Homework 4, Web-Based Applications And Application Servers

Network Programming, ID1212

Vasileios Charalampidis, vascha@kth.se

December 11, 2017

1 Introduction

The purpose of this fourth assignment is to develop a three-tier web-based application using frameworks for all layers, while deploying and running this application on an application server.

To meet these goals, a Java application was deployed to handle the case of a currency converter. In this application, the user opens the browser in a specific url i.e. http://localhost:8080/CurrencyConverter/ . There he is requested first to type the value of 1 SEK in EURO, USD, and GBP. After these values are inserted and the user presses the 'Send' button, all these currencies are stored in a database for future use. After that, he is ready to type the amount to convert and choose from which and to what currency to convert. As soon as the 'Convert' button is pressed, the final conversion is displayed to the user. A more clear view on the UI and the functionalities of the web application will be given later on the results section.

The deployed program meets some specific requirements, so as to present an acceptable solution for this assignment. The first requirement is that the converter must be able to convert between at least 4 different currencies. The second requirement is that the client is a web browser. The third requirement is that frameworks for all layers in the server must be used. For example JSF for the view, EJB for controller and JPA for model and integration. The fourth requirement is that the server must handle transactions. To achieve that it is proposed to use EJB to implement container-managed transactions. The fifth requirement is that the conversion rates must be stored in a database. The last requirement refers to an informative user interface. This means that the current state of the program must be clear to the user, and the user must understand what to do next.

2 Literature Study

In order to accomplish this assignment, it was very important at the beginning to read carefully the application scenario and all its requirements. After the realization of the given context, the given material i.e. code and videos, at canvas for this course, was more than helpful for the java implementation. The videos were very informative, as the instructor explained in good detail the concepts that were necessary for the implementation i.e. JSF, EJB, JPA. Except from the provided course material, information about JSF and EJB found on the web was also part of dealing with the application.

What became clear from the materials used, was that a 3-tier model consists of a client (1st tier-GUI), a business logic, and some system services (and databases).

For the frameworks used in this assignment, JSF is a framework intended to simplify development integration of web-based user interfaces. Some of its benefits (ref. here) are that, a) it provides reusable UI components, b) it makes easy data transfer between UI components, c) it manages UI state across multiple server requests, d) it enables implementation of custom components, e) it wires client-side event to server-side application code.

As for EJB (ref. here), this is a development architecture for building highly scalable and robust enterprise level applications to be deployed on J2EE compliant Application Server such as JBOSS, Web Logic, payara etc. EJB is mainly divided in 3 categories/types: a) Session Bean, which stores data of a particular user for a single session, b) Entity Bean, which refers to persistent data storage, and c) Message Driven Bean, which is used in context of JMS (Java Messaging Service).

3 Method

The Java implementation was created with "NetBeans" by Sun Microsystems. NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules (ref. here).

The application server used in the assignment was Payara. Payara Server is an open source application server derived from GlassFish. It is a drop-in replacement for GlassFish Server Open Source Edition, with the peace of mind of quarterly releases containing enhancements, bug fixes and patches (ref. here).

The database used in the assignment was Apache Derby, which is an open source relational database implemented entirely in Java.

Before proceeding with the actual currency converter, some tests were performed to smaller units e.g. a simple web app using JSF to handle an HTML view, etc. so that to be sure that specific concepts will work in the final implementation.

To evaluate that the final solution met the requested requirements, several manual acceptance tests were performed to the code e.g. store currency rates, convert a specific amount, error messages etc.

