RESTful service with charts

Hammer Clemens MSc
hammer@technikum-wien.at

University of Applied Sciences Technikum Wien
Department of Embedded Systems
May 22, 2017

1 REST Service

This application deals with the realization to get blood pressure information from the **ViTAL Server** and deploy new information on the server. The information exchange is done over HTTP calls like GET and POST. Further the information format is defined as JSON string. Therfore the data from the server or the app must be parsed into the right format and if necessary displayed.

Therefore some activities are needed, which handle this task(s). Expampe layouts are shown in chapter 3 and available as XML files over Moodle. If you deside to create your own layout the following components are required for minimum:

- ListView
- EditText
- Button

The functionality of this application should include the following tasks:
If the user presses the *GET* button, the application should request the information from the ViTAL Server. Further the URL for this call should be read back from a EditText field.
The received data should be parsed to the provided data type (*Bloodpressure.java*). With the knowledge that this data could be received as a list of data, the collection should be displayed by the ListView.

However, not only the read functionality should be implemented, further it is necessary to implement a write functionality. Therefore the app should be enhanced to provide the opportunity to post the vital data to the server.
This functionality should include the following tasks:
If the user presses the *POST* button, the application should read the vital data from the current screen and post them to the ViTAL Server. If this is done correctly the user should be informed, for example via a Toast message.

To prove if your application works correctly, the posted data can be read back by the previous implemented activity.

1.1 Optional:

Optional extend your application by an additional activity, which displays the received data not only as a listview, further display the information as a chart. (For details see chapter 4)

1.2 Possible procedure

To realize such an application, structure your task into several subtask:

1. Mock-up:
   - Implement every graphical element of your design without any functionality
   - Start your application and see if everything is visualized as intended
2. JSON parsing: [30 %]
   - Create a package called Bloodpressure
   - Add the provided class *Bloodpressure.java* to the package
   - Add a new class called BloodpressureParser to the package with functionality from the UML diagramm in section 2.1
   - Add hard coded test strings to the parser to test the implemented functions.
3. AsyncTask:
   a) HTTP GET: [35 %]
      - Implement a new class which extents *AsyncTask*
      - Extend the functionality to add N dummy values from type *Bloodpressure* to a list. After the task finish, visualize the data on the listview
      - Exchange the dummy function with the HTTP GET functionality
      - Add the parsing function to convert the JSON values to the *Bloodpressure* class

   b) HTTP POST: [35 %]

      i. Same class:
         - Enhance the previous implemented functionality by the POST mechanism

      OR

      ii. Separate class:
         - Implement an additional class with the functionality like the previous implemented class
         - Exchange the GET with the POST functionality
4. Charts: [20 %]
   - Implement an additional activity with displays the vital data as chart (see chapter 4)

2 Hints

2.1 Implementation

URL inside Edittext
The URL of the server can be deployed as string inside the EditText field. Prevent hard coded URLs inside the code. If the IP address of the server changes, you only need to edit this field instead of recompiling your source code.

JSON parsing

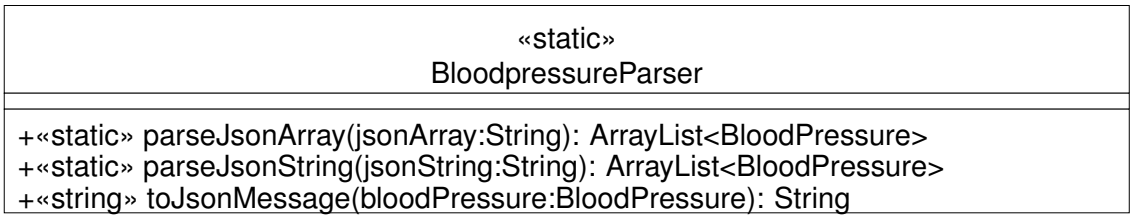

Figure 2.1: UML diagram BloodpressureParser

2.2 Web links

https://developer.android.com/reference/android/os/AsyncTask.html

http://www.mkyong.com/webservices/jax-rs/restfull-java-client-with-java-net-url/

http://blog.appliedinformaticsinc.com/sending-json-data-to-server-using-async-thread/

3 Example Layout

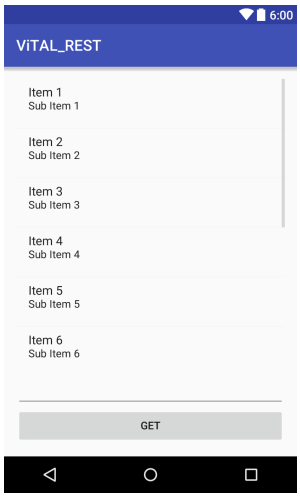

Figure 3.1: Layout for the get activity

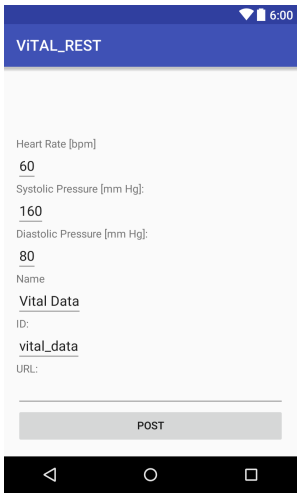

Figure 3.2: Layout for the post activity

4 Charts

This application deals with the realization of the visualization of information via charts. The information itself should be the requested blood pressure information from the **ViTAL Server** (see App REST service).

To realize such charts use an additional library called **MPAndroidChart**.
To use this library inside the project, following the three steps:

- Download the latest .jar file from the releases section
- Copy the .jar file into the libs folder of your Android application project
- Start using the library

In our case we can download the .jar file from the git repository (https://github.com/PhilJay/MPAndroidChart/releases).
After copying the .jar file we have to declare inside the Android Studio, that we want to use an external library (see LocationProvider Assignment).

4.1 Possible procedure

For our new application we are using the previous implemented *REST service* application and extend the visualization. The visualization should be enhanced from a classical textual form to a graphical form (chart).
Furthermore the following values should be displayed as graph:

- diastolic_pressure
- systolic_pressure
- heart_rate

In the provided example a so called BarGraph is used. Every different chart type uses a corresponding datatype, which is a subclass from the DataSet class. In this case we use the *BarDataSet* class to store the information, which should be displayed.
Inside this set every data point is stored as an Entry, which represents the data itself and a corresponding index.
The visualization of the three values should be combined into one graph and eligible via three switches. Further an additional button (*GET*) should be realized, which requests the data from the server. Additionally the X axis, which represent the date, should be scalable

as provided in the example application.

The functionality itself should include the following tasks:
If the user presses the *GET* button the application should request the information from the ViTAL Server. Afterwards the data should be parsed to the provided data type (*Bloodpressure.java*).
Depending on the activated switches, each type of value (*diastolic_pressure, systolic_pressure heart_rate*) from the requested history should be displayed as a separate bar.

4.2 Hints

Examples can be found inside the repository: https://github.com/PhilJay/MPAndroidChart