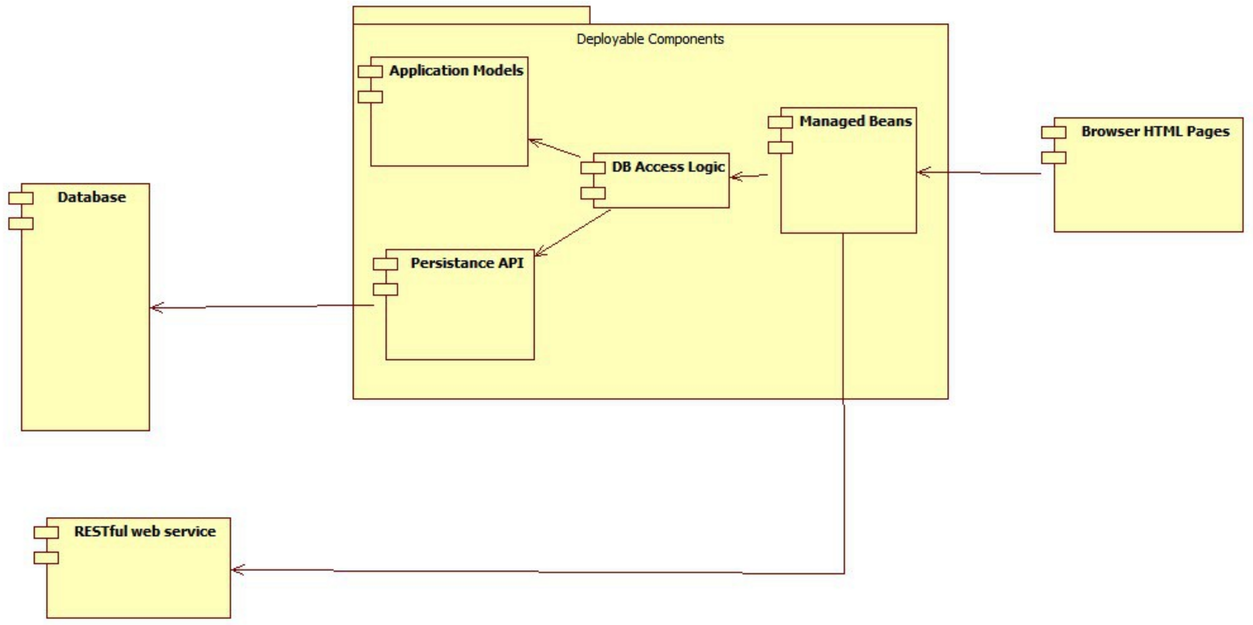
Distributed Systems – assign 1

-Stefan Prisca, group 30441(?)-

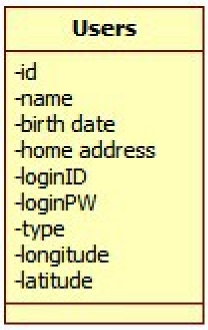
Conceptual Diagram



In the above diagram, you can clearly see all the important parts of this application and how they interect with each other:

- The Database: represents the storage used for the application.
- The Persistence API: represents the API used to connect with the database and to ease data persistence.
  For this application, Eclipse Link was chosen as the persistence API. It offers a wide range of possible database connections (including SQL Dbs and NonSql dbs). It is also easy to configure through xml files, and has a central maven repository that facilitates the maven build.
- Application Models: These are in-memory representations of the data models from the database.
- Java Beans: are java objects respecting the Java Bean conventions. This are used to communicate with the client side, using the Java Server Faces technology.
- RESTful Web Service: this is the web service used for identifying the current location and timezone of the user.
  For this application, Google Maps Timezone API was used. It provides an easy way to determine the location through longitude and latitude.
- RESTful client: this represents the client used to communicate with the web service.
  In order to build this client, the apache cfx jax-rs framework was used.
- Client-side View: contains the web pages that will be displayed in the browser. These were build using the JSF2 technologies.

