

/movie/{id}		GET
Action	Retrieve information about a particular movie.	
Permission	ADMIN	
Parameters	id	The IMDB-id of the movie to retrieve.
Response-body	*record*	The current contents of the MOVIE-record with the specified ID.

/movie/{id}		POST
Action	Add a new movie or update an existing movie. The server retrieves extra information about the movie from external services and returns it to the client.	
Permission	ADMIN	
Parameters	id	The IMDB-id of the movie to add.
Response-body	*record*	The current contents of the MOVIE-record with the specified ID.

/movie/find-by-name?name={pattern}		GET
Action	Search for movies matching a particular name. External sources are consulted to retrieve the list of movies	
Permission	ADMIN	
Parameters	pattern	The name or pattern to search by
Response-body	*list*	List of MOVIE-records that match the query

/movie/poster?id={*id*}&resolution={resolution}		GET
Action	Retrieve the poster of the specified movie in the specified resolution	
Permission	USER / ADMIN	
Parameters	id	The IMDB-id of the movie poster to retrieve
	resolution	Indication of the resolution to retrieve (low/medium/high...)
Response-body	image	image/jpeg data

3.4.3 Decks

/deck/{*id*}		GET
Action	Retrieve the information of a single deck	
Permission	ADMIN	
Parameters	id	The id of the deck to be retrieved.
Response-body	*record*	DECK-record with the current information of the requested id.

/deck		GET
Action	Retrieve a list of all decks	
Permission	ADMIN	
Parameters	id	The id of the deck to be retrieved.
Response-body	*record*	DECK-record with the current information of the requested id.

/deck/list-released		GET
Action	Retrieve a list of all the released decks and their content-hashes	
Permission	USER / ADMIN	
Parameters	-	-
Response-body	*List*	List of DECK-records.

/deck		POST
Action	Adds or updates a deck.	
Permission	ADMIN	
Request-body	*record*	DECK-record to be added/updated.
Response-body	*record*	Current information of the deck.

/deck/release?id={*id*}		POST
Action	Release the specified deck. Basically a shorthand method.	
Permission	ADMIN	
Parameters	id	The id of the deck to be released.
Response-body	*record*	Current information of the deck.

3.4.4 Sets

/set/{*id*}		GET
Action	Retrieve the information of a specific set	
Permission	ADMIN	
Parameters	id	The id of the set to be retrieved
Response-body	*record*	Current information of the set.

/set		POST
Action	Add or update a set	
Permission	ADMIN	
Request-body	*record*	Record containing the information of the set to add or update.
Response-body	*record*	Current information of the set.

/set/{*id*}		DELETE
Action	Delete a set	
Permission	ADMIN	
Parameters	id	Id of the set to be deleted
Response-body	*record*	Current information of the set before it was deleted.

3.4.5 Data-synchronization

/sync?id={id}&hash={hash}		GET
Action	Retrieve all the required information to play all the sets in the specified deck. The hash is included in the URI to easily force a client to ignore a cached version of the deck.	
Permission	USER / ADMIN	
Parameters	id	The id of the deck to be retrieved
	hash	The contentHash of the deck to be retrieved
Response-body	*record*	Information about the requested deck augmented with two lists :
	+ movies	A list of all the MOVIE-records used in the deck.
	+ sets	A list of all SET-records included in the deck.

3.4.6 Games

/game/results			POST
Action	Store the results of a game-session.		
Permission	USER		
Request-body	startTime	UTC time when the game-session was started	
	endTime	UTC time when the game-session ended	
	set-results	List of records with the results of each played set : - setId - timePlayed : UTC time when the game was played - score : the obtained score by the player for this game - rating : user rating (0 – 5)	
HTTP-Response	200 - OK	Operation successful	

3.4.7 Leaderboard

/game/leaderboard?from={*from*}&count={*count*}			GET
Action	Retrieve a part of the leaderboard		
Permission	USER		
Parameters	from	The starting position of the portion of the leaderboard to retrieve.	
	count	The number of entries to return	
Response-body	entries	List of leaderboard entries. Empty if the requested indices lie beyond the available range. - ranking : position in the leaderboard - nickname : player's nickname - bestScore : the highest registered score of this player - playedGames : the amount of games played by this player.	

/game/leaderboard-player?player={nickname}&count={*count*}			GET
Action	Retrieve a part of the leaderboard centered around the specified player		
Permission	USER		
Parameters	player	Nickname of the player to center the results around	
	count	The number of entries to return	
Response-body	entries	List of leaderboard entries (see previous entry).	

3.4.8 Multiplayer

Multiplayer has not been implemented yet.

/multiplayer/challenge_random		POST
Action	Challenge a random player to a multiplayer match. If no opponent is currently available, place this player in the wait queue.	
Permission	USER	
Parameters	-	-
Response-body	multiplayerId	Id of this multiplayer-session to use in subsequent requests.
	ready	Is an opponent available or not (0/1)
	opponentName	(optional) name of the opponent
	setId	Id of the first set to play

/multiplayer/challenge_friend		POST
Action	Challenge a particular friend to a multiplayer match. If the user is not a known Mutibo-player a message will be sent on the socia network	
Permission	USER	
Parameters	friend-id	Id of the friend to challenge
Response-body	mutibo	Is the challenged friend a known Mutibo-player (0/1) ?
	sessionId	(optional) id of this multiplayer-session to use in subsequent requests. Only if the challenged friend is a known Mutibo-player

/multiplayer/game		POST
Action	Post the results playing a set in a multiplayer match	
Permission	USER	
Parameters	multiplayerId	Id of the multiplayer-session you are engaged in.
	setId	Id of the played set
	score	Result of the played set
Response-body	setId	Id of the next set to play

/multiplayer/ping		POST
Action	Notify the server that your still active in the multiplayer game.	
Permission	USER	
Parameters	multiplayerId	Id of the multiplayer-session you are engaged in.
Response-body	-	-

Information and events about the opposing player are sent through Google Cloud Messaging for

Android. The contents of these messages is in a JSON-format with a fixed format :

```
{
  'type' : <type>
  'params' : [
     <type specific params>
  ]
}
```

A list of the currently defined messages :

Type	**Explanation and parameters**
OPPONENT_READY	An opponent is ready to join your multiplayer game - nickname : name of the opponent
OPPONENT_QUIT	The opponent exited the multiplayer-session before the end
OPPONENT_SCORE	Current score of your opponent - nickname : name of the opponent - totalScore : current total score - lives : remaining number of lives
SCORE	Your current score according to the server. - totalScore : current total score - lives : remaining number of lives
NEWS	Information about an event to be displayed in the app's news ticker .

