Development Manual

Installation

1. Fetch the Git repository from https://github.com/TeamLegendaryAwesome/WYSIWYD.git
2. Download the Android SDK and install it. Preferably use Eclipse and install the Android plugins. You should also create an Android Device and make sure that Java JDK is installed.
3. Import the Git repo into Eclipse and start to develop!

General

Our program source structure is based on which type the class is. We are trying to follow the MVC model and therefore we will have one package with the Controller parts, one package with Model-specific classes and then one “View”-package.

There are classes that handle the connection to our SQL database that will be downloaded to our local database on the android device. The database backend is based on MySQL.

View

The view package contains all of the activities and classes that pertain to the layout of the app. Most of the classes are activities and are visible to the user. These activities are very straight forward and incorporate a list view, a list refresh method, and pass intents with drink information to another activity used to view the drink recipes. Some of these files have a layout based on custom XML files, where we’ve defined how they should look.

Some of the files in the view package are not activities, but are useful to correctly populate our lists with dynamic information located in our database. These classes are custom adapters that extend ArrayAdapters. These adapters get drink or ingredient information from the controller classes, and then create a list view to put them in.

Controller

We have tried to implement the Controller as a MVC Controller object. The object is implemented in the se.turbotorsk.mybar.controler package. When the View needs to update, get new data or save data the call is made to the controller. The controller then gets data from the model or uses methods in the controller to sort and filter drinks.

Model

The model package contains all the I/O code the application needs. We choose to use the class Data.java to be an “interface” for the whole package. This is a good design the Controller does not need to know anything about a specific I/O type and we do not have to change the Controller if we change the way we implements how the I/O is implemented. We can also use the Data.java class to fake functions in the model package

MyBarContentProvider

This layer lies in the Model package and provides all content to Controller. It contains a SQLite database for local data. The content pushed to the activities may come from either the internal database or from the network. The activities don’t know and don't care if the data is from SQLite or the external MySQL database.

External data

Do be able to add new drinks and ingredients to the application we have chosen to use remote MySQL server to store the main drink database onto. The data is then synced to the application during the start.

The SQL server contains two tables, Ingredient and Drink. All the drinks can be identified either by ID (Integer) or by the Name of the item. Both of these is unique.

We chose not to implement a direct SQL connection to the MySQL because of the risk of SQL injections. Another good reason not to use SQL is that the application needs to interact directly with the SQL engine. Instead we use a JSON-data structure to send data.

The JSON-formatted data

We chose to use JSON because it is supported by Android and is widely used in application that gets data from external applications and data source. It is also human-readable which makes it easy to debug.

The JSON code is generated by PHP. The PHP quires the database and creates an array of the dataset. The array is then converted to JSON by the PHP command “json_encode($array);”.

If the load on the SQL server increases by a growing dataset and more users we van cash the JSON page and eliminate unnecessary SQL quires.

We can also implement functions that checks if the data really has to be updated. In version 1.0 all the data gets sync when the applications start.

Apache 2

The JsonParse.java class receives the JSON data is then echoed in to a HTML document. The class preforms an http-get and recvies the JSON in a text string. This text is then parsed with the JSON-reader and inserted in the SQlite database.