Bulletin of the *Transilvania* University of Braşov • Vol 6(55), No. 2 - 2013
Series III: Mathematics, Informatics, Physics, 51-60

JAX-WS LOGIN WEB SERVICE AND ANDROID CLIENT

Constantin ALDEA[1]

Abstract

In this paper the steps made to send requests and receive responses between an Android client application and a JAX-WS[2] login web service which is hosted on the JBoss application server are presented.

The base architecture for an enterprise application is presented and the database access mechanism is shown by implementing the authentication mechanism for a user connected to the mobile device by using the login web service. In this paper are presented the steps need to be done to send requests from an Android client application to a web service which is hosted on the JBoss application server.

2000 *Mathematics Subject Classification:* 68N19.
Key words: web service, mobile device, security.

1 Introduction

In this paper a communication strategy between the mobile device and the application server is presented. Often it is required that a mobile client application should access data stored in a database. More than accessing the database tables the developer wants to use some of the enterprise application modules that have generated the data.

The following use case is required. An existing application has a web user management module. By using the user management module users are created, modified, activated or deactivated. Using an existing user the customer has access to the other application functionalities (e.g. reports). In the classical way the customer accesses the web page and uses its username and password, logs into the application and uses the application functionalities [1]. Some functionality (reports, news, etc) must be ported to be accessible on the mobile devices. While the web interface of the enterprise application cannot be customized to be used on the mobile device a new client application is required on the mobile device.

The architecture for the communication between mobile client device and the enterprise application is drawn in figure 1. The following components are identified:

[1]Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: costel.aldea@unitbv.ro

[2]Java API for XML Web Services

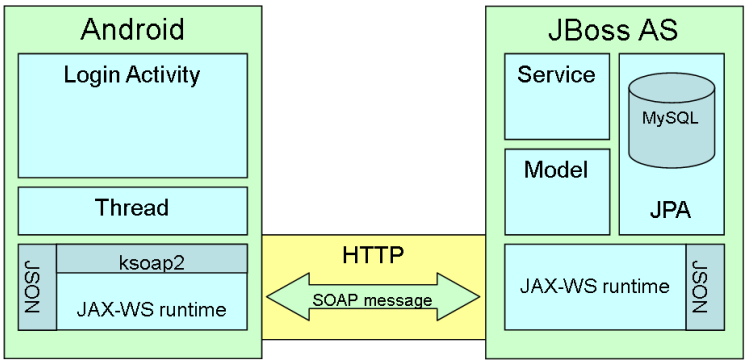

Figure 1: Application architecture

- application server - in this case JBoss - it runs and publishes the web service.
- web service - JAX-WS login web service - waits for SOAP[3] envelopes containing request and provides answer envelopes.
- JSON[4] module that encodes and decodes JSON data (readable texts used for data exchange between applications).
- model - module used for ORM[5] data representation.
- service - module that implements methods for managing interaction with data.
- JPA[6] - provides relational persistence of data - in this article the connection is made with MySQL database server.
- ksoap2 - represents a SOAP client library for the Android platform.
- thread - in this case it represents only a single thread for performing the network operations (it is mentioned while if the application is extended it is necessary to control the network flows more generally).
- Login activity - implemented Android application's lifecycle component.
- Android - mobile device client.

2 Main results

2.1 Prerequisites

To implement the proposed architecture the following prerequisite installation steps should be done:

[3]Simple Object Access Protocol - XML based protocol
[4]JavaScript Object Notation
[5]Object-relational mapping
[6]Java Persistence API

1. java installation - Java Platform (JDK) 7u45 [5].

   The environment variables $JAVA\_HOME$ and JRE_HOME must be created and concatenated to the environment variable $PATH$.

2. eclipse installation - download and unzip (e.g. Kepler). Maven plugin will be installed by using "eclipse marketplace" menu option.

3. android sdk - download and setup [3].

4. installation of the eclipse android plugin [4].

5. apache maven - build automation tool for Java projects [9]. Note that for maven the local repository must be set otherwise the user profile folder will be used.

