Bibbla Developer's guide

Version 0.2
Arild Matsson, 2012-10-14

This document is definitely subject to change, and might be slightly outdated. Contact Bibbla developers for help.

Acquiring source

Clone the GitHub repository.

```
git clone git@github.com:aommm/bibbla.git
```

The repository contains three Eclipse projects for the app, additional libraries, and unit testing respectively. Import the projects to Eclipse.

Setting up Eclipse

The projects must be named accordingly: Bibbla (app), ActionBarSherlock (libs) and UnitTesting (tests). Note that the UnitTesting project lies in a subdirectory in Bibbla.

Ensure that *Compiler compliance level* setting is set to 1.6 in the properties of all projects.

Note: Even though the UnitTesting project does not make any use of the ActionBarSherlock project, the latter must be present.

Project design

Please see the *Arkitektur-specifikation* document for a general understanding of the software architecture.

GUI

For the implementation of a tabbed user interface, we utilise the ActionBarSherlock library. Most of the GUI is implemented in Fragment classes. A single activity, MainActivity, contains all these and handles changing between fragments.

Fragments can have many different sizes and so take up only a selectable item or a whole screen. Most of the fragments used in this app take up the phone's main area, practically defined by the screen minus the notification tray, software buttons and this app's tab bar.

Backend and jobs

The backend of the app communicates with the Gotlib system. Since there is no API available, the application scrapes HTML from the web server. Towards the application frontend, the backend exposes itself in the form of a façade class. In the hypothetical event of changes in the remote system, i.e. HTML output changes drastically or an open API is introduced, the backend may be easily replaced, with no changes needed in the frontend.

To deliver results back to the frontend, we cannot use ordinary return values because significant relay times would cause the GUI to freeze. Instead, we use Callback objects. When calling a façade method from frontend, a new Callback is passed as a parameter. When the result from the backend is ready, the backend manipulates the Callback object and calls a method in the frontend. (Circular dependency is avoided here through polymorphism).

Parsing HTML is done by means of the library JSoup. If you have a basic understanding of HTML or XML, this code is quite easy to understand, especially with help from the JSoup API reference.

When logged in, data from session cookies is stored in a Session object which is held by the façade class and passed to the constructor of every new job class that utilises it. (Some jobs, like searching for a book, don't require being logged in and therefore doesn't make use of sessions.)