3.4.9 Social network features

Not been implemented yet.

/social/friends?&mutibo-only={1/0}		GET
Action	Request a list of friends from the social network used to login the current player.	
Permission	USER	
Parameters	mutibo-only	Boolean : only return friends with a Mutibo account or all of them
Response-body	*list*	List of friend records with fields : - friend-id - name - mutibo

4. Android App

4.1. Philosophy and points of interest

4.1.1 Dialog with the player

To avoid having the app feel as an exam and not a game we'll try to adopt a more informal tone when communicating with the player. This is reflected in some of the screen mockups presented later but will be refined later on in the project.

4.1.2 Authentication

Authentication solely through social networks has been chosen for this project. Mostly to take advantage of the friends lists of those networks for multiplayer games but also to to avoid forcing players to manage yet another account.

Google+ is supported for obvious reasons as most android users can be expected to have one for the Google Play Store. Facebook support also seems worthwhile because it's arguably still the most popular network. The app requests a minimal set of permissions from the social-network and the server does not store any of this information. The app provides functionality to revoke the granted access to the social-network and remove the account on the server.

4.1.3 Offline play

Given the relatively short development time to implement the project it was decided not to implement full offline play functionality, at first.

The player will have to be online when finishing a game to post the results to the leaderboards. Efforts to cache results will only be undertaken when sufficient time remains at the end of the development cycle.

However the application will do its best to fetch the information required to play the games only once and caches it locally.

4.1.4 Multiplayer

The player will be able to challenge his friends from a social network to a Mutibo match. A basic multiplayer lobby will also be implemented to allow random strangers to play each other.

During a multiplayer game both players will receive the same questions. The game continues until a player answers three questions wrong. The player with the highest score wins the game. When both players have the same amount of points no tie breaker will be implemented.

The score of the opponent will be shown alongside the score of the player.

4.1.5 Set selection

The game selects the next set to be played in manner to appear random to the player. Under the hood though certain factors can be used to skew the random selection. Sets with lower rating will have a lower chance to be chosen.

4.2. Screen flow

4.2.1 The login screen

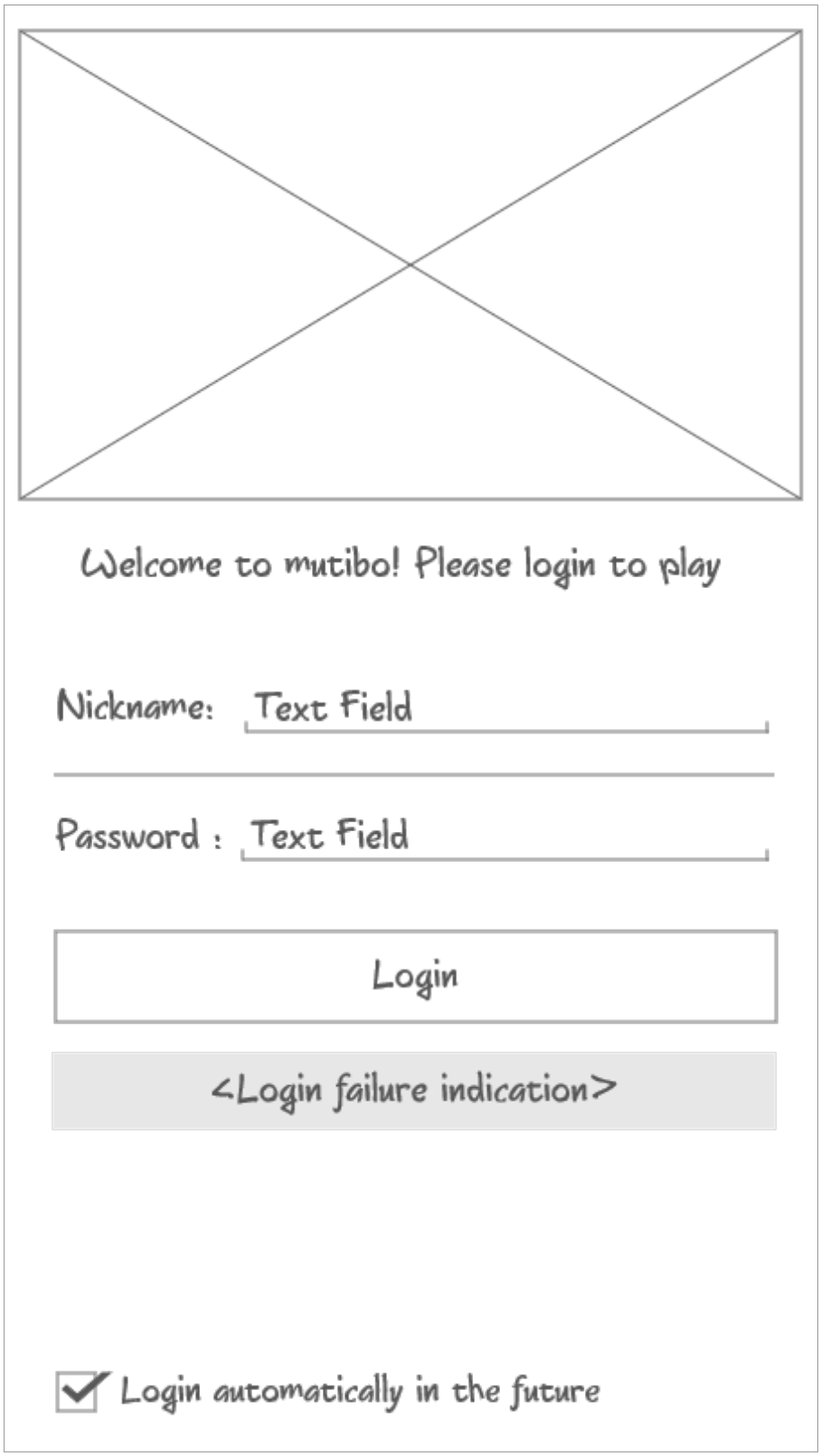

Illustration 2: Preliminary login screen

In the first stage of the implementation only a basic user name/password login screen will be implemented. Once the basis functionality of the app is complete this basic login procedure will be replaced with a login procedure using a social network account of the user.

