Coursera Android final project
October 2015

Got It! Project

Table of contents

Summary

This project is composed of two complementary parts, a Spring based server, and an Android based client. The following document will attempt to explain the current implementation plans, and the current progress in both, with the help of the diagrams provided along with it. The requirements for the project are set as follows, with entries marked in italics already implemented:

- A *Spring based server*, and an Android based client.
- *At least two types of user privileges: Post quiz, follow another user.*
- *There must be at least one operation only available to autenticated users.*
- *Users must be able to allow (or disallow) the follow relationships.*
- At least one advanced capability must be implemented (pending inclusion of profile photo for users)
- *Must use at least two fundamental android components.(Activity, Service. BroadcastReceiver if time allows).*
- Users must be able to navigate through at least three user interfaces (pending implementation of a fragment-based approach for UI1: register new user, UI2: login screen, UI3: logged user profile info, UI4: user list of those who allow followers, UI5: quiz screen, UI6: notifications list screen, UI7: feedback screen. A test activity for the service and process class is already implemented to allow for ease of debugging.).
- *Background operations must be implemented.(All network requests are done through HaMER and callbacks in the service and it's listeners)*
- *User data object for teens which define certain basic properties of the user.*
- Reminders must be send through notifications to the teen (checking the 'POST_QUIZ' grant the 'teen' users have to know if the reminder has to be created or not.)
- Feedback mechanism to return information about the teen quiz results to himself.
- *Follow mechanism so authorized users (with privilege "CAN_FOLLOW") will receive notifications on polling the server about new quizzes submitted by the teens they follow and who allow their quizzes to be seen.*
- *Teens can choose to share or not data.(Currently implemented as a boolean to define if the user can be followed, or not.)*
- *HTTPS will have to be used when disseminating data of teens for security reasons (HTTPS is used for everything)*