   To start the mvn tool the environment variable $maven = pathToMaven$ must be created and then added to the path variable ($PATH = \%path\%$; $\%maven\%\backslash bin$).

6. JBoss installation - application server [7]. After unzip the environment variable $JBOSS\_HOME$ must be created and concatenated to the environment variable $PATH$ (e.g. jboss-eap-6.2).

In the standalone.xml configuration file of the server the data source must be defined. To define the data source the driver module must be installed (in this case mysql driver). The following lines must be added into the standalone.xml configuration file of the application server:

```
<driver name="mysql" module="com.mysql"/>
```

into the section drivers and the following lines into the section datasources:

```
<datasource jta="true" jndi-name="java:jboss/datasources/butAndroidDS"
pool-name="my_pool" enabled="true" use-java-context="true" use-ccm="true">
   <connection-url>jdbc:mysql://localhost:3306/butAndroid</connection-url>
   <driver>mysql</driver>
   <security>
      <user-name>dbuser</user-name>
      <password>dbpassword</password>
   </security>
   <statement>
      <prepared-statement-cache-size>100</prepared-statement-cache-size>
      <share-prepared-statements>true</share-prepared-statements>
   </statement>
</datasource>
```

The mysql driver isn't installed default such that it must be installed as module by making the following steps [10]:

1. download the mysql connector mysql-connector-java-5.1.28.zip [11].

2. create the subdirectory *%JBOSS_HOME%\modules\com\mysql\main* and unzipp into it the connector.

3. create the module.xml file into the same directory

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
   <resources>
      <resource-root path="mysql-connector-java-5.1.28-bin.jar"/>
   </resources>
   <dependencies>
      <module name="javax.api"/>
   </dependencies>
</module>
```

If the module was installed successfully the application server can be started.

2.2 Login web service

The login web service is a function that receives two parameters (username and password). The service checks (using JPA) into the database the rights of the user. If the user is found and the password is ok, a JSON response is created.

The login web service has the general structure of a web service [2]. Besides its scope this web service accesses the entity manager of the application server through the CDI[7] mechanism.

To create the login web service the follwoing steps need to be done:

1. Create a maven web project using eclipse.

2. Add the dependencies into the maven *pom.xml* file (org.json for working with JSON objects, hibernate-jpa-2.0-api for database persistence, javaee-api for dependencies injection).

```
<dependency>
   <groupId>org.json</groupId>
   <artifactId>json</artifactId>
   <version>20131018</version>
</dependency>
<dependency>
   <groupId>org.hibernate.javax.persistence</groupId>
   <artifactId>hibernate-jpa-2.0-api</artifactId>
   <version>1.0.1.Final</version>
</dependency>
<dependency>
   <groupId>javax</groupId>
   <artifactId>javaee-api</artifactId>
   <version>7.0</version>
</dependency>
```

[7] Contexts and Dependency Injection

3. Create the database model.

```
@Entity
public class User implements Serializable{
   @Id
   @GeneratedValue(strategy=GenerationType.AUTO)
   private Integer id;
   @Column(unique = true)
   private String username;
   // ...
```

With the @*Entity* annotation, the connected database knows to format that class into an object and save it to the database. The connection to the valid database is established through the *persistence.xml* file where the class is mapped.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="myDS">
    <jta-data-source>java:jboss/datasources/butAndroidDS</jta-data-source>
    <class>acl.but.model.User</class>
    <properties>
      <property name="eclipselink.jdbc.cache-statements" value="false"/>
      <property name="eclipselink.jdbc.native-sql" value="false"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

4. Create the web service interface. The web service is annotated using the @*WebService* annotation.

```
import javax.jws.WebMethod;
import javax.jws.WebService;
@WebService
public interface Login {
    @WebMethod String doLogin(String username, String pwdHash);
}
```

5. Implement the web method in the web service class. By using the userservice the input parameters are checked and the query response is packed into a JSON object.