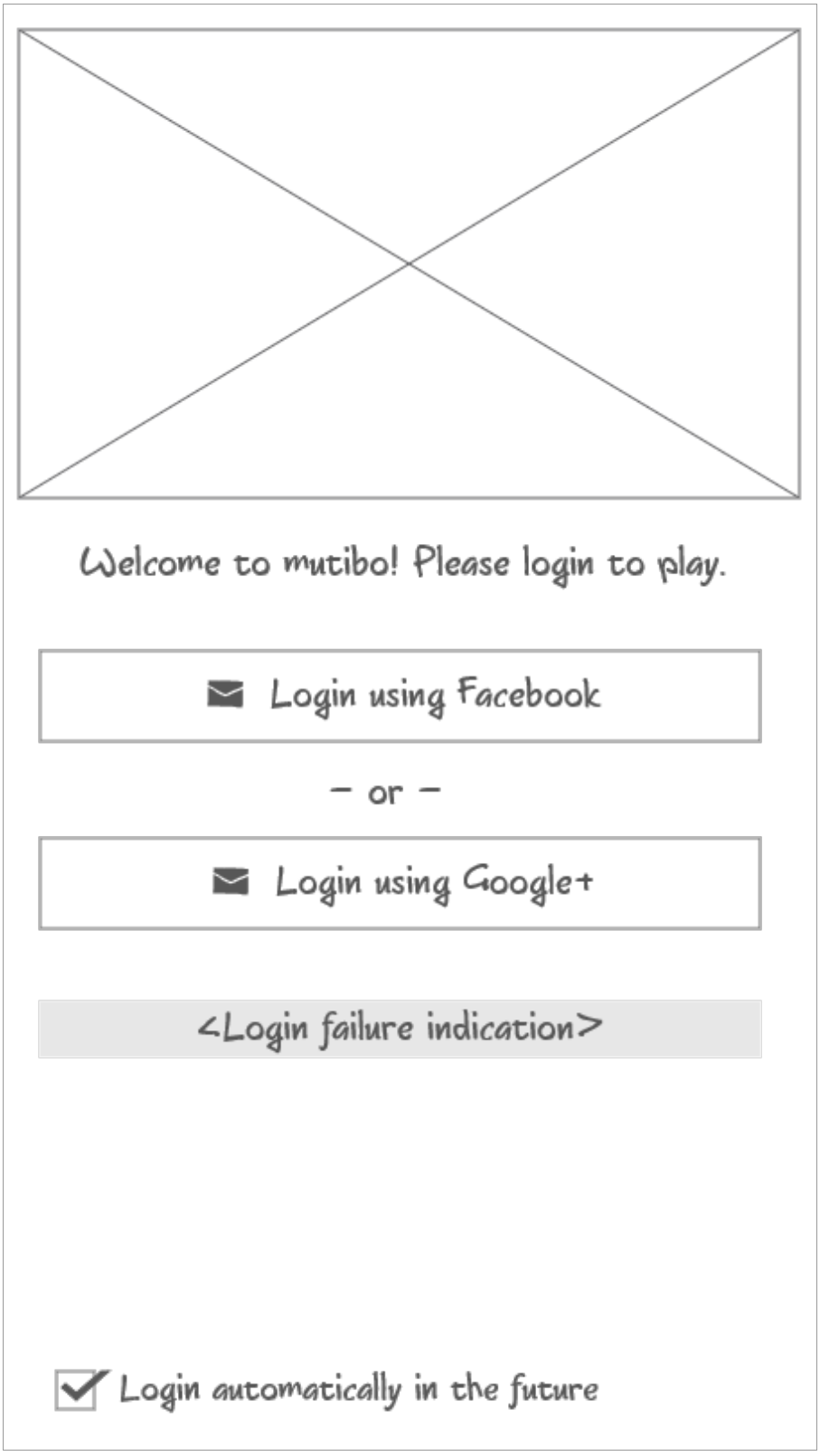

Illustration 3: Final login screen

When a user logs in for the first time, he/she will be prompted for additional information. Currently the user is able to change the nickname to be used on the leaderboards. The name of the user on the social network is used a default.

Illustration 4: First login of a user

4.2.2 Menu

After login the user is presented with a menu screen.

