Documentation - Milestone 2

Apache CXF

The implementation of the REST and SOAP Service was done with the Apache CXF library. We added it through Maven and the Servlet is loaded in the web.xml of our project. We added a beans.xml just in case if we ever decide to do a Beans/Spring implementation of our project.

REST

We expanded our AlarmClockService (ACS) with GET, POST and DELETE REST methods. They are available at “localhost:9000/acrestservice/{methodpath}” after the server was started and finished booting up.

For GET there are two APIs. One to get every created alarm so far and one to get a specific alarm. “/getalarms” produces a JSON filled with an array of dates in milliseconds that are set alarms and returns it to the REST client if the system runs correctly, that means the ACS was initialized. The HTTP status code is 200 (OK) if everything goes correctly, else 404 Not found will be returned. For "/getalarm/{msdate}" a parameter is required to get the specific alarm. “msdate” here is representing a date in milliseconds. The service will return also a JSON structured String with dates in milliseconds but additionally it has the info if the alarm was already started (and will go of at that date).

The POST method adds a new alarm and starts the counter in the ACS. It’s accessible through the path “/postalarm” and needs a form parameter named “msdate” that has a date in milliseconds saved. It will return the HTTP status code 400 (BAD REQUEST) if the parameter is not a number or if an error happened while adding the alarm. If all went well the code 201 (CREATED) will be returned with the location of the newly added element.

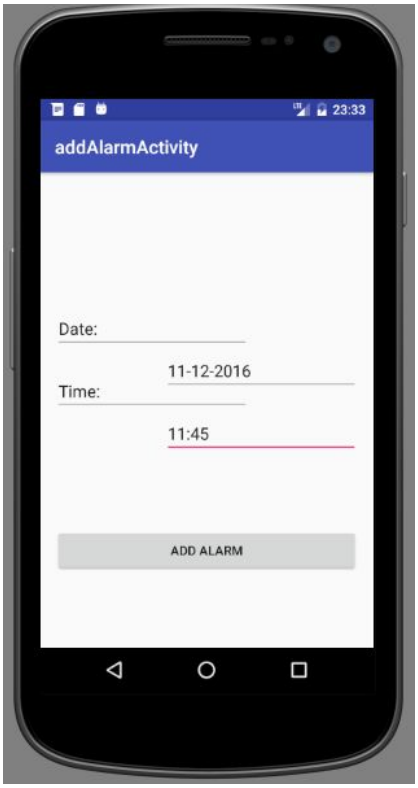
Last but not least there’s a DELETE method that removes an alarm that was previously added. Like the POST method this one needs a form parameter named “msdate” that has a date in milliseconds stored. It’s reachable through the path "/delalarm" and returns HTTP status code 400 if the parameter is not a number, 404 if the alarm wasn’t found or 200 if the alarm was removed succesfully.

All these methods and informations for Apache CXF are implemented in the class ACRestService. It needs a working instance of the ACS to communicate with it to change the state of the alarms (i. e. add a new alarm) because there is no working persistent data storage implemented for now (Milestone 3). The service is instantiated in the IOTRunner class that is called during the server startup.

Android REST Client:

To install this client the SDK version should be at least 17. The code was compiled on Android Studio v2.2.2 using SDK version 23. But if you fulfill minimum SDK requirements the app should just run like a charm.

On startup the client directly connects to the application servers REST service and polls all alarms from connected alarm device using HTTP get on port 9000, and presents the alarms retrieved from the GET response. The connection is handled by the class AsyncRESTClient. It has only one method "request" which is to take care about method (get, post, delete), data sending and retrieval, as well result code delivery.
At this stage the current clock time presented comes from the mobile device or emulator. But will show the current time from the alarm device as soon the function to retrieve the "server" time is available in next milestone.
As shown on screenshot below, there are already two alarms on the device. Using the grey "+" button another activity will be started which allows to enter a new alarm time (as shown right). The input format is ddmmyyyy for the date field, and hhmm for the time field. Nice looking date and time pickers shall be used in final release.
When "ADD ALARM" is pressed, the alarm time (in milliseconds) is sent using HTTP post in RESTfull manner ;) , and application switches back to the main activity.



Alarms can also be deleted in main activity from the alarm list, using the dustbin icon to the right of the alarm. Once the icon gets clicked an HTTP post is send towards the application server, requesting the resource deletion and refreshes the alarm list after receiving the HTTP response. In case of an error response the HTTP response code will be toasted to inform the user about the bad situation.

SOAP

For SOAP, we decided to expand our AlarmClockService with SOAP-driven methods for retrieving, adding and deleting alarms. They are available under the URL "localhost:8080/ACSoapService/" after being instantiated by our class IOTRunner.
As there was not enough time to write the SOAP service from scratch using WSDL (technical difficulties), we decided to take a "code first" approach via a annotated SEI.

All SOAP-specific classes lie in the package IOT.IOT_SOAP.
The class "IACSoapService" is the SEI (service endpoint interface). It contains four fully annotated methods for interacting with the underlying AlarmclockService, which are implemented in the class "ACSoapService". The following methods are currently supported:

- getAlarms() … returns all currently saved alarms
- getAlarm(String msTime) … returns a certain alarm as specified by msTime
- postAlarm(String msTime) … allows to add an alarm to the AlarmClockService
- delAlarm(String msTime) … allows to remove an alarm

The method getAlarm(String msTime) returns a HashMap<Long,Boolean> object which the SOAP parser cannot handle, thus there are two classes acting as wrapper: LongBooleanHashmap and LongBooleanHashmap Adapter. They are bound to the getAlarm function via a @XmlJavaTypeAdapter annotation.

General

As in the last milestone, the number of commits on GitLab says nothing about the amount of work that went into the project. For this milestone, we encountered many technical problems (marshalling, deployment, choosing the correct dependencies etc.), but most of our time was invested into researching and understanding Apache CXF, as well as the concepts of REST and SOAP itself. Also, we spent a long time considering how to best integrate them into our current project.

In the end, we divided work like so:

Aichinger Mara	SOAP services
Balaz Melanie	Melanie wasn't in Austria and thus not available for work on this milestone.
Eichinger Rene	REST android client
Klosowski Peter	REST services

The following hardware/software/frameworks have been used:

- Raspberry Pi 2
- Plantuml / Visual Paradigm 13.1
- Eclipse 4.6.1 / IntelliJ IDEA 2016.1.4 / Android Studio 2.2.2
- Java 1.8.0_111
- Android SDK 17 (tested with 23)
- Apache Tomcat 8.0.38
- Maven Dependencies:
    - GSON 2.7
    - Java Servlet API 3.1.0
    - Jersey Core Server 2.23.2 (Server & Client)
    - JUnit 4.12
    - Pi4J 1.1
    - Apache CXF 3.1.8
    - Spring 4.3.4