General

Unless explicitly stated otherwise paths for source-files use the following root-directories :

- app : `mutibo_android/app/src/main/java/org/coursera/mutibo`
- server : `mutibo_server/src/main/java/mutibo`

Basic

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

App : starts with a login-activity to identify the user using Google+ or Facebook

- `LoginActivity.java`

Server : user accounts are stored in the associated database

- `data/User.java` : implementation of the User data element
- `repository/UserRepository.java` : CRUD-repository for users
- `controller/UserController.java` : implementation of user management API
- controller/LoginController.java : implementation of login API

No artificial limits are imposed on the number of supported user accounts.

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

App : With exception of the login related functions all other server functions that are used require authentication.

- `SyncService.java` : handles communication with the server
  - `RestClient` : interface for the REST-client implemented using Retrofit.
  - A custom `RequestInterceptor` is used to add a `X-Auth-Token` header to each request.

Server : secutiry is setup to require authentication for every request with exception of request to the login-controller and requests for resources (HTML, CSS, etc.) by the management web-app. By default the “USER”-role is required. This is overruled by @PreAuthorize annotations where required.

- SecurityContext.java : configures default security settings
- `controller/UserController.java` : an example of overriding the default security settings using @PreAuthorize.

In hindsight, it might have been a better idea to structure the API so that required roles could be deduced from the URI. For example everything requires the admin-role except the URIs starting with “/game”. But there was insufficient time remaining before deadline to make a breaking change.

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
  - `LoginActivity.java` : handles user login and authentication
  - `MenuActivity.java` : main menu of the app.
  - `GameActivity.java` : the game is played in this activity
  - `GameDoneActivity.java` : this activity is started after the game has completed
  - `LeaderboardActivity.java` : consultation of the leaderboards.
  - `AboutActivity.java` : help and support activity.
- `BroadcastReceiver` : a `BroadcastReceiver` is used in the multiplayer component of the application but unfortunatly this functionality was not completed before the deadline.
  - `game/GameControlMulti.java` : see `gcmBroadcastReceiver`
- Service : a simple service is used for all communication with the server. As this service is only intended to be used by this app no complex design (Messengers, AIDL) was chosen. Most of the public functions are designed to be called from an AsyncTask (or equivalent) because this pattern is easier to comprehend IMHO.
  - `SyncService.java` : service for with communication the server
  - `SyncServiceClient.java` : wrapper to make binding to the service simpler.
- ContentProvider : no data is loaded in memory and results of HTTP-request are cached.

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

App : all data is fetch from a remote Java Spring service and results are sent back to the server.

- `SyncService.java` : service for with communication the server

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

Yes, see setup of Retrofit in `SyncService.java` :

```
.setEndpoint(this.serverBaseUrl)
```

and initialization of `this.serverBaseUrl.`

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

Multiple screen are used to implemented the required functionality. The layouts of these screens can be found in the `layout` sub-directory of `mutibo_android/app/src/main/res`. Specific version for landscape-mode can be found in the `layout-land` sub-directory.

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.**

Animations are used in the game activity to indicate correctness of the chosen answer. When the player was incorrect the right answer is emphasized using another animation.

- GameActivity.java : animation are loaded into `animAnswerCorrect`, `animAnswerCorrect` and `animMovieCorrect` variables
- animations are defined in the files under the directory : `mutibo_android/app/src/main/res/animator`

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

The service that implements communication with the server uses a background Thread to download all the set and movie data.

- `SyncService.java` : see function `downloadDataAsync`

Several `AsyncTask` are launched on a thread-pool by various activities :

- `LoginActivity.java` : authentication and token requests are done in the background :
  - class `GoogleAuthTask` : request a token from Google+ to authenticate
  - class `FacebookAuthTask` : request a token from Facebook to authenticate
  - class `FacebookRevokeTask` : revoke access to Facebook from the app.
- `GameActivity.java` :
  - class `DownloadPosterTask` : loads a movie-poster from the server and displays it
- `LeaderboardFragment.java` :
  - class `LeaderboardRetrieveTask` : retrieve data from the server for the leaderboard
- `NewPlayerActivity.java` :
  - an `AsyncTask` is used to change the nickname of a new player.

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.

On the server :

- data/MutiboMovie.java : info about a single movie (id, name, year, plot summary)
- data/MutiboSet.java : info about a single set :
  - goodMovies[] holds ids of the three movies that fit together
  - badMovies[] holds the id of the movie that doesn't fit in
  - reason : explanation of the relationship between the three related movies
- data/MutiboDeck.java : an extension to specification to allow easy synchronization of groups of sets to the client app.

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

In the app :

- `LoginActivity.java` : The user can choose to login using Google+ or Facebook. An access-token is obtained from the social-network and sent to the server for validation. When validated the server returns a token that is used for the rest of the session
  - class `GooglePlusClient` : functionality to sign in to Google+, sign back out and revoke access to the account on Google+
  - class `GoogleAuthTask` : login to the server with a token from Google
  - class `FacebookAuthTask` : login to the server with a token from Facebook
- `SyncService.java` : communication with the server

On the server :

- `WebAppContext.java` : sets HTTPS on port 8443
- `controller/LoginController.java` : implementation of the login API-functions
- `google/GoogleTokenChecker.java` : validate a Google-token
- `facebook/FacebookTokenChecker.java` : validate a Facebook-token
- `security/TokenHandler.java` : generate and validate our security tokens
- `security/TokenAuthenticationService.java` : bridge between Spring Security and our security tokens
- other classes in the security package implement required Spring Security functionality

The authentication implementation is based on the blog post “Stateless Spring Security Part 2: Stateless Authentication” on JDriven (see http://blog.jdriven.com/2014/10/stateless-spring-security-part-2-stateless-authentication/)

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

In the app game logic is implemented in `game/GameControlSingle.java` for a single-player game. The function `chooseNextSet()` determines the next set to play and randomizes the movies in the set. Function `answerSet()` receives the user's choice and determines the correctness of the guess and the effect on the remaining "lives" of the player and the state of the game.

The `GameActivity` (in `GameActivity.java`) takes care of the visualization of the game state and the interaction with the player. Different fragments are shown depending on the state of the game.

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.

In the `GameActivity` after the player makes his choice the `GameDoneFragment` is shown. Not only does this fragment visualize the explanation of the link but it allows the user to rate the Set by assigning a number of "stars".

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

The ratings given by the user are sent back to the server together with the results of the game. These rating are visualized in the administration web application provided by the server and an administrator can decide to remove low rated sets. Sets are not removed automatically.

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

See function `answerSet()` in GameControlSingle. When the user chooses correctly the point value of the set is added to his current total score for the game.

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.

On the server the data is stored in a Mongo-DB database. The classes that describe the data elements can be found in the `data` sub-directory. Repository interfaces to the database are located in the `repository` sub-directory and the controllers that provide web-base access to this data are in the `controller` sub-directory.

The app uses a Retrofit-based client to access these functions. This is implemented in the `SyncService` class.

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.

Multi-player matches were planned as feature and partially implemented. But insufficient time was available to properly implement this feature before deadline.

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).

Not implemented at the moment. Sets do have a difficulty-field that could be taken into account when selecting the next set to play. Taking a users' previous success into account will be harder because that data is not stored on the client.

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.

Not implemented.

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.

This was planned as part of the multi-player feature and a large part of the decision to implement Facebook authentication as an alternative to Google+. This feature could not be finished in time for the deadline.

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.

Not implemented. Leaderboards are provided through our own server.