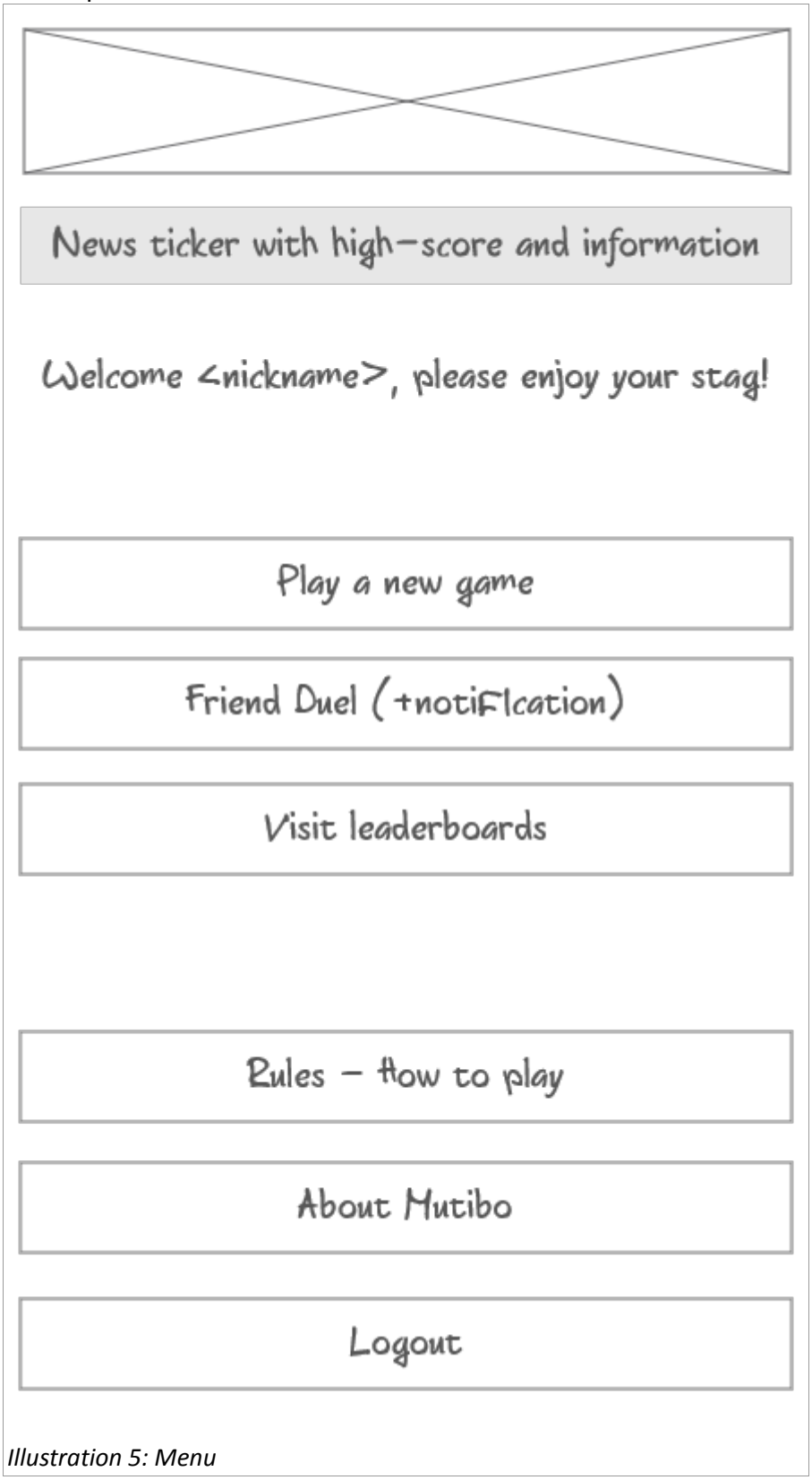

Illustration 5: Menu

A news ticker displays information about events that could be important to the player. Besides choosing to play a game or looking up the ranking a player can peruse a help screen and an screen providing more information about this project and its background.

4.2.3 Playing a game

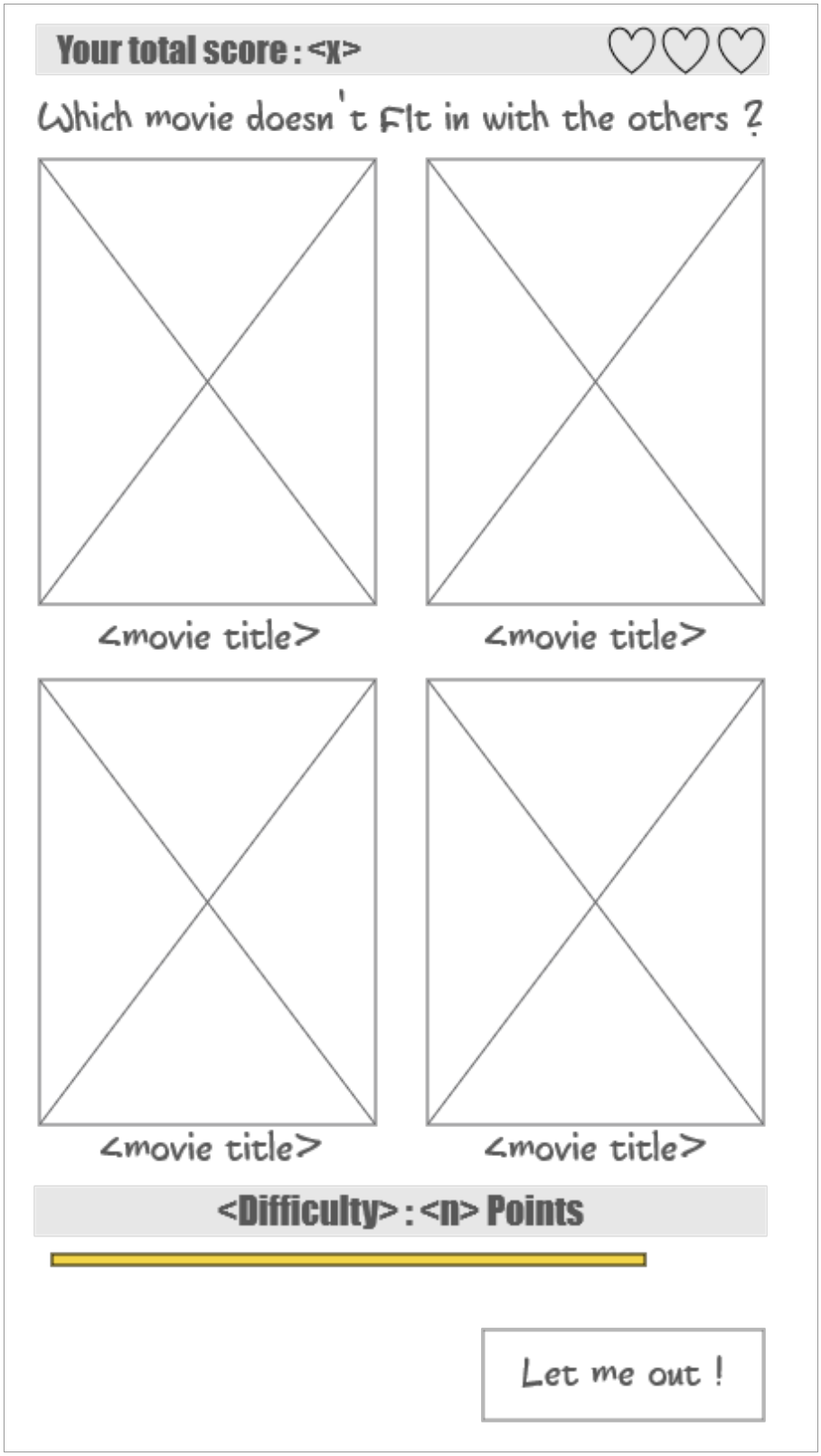

Illustration 6: Game screen

Choosing “Play a new game” starts a new single-player game.
Posters for four movies are shown along with the movie-title. An animated progress bar indicates the time that is left for the player to make his/her choice.

The difficulty of the question and the associated number points to earn are listed at the bottom of the screen along with a button to preliminary end the game. At the top of the screen the current score and the remaining 'lives' are displayed.