```
   JSONObject object = new JSONObject();
   // ...
   User user = userservice.find(username, pwdHash);
   if (user != null) {
      dbusername = user.getUsername();
```

```
      dbpwdHash = user.getPassword();
      try {
         if (dbusername.equalsIgnoreCase(username)
               && dbpwdHash.equalsIgnoreCase(pwdHash)) {
            response = "ok";
         }
      } catch (JSONException e) {
         e.printStackTrace();
      }
   }
   object.put("username", username);
   object.put("response", response);
```

6. Register the web service in the main method of the web project.

```
Endpoint.publish("http://localhost:8080/WS/Login", new LoginImpl());
```

7. Deploy the web service. The generated archive (war) is copied into the *deployments* subdirectory of the JBoss server. The application server automatically detects the war files and deploys or redeploys them.

2.3 Android client application

The Android client application creates an activity with two input fields (username and password). After clicking the login button a thread is created. In this thread the web service is accesed. To access the web service the library ksoap2 is used. The JSON response of the web service is retreived. A message is shown on the screen indicating whether the login was successful or not.

Steps made by the client application:

1. Create an activity and add butons and textviews to its layout

```
...
 <EditText
 android:text=""
 android:id="@+id/txtUserName"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:inputType="text">
 </EditText>
...
<Button
 android:text="@string/txt_login"
 android:id="@+id/btnLogin"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content">
</Button>
...
```

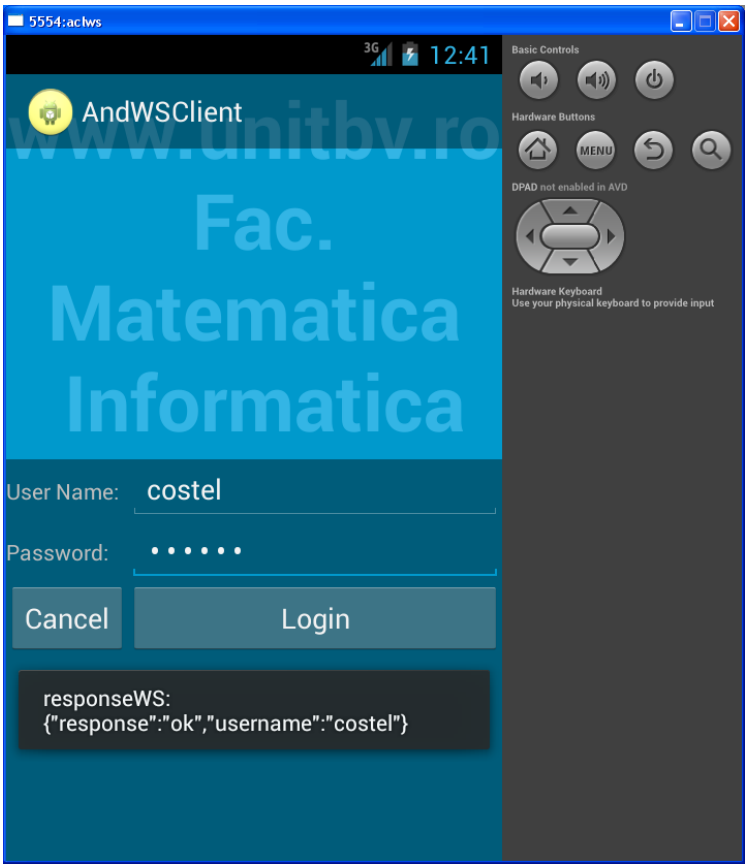

Figure 2: Android client application response

2. Initialize the SOAP parameters for accessing the login web service. These parameters can be read by accessing the WSDL[8] *URL*.