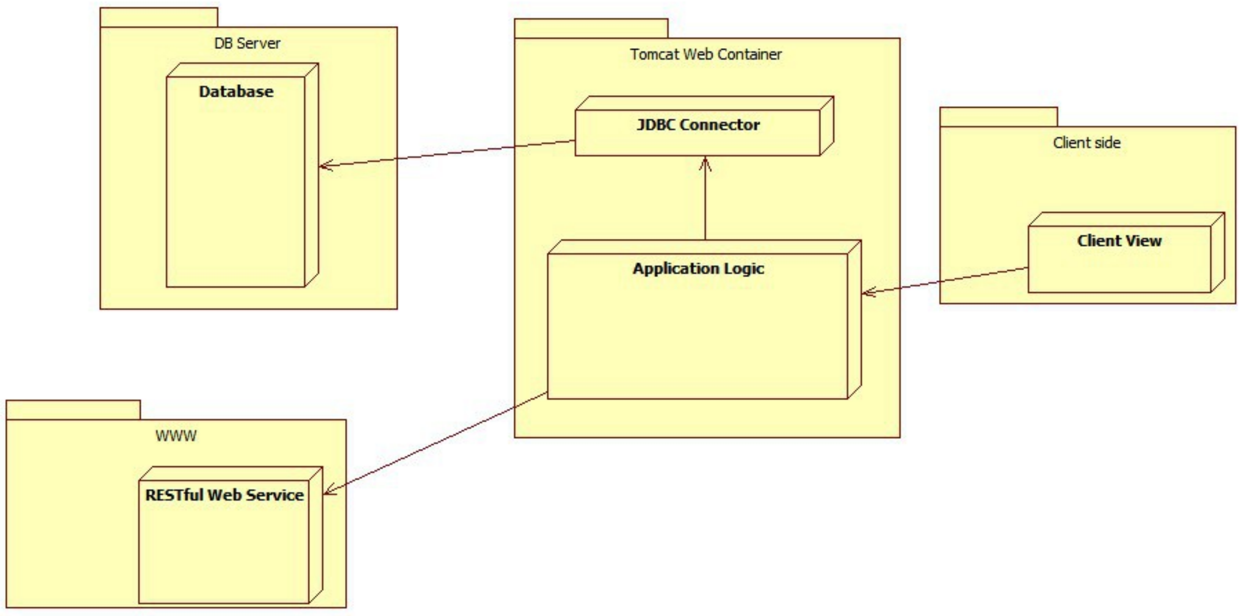
Data Design



The data design is quite simple for this application.
It consists of only one table that contains all the user information that is required for the application to run:

- id: database identifier for the current user
- name: name of the user
- birth date: the birth date of the user
- home address: the home address of the user.
- LoginID: the id the user is using to log in the application
- LoginPW: the password used to log in to the application
- type: the type of the user: administrator or regular user.
- logitude and latitude: world coordinates for the user to determine timezone.

Deployment diagram



The above diagram clearly separates the server-side, client side and web service components of this application.
You can see what runs of each of the three:

1. Server side:
   1. Database will be stored on the server, the application having direct access to it through the JDBC connector and the persistence API.
   2. The Java Data Base Connector (JDBC) driver will also reside on the server side. This allows the persistence api to connect to the data base.
   3. Server-side application: contains the EclipseLink persistence API, the applicaiton models, the Java Beans and the RESTful client.
      This is basically the application parts that run on the server.
2. On the WWW: the RESTful web service will reside on the WWW. This is the google maps timezone api.
   The server-side RESTful client will access this service through a *get* request to the server, and will receive the responce in a XML format.
3. Client side: this contains the JSF2 web pages that will be diplayed on the user browser.

Installing the application

It is recommended to use maven tool for building and deploying the project.
All the pom files are configured, so in order to run the build, do the following:

1. Clone the project git repository
2. Go to the App1 folder.
   There should be two projects here:
   1. ro.stefanprisca.distsystems.app1
   2. ro.stefanprisca.distsystems.app1.tests
      This project contains the tests for the application. It is also included in the build.
3. Before installing the application, you'll have to install the mssql jdbc driver:
   1. go to the mssqljdbc folder
   2. you should have the *sqljdbc4.jar* driver there.
   3. For windows systems, there is a .bat file that will install this driver using maven.
   4. For other systems, use the command:
      mvn install:install-file -Dfile=sqljdbc4.jar -DgroupId=com.microsoft.sqlserver -DartifactId=sqljdbc4 -Dversion=4.0 -Dpackaging=jar
4. After maven has installed the jdbc driver in your local repository, go back to the application folder (App1) and run *mvn clean install*
5. This will install the application, and generate the *ro.stefanprisca.distsystems.app1.war* file in ..\App1\ro.stefanprisca.distsystems.app1\target

Deploying the application using Tomcat.

In order to deploy the application, you'll have to install Tomcat as a local service first.

After you did this, copy the above mentioned *ro.stefanprisca.distsystems.app1.war* file in the webapp folder of Tomcat.
Now you just have to start Tomcat and access the page:

http://localhost:xxxx/ro.stefanprisca.distsystems.app1/home.xhtml

Further instructions on how to deploy applications with apache tomcat can be found here.

Opening the application in Eclipse:

If you have the Eclipse Maven plug-in installed, all you need to do is import the project as a Maven plug-in.
Note that this is the recommended way to import the project in Eclipse.

If not, you'll have to select *Create New Project,* deselect the *Use default location* option, and set the project location to the *ro.stefanprisca.distsystems.app1* folder.