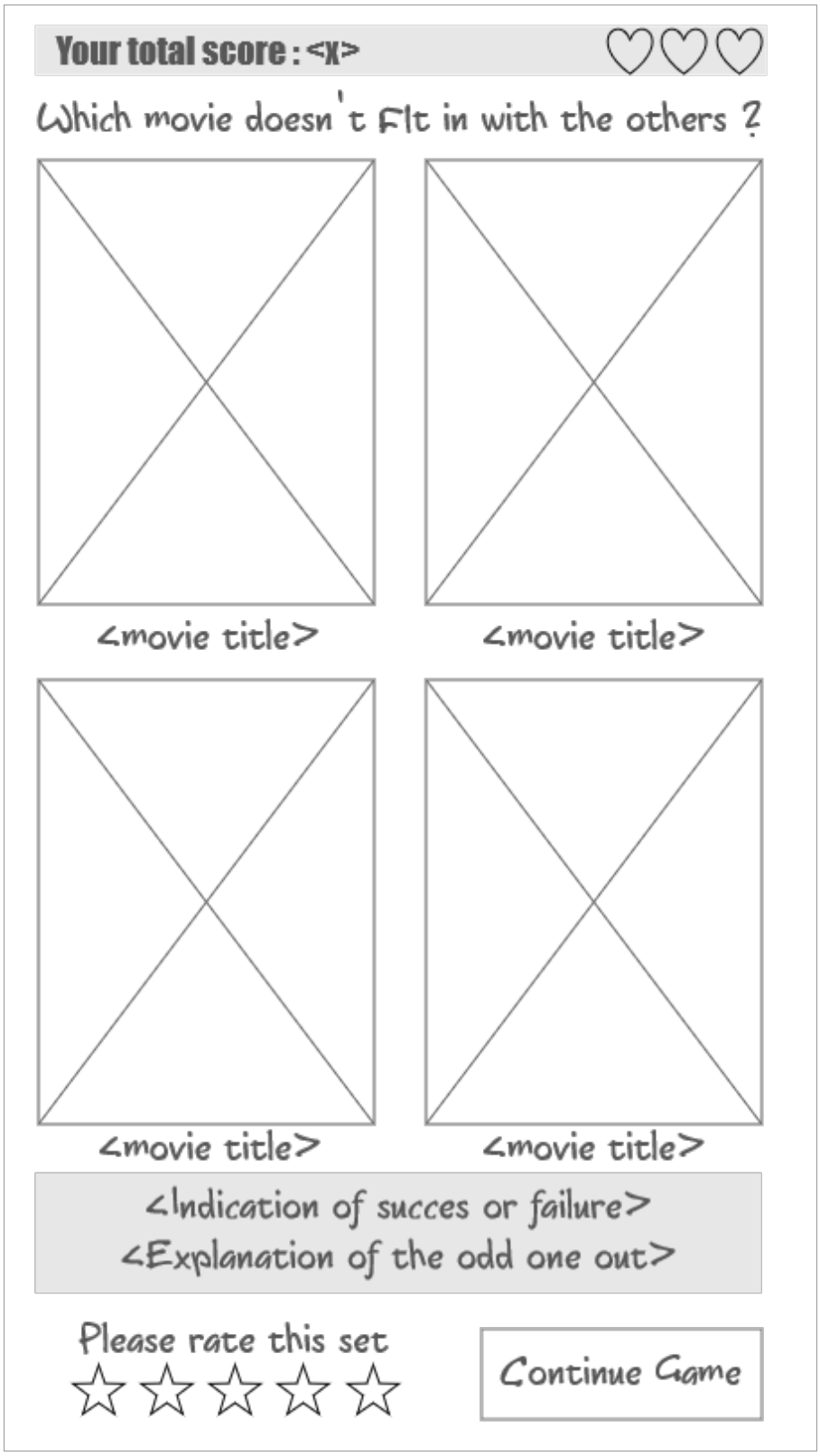

Illustration 7: Game results

The selected movie is emphasized using a neutral colored frame. The color of the frame will change to a color indicating success or failure and a appropriate sound will be played. All this information will also be displayed as a text message.

The player can, optionally, rate this set and then move on to the next question.

Slightly alternative layouts for landscape-mode are also implemented.

