Architecture Description

of the Croud Trip project

1 Server architecture

We use *DropWizard 0.8.0* [1] as our main server framework and build the server application as a war-file using *WizardInABox 0.8-1-1* [2] that is deployed on the chair's tomcat server. DropWizard uses the Apache v2.0 license.
DropWizard as a server framework provides us:

- *Jersey* to provide a RESTful Web service
- *Jackson* as a JSON converter
- *Metrics* for behaviour measuring

On server side we use additionally *Google Guice 4.0-beta5* [3] as our framework for dependency injection (Apache v2.0) and *Hibernate 4.3.5* [4] as object-relational mapper (LGPL v2.1).

2 Client architecture

On client side we use Android SDK version 21 as target platform and support version 16 as minimal SDK version.
As REST client we use *Retrofit 1.9.0* [5] (Apache v2.0) which allows us simple access to the REST service provided by our server. It gets even more simple by using *JaxRs2Retrofit 0.3.0* [6] (Apache v2.0, written by one of our team members) which automatically generates classes for the REST client from the given Jersey classes. So we can use the same classes for the REST service and the client.
As object-relational mapper on our Android client we use *ORMLite 4.48* [7] (Creative Commons Attribution-Share Alike 3.0 License). To save some additional work we want to use the Java Persistance API to share classes for server and client database tables.
To improve the workflow even more we are currently evaluating *Timber 3.0.1* [8] (Apache v2.0) as a advanced logging tool and *RxJava 1.0.8* [9] (Apache v2.0) for reactive programming.
To get a clean and modern UI design on android devices we use additionally *FloatingActionButton 1.9.0* [10] and *MaterialNavigationDrawer 1.3.3* [11] (both Apache v2.0)
Since we have to provide route computation and evaluation for our application we use the *Google Play Services v7.0* [12]. From these we mainly use the Google Maps APIs. To extract the downloaded route we use the *Google Maps Android API utility library (v0.3.4)* [13]. We plan to do our route computation on the server and the client has to send a request to get this computed route. So we do not access the Directions API directly from the client, but from the server using *Google Maps Services for Java (v0.1.6)* [14].

3 Other tools

As programming language we use Java 7 for both client and server. Our project is hosted on Github [15] and we use Travis CI [16] as continuous integration tool that builds our project after every push and automatically deploys our releases to Github.
We decided to use gradle 2.3 [17] as build automation system.

[1] http://dropwizard.github.io/.

[2] https://github.com/rvs-fluid-it/wizard-in-a-box.

[3] https://github.com/google/guice.

[4] http://hibernate.org/.

[5] https://github.com/square/retrofit.

[6] https://github.com/Maddoc42/JaxRs2Retrofit.

[7] http://ormlite.com/.

[8] https://github.com/JakeWharton/timber.

[9] https://github.com/ReactiveX/RxJava.

[10] https://github.com/futuresimple/android-floating-action-button.

[11] https://github.com/neokree/MaterialNavigationDrawer.

[12] https://developer.android.com/google/play-services/index.html.

[13] http://googlemaps.github.io/android-maps-utils/.

[14] https://github.com/googlemaps/google-maps-services-java.

[15] https://github.com/AMOS-2015/amos-ss15-proj2.

[16] https://travis-ci.org/AMOS-2015/amos-ss15-proj2.

[17] https://gradle.org/.