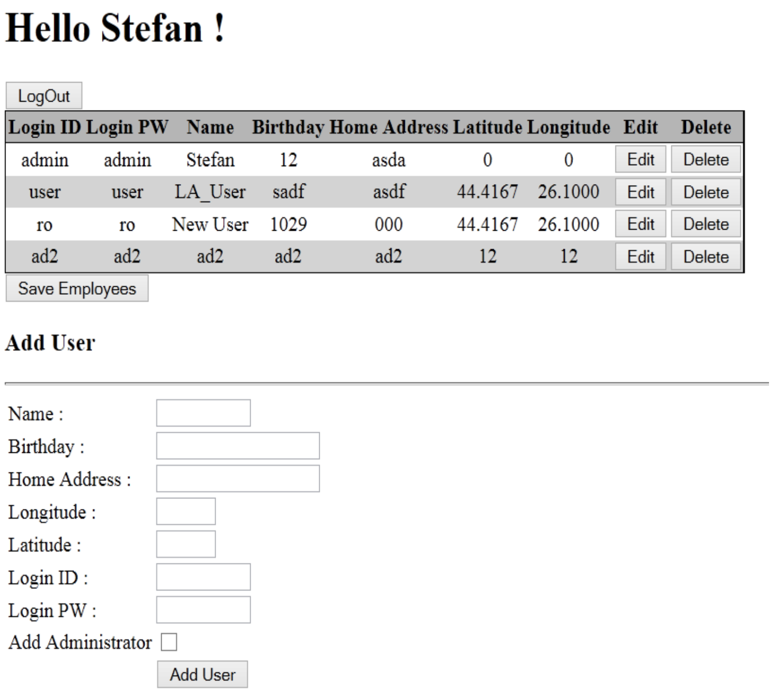
Running the application.

After you deploy the application using tomcat, and you access the application through the browser, you'll have to log in.
There are two testing accounts available:

1. administrator account: id = admin; pw = admin
   Use this account to test the administrator functions
2. regular user account: id = user; pw = user
   Use this acount to test the regular user functions.

Depending on your log in credentials, you will be redirected to the administrator page or to the regular user page.
Note: If you are not logged in as an administrator, and you try to access the administrator page, you will be redirected to the log in page. Only logged in administrators can access this page!

1. Administrator page:
   Here you will see a table with all the user records from the database.
   You can edit users information, delete users or add aditional users to the data base.
   Note that after you edit user entries, you'll have to press the *Save Users* button in order to persist the changes in the data base.
   If you edit the longitude and latitude fields, it is highly recommended to use real world coordinates for them. This will allow you to further test the Timezone recognition functionality.



Hello Stefan !

LogOut

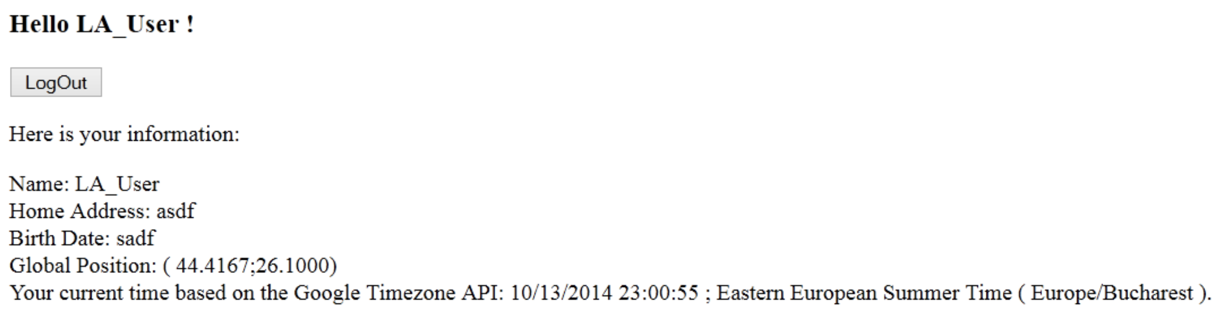
Login ID	Login PW	Name	Birthday	Home Address	Latitude	Longitude	Edit	Delete
admin	admin	Stefan	12	asda	0	0	Edit	Delete
user	user	LA_User	sadf	asdf	44.4167	26.1000	Edit	Delete
ro	ro	New User	1029	000	44.4167	26.1000	Edit	Delete
ad2	ad2	ad2	ad2	ad2	12	12	Edit	Delete

Save Employees

Add User

Name :	
Birthday :	
Home Address :	
Longitude :	
Latitude :	
Login ID :	
Login PW :	
Add Administrator	☐
	Add User

2. Regular user page:
   Here you will see basic information about the regular user.
   The page will display the user record from the database, and the user's current global position and timezone, based on the longitude and latitude fields.


Hello LA_User !

LogOut

Here is your information:

Name: LA_User
Home Address: asdf
Birth Date: sadf
Global Position: (44.4167;26.1000)
Your current time based on the Google Timezone API: 10/13/2014 23:00:55 ; Eastern European Summer Time (Europe/Bucharest).

References

- JSF2: <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>
- EclipseLink: <https://www.eclipse.org/eclipselink/>
- Tomcat: <http://tomcat.apache.org/>
- Maven: <http://maven.apache.org/>
- Project github repo: <https://github.com/stefanprisca/FacultDistributedSystems>