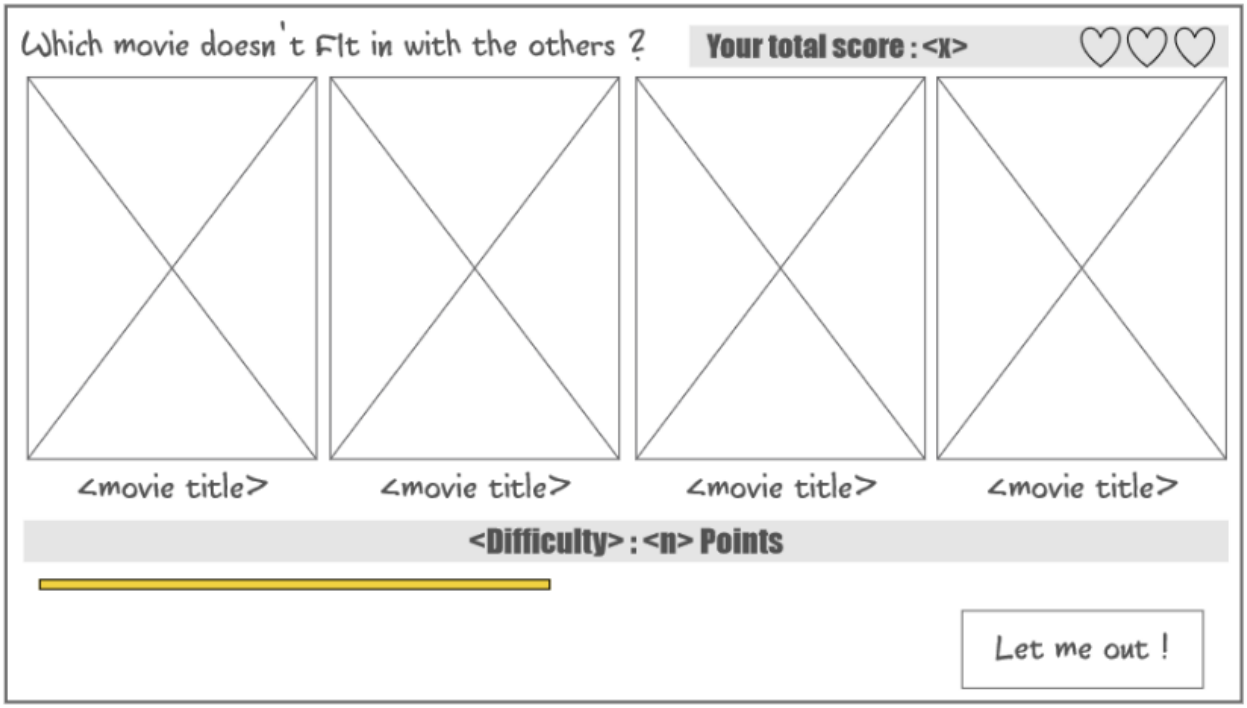

Illustration 8: Landscape game

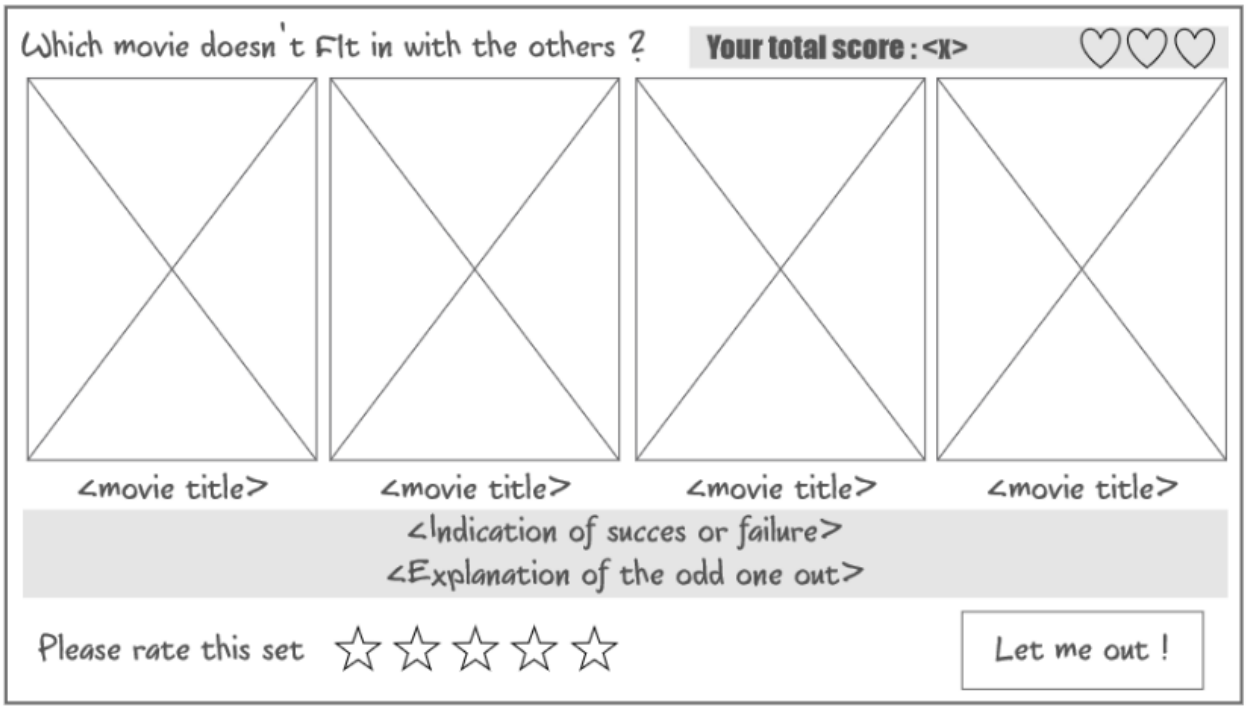

Illustration 9: Landscape set results

The game finishes when a player answers three sets wrong. But the game also ends when the player has exhausted all available sets before getting three wrong.

Illustration 10: End of game

4.2.4 Leaderboards

The leaderboard screen displays the ranking of the people who have played Mutibo. By default the screen show the ranking of the authenticated player and provides several options to quickly display the top ten and the relative ranking of your friends. The list can be scrolled to view more players.

Illustration 11: Leaderboards

4.2.5 Multiplayer

On the multiplayer screen the player can challenge other players to a multiplayer duel. The player can opt to player a random opponent or choose to challenge a friend. The friend list indicates which players were online recently and can also be filtered.