```
private static final String SOAP_ACTION = "";
private static final String METHOD_NAME = "doLogin";
private static final String NAMESPACE = "http://but.acl/";
private static final String URL =
    "http://192.168.22.101:8080/DynWP-WS/LoginImpl?wsdl";
```

3. Download and add the ksoap2 library to the Android project [8]. It must be also marked in *Order an Export* configuration tab of the *JavaBuildPath* for the Android project.

4. Implement the login activity and the onclick listener method

```
public void onClick(View v) {
   switch (v.getId()) {
   case R.id.btnLogin:
```

[8]Web Services Description Language

```
        System.out.println("login button clicked");
        String username = txtUserName.getText().toString();
        String passwd = txtPassword.getText().toString();
        passwd = Hashing.md5(passwd);
        callDoLogin(username, passwd);
        break;
\\ ...
```

5. Call of the login web service using a thread

```
SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);

PropertyInfo propertyInfo = new PropertyInfo();
propertyInfo.type = PropertyInfo.STRING_CLASS;
propertyInfo.name = "arg0";
request.addProperty(propertyInfo, username);

propertyInfo = new PropertyInfo();
propertyInfo.type = PropertyInfo.STRING_CLASS;
propertyInfo.name = "arg1";
request.addProperty(propertyInfo, passwd);

SoapSerializationEnvelope envelope = new
    SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.setOutputSoapObject(request);
HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);
androidHttpTransport.call(SOAP_ACTION, envelope);
SoapPrimitive response = (SoapPrimitive) envelope.getResponse();
responseWS = response.toString();
```

6. Decode and show the response

```
JSONObject jsonObj = new JSONObject(responseWS);
String res = "responseWS: " + jsonObj.toString();
System.out.println(res);
Toast.makeText(LoginActivity.this, res, Toast.LENGTH_LONG).show();
```

2.4 Remarks

- When the error message "Webservice invocation failing with Unmarshalling Error" appears, the parameters and their order must be verified.
- Timeout for thread - when the http connection (trasnport for SOAP messages) isn't available the background thread must have a timeout.
- a simple Java client can be implemented by generating and using the stubs for the web service by using the command $wsimport - s.http : //localhost : 8080/DynWP-WS/LoginImpl?wsdl$ [6]. The classes $LoginImplService$ and $Login$ are automatically generated.

```
LoginImplService serviceLogin = new LoginImplService();
Login portLogin = serviceLogin.getLoginImplPort();
System.out.println(portLogin.doLogin("costel", Hashing.md5("costel")));
```

- The method $md5(String)$ from the class $Hashing$ is used by the Android client application to compute the hass for the given password before sending it through the network.

```
digest = MessageDigest.getInstance("MD5");
digest.update(inputPassword.getBytes(),0,inputPassword.length());
String hash = new BigInteger(1, digest.digest()).toString(16);
return hash;
```

- *persistence.xml* must be placed into the web project into the subdirectory $src\backslash main\backslash resources\backslash META-INF$
- For creating the war files using maven command *mvn clean install* is used. For creating executable jars command *mvn package* is used.

3 Conclusion

The Java enterprise applications are scalable. Many of them are ported to mobile devices. One of the porting mechanisms is by using the web services architecture presented in the paper.

The login web service architecture can be extended and adapted to any other application functionality. The proposed web service uses the CDI mechanism and also uses JSON format for representing requests and responses such that any fields from the database can be transferred between client application and the database using the web service. The web service has also access to all local resources on the server where it runs. It is run in JBboss application server in the context of the enterprise applications which are to be extended.

References

[1] Aldea, C., L., *Elemente de securitate în reţele de calculatoare*, *Transilvania* University Publishing House, Braşov, 2010.

[2] Aldea, C., Sangeorzan, L., Aldea, A. (2009, September). *Web services and enterprise games*, In I. Rudas, N. Mastorakis (Eds.), WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering (No. 5). WSEAS.

[3] https://developer.android.com/sdk/index.html

[4] http://developer.android.com/sdk/installing/installing-adt.html

[5] http://www.oracle.com/technetwork/java/javase/downloads/index.html

[6] http://java.dzone.com/articles/jax-ws-hello-world

[7] http://www.jboss.org/jbossas/downloads/

[8] http://sourceforge.net/projects/ksoap2/

[9] http://maven.apache.org/download.cgi

[10] https://zorq.net/b/2011/07/12/adding-a-mysql-datasource-to-jboss-as-7/

[11] http://dev.mysql.com/downloads/connector/j/