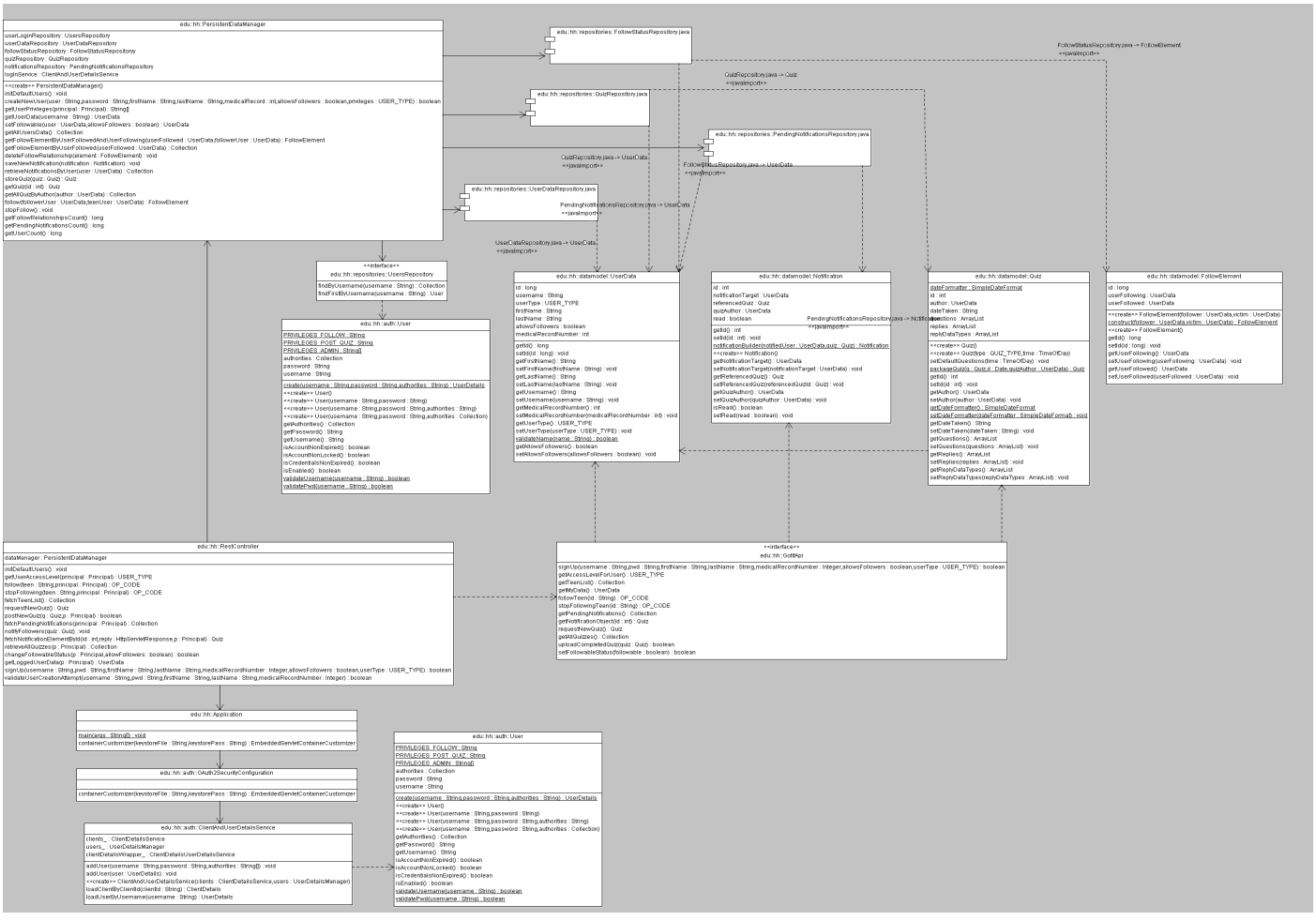
Server

The server is mostly done, with junit tests showing a full-green result. The server can handle new registrations, polling for notifications, requests for new quizzes (with hard-coded questions for now), accept anwered ones, and manage follow relationships and privileges, plus destroy any follow relationship once the teen decides he doesn't want to share data anymore.

HTTPS has been implemented for all rest points, and authorization and proper privileges are required to access any funcionality other than registering a new user. OAuth2 is used to handle authentication attempts and disseminate access tokens.

Persistence is implemented by the use of JPA for users, notifications, follower relationships, and quizzes. Relationships have been enstablish between all these entities to guarantee relational integrity. The only entity which is persisted and does not have JPA based relational integrity with the rest is the User (authentication) object. For security reasons, I decided against linking that object with the UserData object which holds non-critical user data like name and last name. The "link" of both objects is done through declaring a username variable in the UserData object which points to the username in the User (authentication) object, and using java to link them both when required, which isn't very often.

The following diagram (available in png format in the same folder as this document) holds an overview of the server structure.



Client

The client will be divided in three main diferent parts:

- Activity: Will hold references to both headless and graphic fragments which will be what will define the actual UI which the user will see. Also will manage which menu options are available for each screen, which may differ according to available time.

- Fragment: Fragments will be our UI units. Every time the user switches screen, what he will be doing is swapping a fragment for another, instead of killing and creating a whole activity. This approach has been taken due to the ease of implementation, and performance improvement and activity-server connection persistence and stability. The minimum number of fragments which we will have will be: Login, register new user, take a new quiz, get feedback data, find users to follow, and check own profile. All of these fragments will only be available to the user upon a successful login. Some of them (like post quiz, or list of users to follow) will only be availabe to users with the proper privileges defined in their UserData. Nonetheless, should anyone create a client without such restriction, or discover how to access those fragments in a malicious way, the server of course also authenticates the clients and guarantees the user has the privileges required to take any action.

- Service: The service will be tasked with the following:
  - Network connections: It will make all tasks related with client-server communication, and use callbacks through a listener interface (which the main application must be registered to) to send data and status information.
  - It will also be tasked with generating notifications on proper times for the users using timers, both for checking for new notifications (if time allows it) and for reminding the "POST_QUIZ" allowed users to post a new quiz at user-defined intervals.

Some decisions had to be made when designing the client. The "Feedback" data will be client-generated. The server will send some quizzes from the user if available (or all) so the client can process the information and show it accordingly. This gives us two main advantages: first, if we want to add some kind of new analysis into our feedbacks, it's more likely we won't have to touch the server at all, incurring in less downtime for updates and maintenance. Second, this allows third parties (in the case we make our API free) to implement their own analysis tools for the teen data. The drawbacks are bigger network usage on the client, which is taken as a non issue due to the current quotas most networks provide, and on the server, this last one being somewhat compensated by the reduced processing requirements as the data is not processed server-side.

Conclusion

The conclusion is that I'm going to have a hard time finishing everything, as I decided to implement every part almost from scratch. So, if you believe in a god, please pray for me. If you don't, with me luck.

Thanks for your patience and dedication, and I wish you good luck, too!