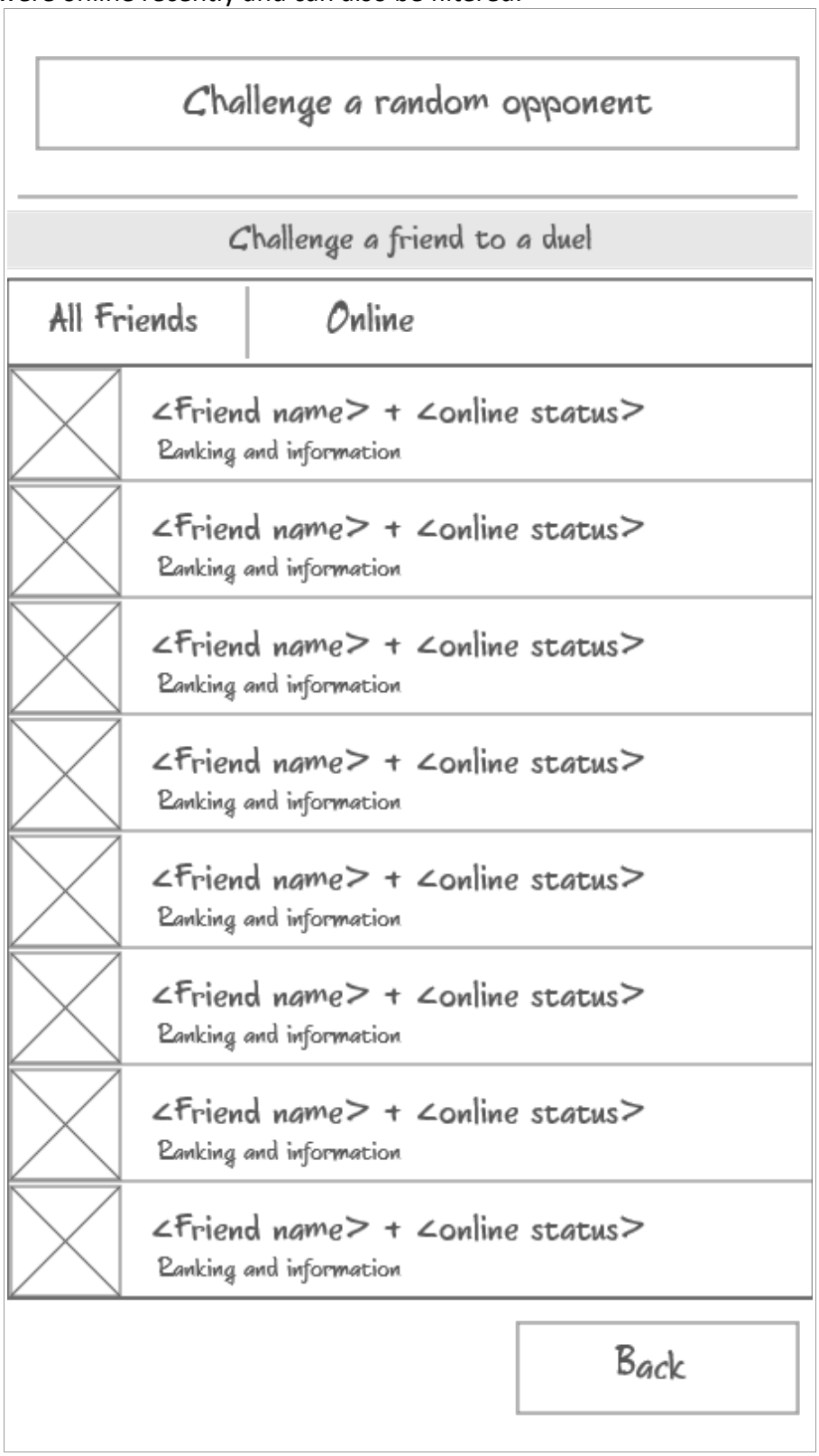

Illustration 12: Multiplayer challenge

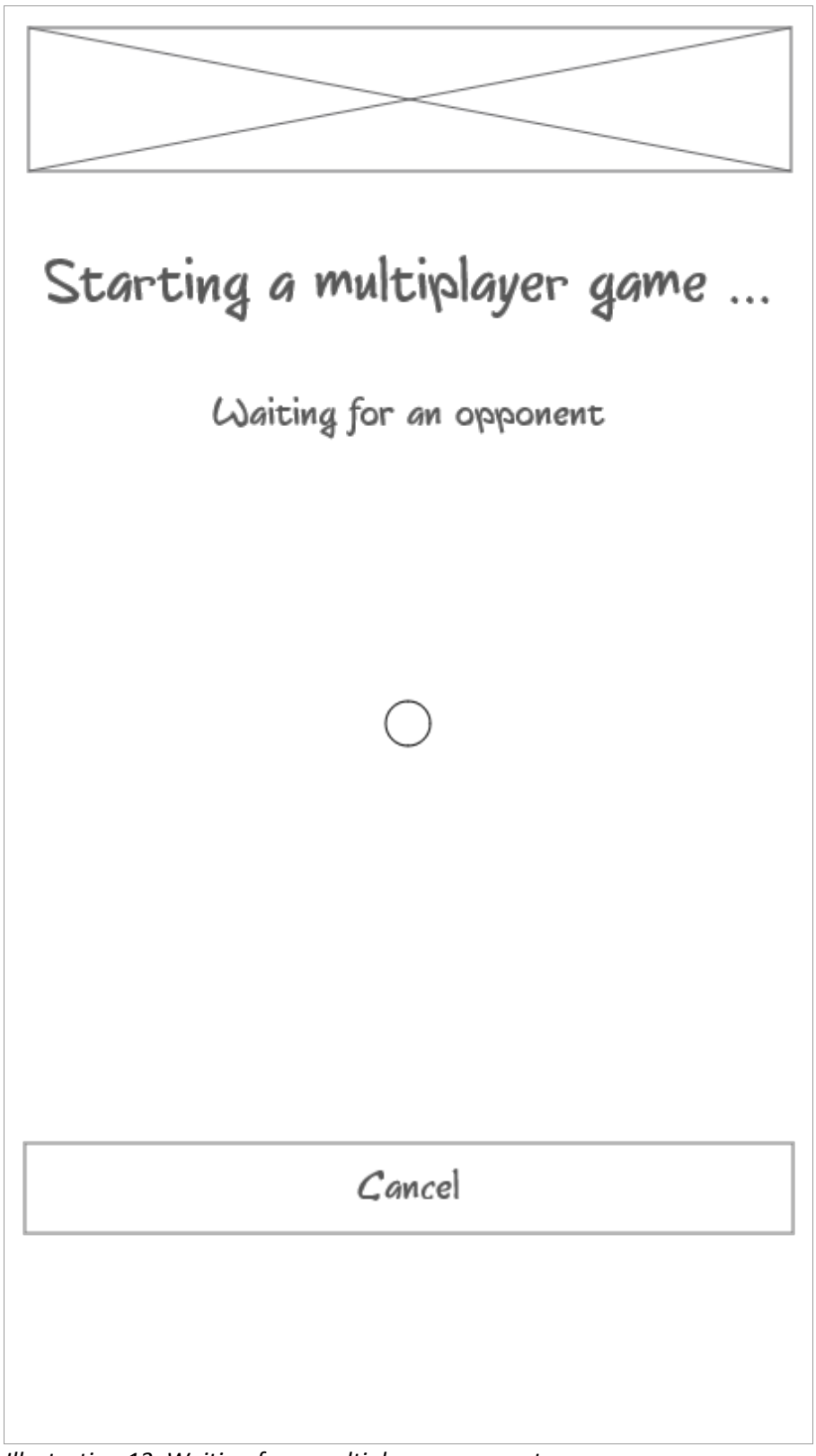

Illustration 13: Waiting for a multiplayer opponent

When a challenge is made the app proceeds to show a screen waiting for the challenged player to respond. The player can elect to rescind the challenge if the other party doesn't respond in a timely manner. If the challenged player does not have a known Mutibo account the player is asked if a message should be sent to the other player detailling how he/she can obtain Mutibo.

4.2.6 Changes during implementation of the screens

The screens have been implemented with only minimal differences from the mock-ups. Most of the back-buttons have been removed in favor of the standard android back-button.

4.3. Implementation details

Retrofit will be used as the REST-client together with OkHttp as the network layer. OkHttp will be configured to cache as much information as possible.

Google Cloud Message for Android will be used to send messages to the client when events occur that the server needs to make clients aware of as soon as possible, mostly during multiplayer matches.

We don't intend to use a local database. The potential benefits seem limited and do not warrant the increased complexity of keep the client up-to-date with the server. The required data will be loaded at application start up and kept in memory. In most cases the data will be loaded from a local cache without requiring contacting the server.

Game logic will be implemented in a separate class from the android activities.

5. Development cycle and milestones

Server	Done	Implement API's to serve information about movies, sets and decks (see chapters 3.4.2, 3.4.3, 3.4.4 and 3.4.5)
Client	Done	Implement the game logic and the required activities to play a game of multiple sets. (see chapter 4.2.3)
Milestone-1		Play a single-player game without sending data back to the server
Server	Done	Basic authentication (see chapter 3.4.1)
Client	Cancelled	Basic login screen and menu (see chapters 4.2.1 and 4.2.2)
Milestone-2		Able to login and play a single-player game.
Server	Done	Implement API's to receive results of a played game. (see chapter 3.4.6)
Client	Done	Sent results of a play game back to the server and show the leaderboard. (see chapter 4.2.4)
Milestone-3		Able to login and consult the leaderboards
Server	Done	Login using Google+
Client	Done	Login using Google+
Milestone-4		Able to login using an existing Google+ account
Server		Multiplayer games (see chapter 3.4.8)
Client		Multiplayer games (see chapter 4.2.5)
Milestone-5		Able challenge someone to a match and play a multiplayer game.
Server		Deploy to cloud platform
Client	Done	UI/UX polishing and advanced features (e.g. filtering the leaderboard)
Milestone-6		Application in a releasable state
Bonus	Done	Server/Client : Login using facebook.

5.1.1 Remarks after implementation

- Username / password authentication was cancelled when initial research showed that authentication through Google+ would not be very difficult.
- UI/UX polishing was pushed before multiplayer two weeks before deadline
- Facebook authentication was pushed forward because it was estimated to only take a few hours (estimate was mostly correct).
- Multiplayer was pushed to the last position because my work-schedule in the last weeks before deadline.

6. Mid-Point Peer Assessment Rubric

6.1. Basic

1. Basic Project Requirement:
App supports multiple users via individual user accounts

Yes, see chapters 3.2.1,3.4.1 and 4.1.2

2. Basic Project Requirement:
App contains at least one user facing function available only to authenticated users

Yes, users have to login to use the app. Most API-functions exposed by the server (chapter 3.4) are only available to authenticated users.

3. Basic Project Requirement:
App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components.

- Activity — Yes : multiple activities (see chapter 4.2)
- BroadcastReceiver — No
- Service — Yes : all communication with the server
- ContentProvider — No

4. Basic Project Requirement:
App interacts with at least one remotely-hosted Java Spring-based service

Yes, please see chapter 3.4 for an API reference.

5. Basic Project Requirement:
App interacts over the network via HTTP

Yes, using OkHttp and Retrofit (see chapter 4.3)

6. Basic Project Requirement:
App allows users to navigate between 3 or more user interface screens at runtime

Yes, the app has multiple interface screens (see chapter 4.2)

7. Basic Project Requirement:
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.**

Yes, the app plays sounds to indicate a correct or a wrong answer. After answering a set the app animates the game screen to visualize the chosen movie and the correctness of the response.

8. Basic Project Requirement:
App supports at least one operation that is performed off the UI Thread in one or more background Threads of Thread pool.

Yes, all communication with the server is performed in a background thread.

6.2. Functional

1. Functional Description and App Requirement:
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.

Please see chapter 3.3.

2. Functional Description and App Requirement:
A User should be able to log into the game using an authenticated user account.

Yes, please see chapters 3.2.1 or 4.1.2.

3. Functional Description and App Requirement:
A single game presents a series of Sets and guesses, until the User has made three Incorrect Guesses.

Please see page 22.

4a. Functional Description and App Requirement:
After viewing a Set, a User will be able to rate a Set based on the explanation of the link between the movies.

Please see page 20.

4b. Functional Description and App Requirement:
If a Set receives a large number of poor ratings, it can be removed from the game.

Sets with poor ratings will have a lower chance of being chosen by the app to be played. An admin-user can choose to remove a set with a poor rating but this will not happen automatically.

5. Functional Description and App Requirement:
For each successfully completed Set, the user will get Points.

Yes, the points that can be earned and the current score are shown on the game screen.

6. Functional Description and App Requirement:
All data (questions, answers, points, etc.) are stored to and retrieved from a web-based service accessible in the cloud.

Yes, please see chapter 3.4.

6.3. Bonus

1. Bonus* Functional Description and App Component:
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.

Multiplayer will be implemented (see chapters 3.2.5, 3.4.8, 4.1.4 or 4.2.5). Although currently this is not designed as a sudden death playoff.

2. Bonus* Functional Description and App Component:
Users could be given progressively difficult questions (e.g., based on other users' previous success with each Set).

Sets have a fixed difficulty. The set selection procedure can take this into account.

3. Bonus* Functional Description and App Component:
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.

No plans to implement this at the moment.

4. Bonus* Functional Description and App Component:
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.

Yes, players will be able to challenge friends from a social network.

5. Bonus* Functional Description and App Component:
The app could be optimized for the Amazon Appstore and could leverage Amazon’s GameCircle API to incorporate leaderboards.

No plans at the moment but if enough time remains at the end of the project implementation this might be considered as a bonus features.

7. Appendix – external resources

7.1. Development tools

Android Studio	For development of the android app	-
Netbeans IDE	For development of the server component.	CDDL
NinjaMock	For the screen mockups	Non-commercial use.
OpenOffice	Documentation	Apache License 2.0

7.2. Server

MongoDB	Database backend	GNU AGPL
Spring IO	For the REST service	Apache License 2.0
TMDb	Queried for movie information	Non-commercial use.
JUnit	Regression testing framework	Common Public License Version 1.0
Mockito	Mock objects library	The MIT License
JsonPath	A Java DSL for reading JSON documents.	Apache License 2.0
Hamcrest	Library of matchers for building test expressions	New BSD License
Commons IO	Utility classes	Apache License 2.0

7.3. App

Retrofit	REST-client	Apache License 2.0
OkHttp	HTTP client library	Apache License 2.0