4 Result

A layered architecture was used for this assignment (Figure 1). There is a folder 'webapp' for the client side (GUI), where an 'index.xhtml' (here) occurs for the communication with the client; and four packages for the server side:

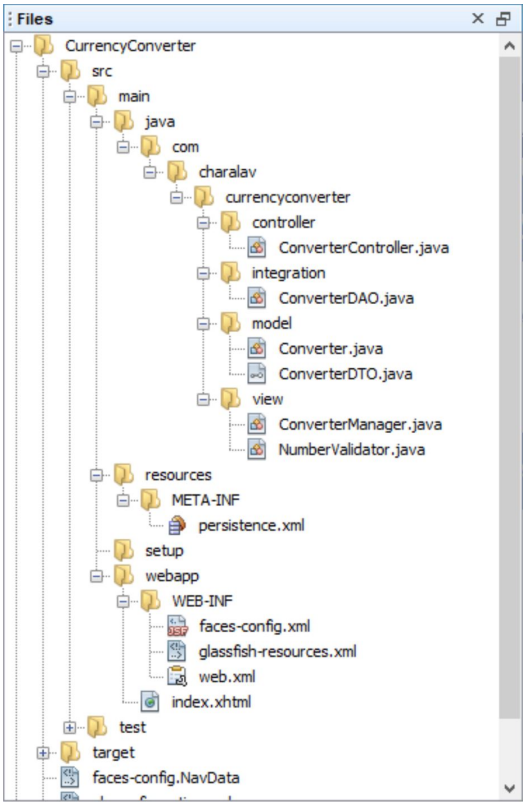

Figure 1: The architecture of the currency converter program

- The view package contains two java classes: a) the ConverterManager, which works as a gateway between the client and the server. Several set and get methods occur here for handling the input and output of the client, e.g. a setRateEuro method (line 101) for handling the value of euro rate. There are also methods for storing the input currency rates (line 75) by calling methods of the controller package, and methods for starting the calculation of the currency conversion (line 88), again by calling controller package methods, b) the NumberValidator class (here), which is

used as a validator for the input values of the client. This means that if the client tries in his browser to send negative values to the server, then a warning message will be displayed (Figure 2) and nothing will be stored to the database.

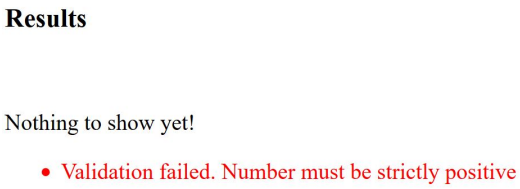

Figure 2: The message when the client tries to input negative values

- The controller package contains just one class: the ConverterController, which serves as a gateway between the view and the model package. All calls to the model pass through this class. It includes methods for storing currency rates (line 32), finding a converter with specific currency rates from the database (line 41), and starting a new currency conversion (line 55).
- The model package contains two classes: a) the Converter, which creates converter objects with specific id, euro rate, usd rate and gbp rate. It also calculates a new amount conversion (line 56) by providing to it the amount, the starting currency and the desired currency. The algorithm for the conversion calculation is mainly one line of code (line 87), b) the ConverterDTO, which is simply an interface for returning the converted amount (line 21), the euro rate (line 27), the usd rate (line 33) and the gbp rate (line 39). All of these methods are override by the Converter class.
- The integration package contains just one class: the ConverterDAO, which is used for storing currency rates to the database (line 33) and querying through the database to find the last saved Converter class (line 42), with all the currency rates.

Considering now each requirement separately:

- There should be 4 different currencies for the application. And that is the case here, as the user can select to convert between sek, euro, usd and gbp (line 43 and line 52). A drop down list is displayed to the user's browser for that reason (Figure 3).
- The client must be a web browser. As it was mentioned briefly previously, there is a specific .xhtml file for this reason (here). This file is running on the following url: http://localhost:8080/CurrencyConverter/faces/index.xhtml . There is an .xml file for configuring this website (here). In the browser, the client can input currency

Figure 3: The drop down list with the 4 currencies

rates, as well as type the desired amount to convert and select the corresponding currencies. After some process by the server, it is displayed in the browser the converted amount.

- It is needed to use frameworks for all layers in the server. In the view layer, it is clear that the ConverterManager class handles a managed bean with the name 'converterManager' (line 21). This class contains everything related to the JSF index.xhtml page (set, get methods according to the user's input). Continuing with the controller layer, there the implemented class is a stateless EJB (line 22). It is used to perform independent operations. For the model layer now, the Converter class uses a JPA entity (line 19), with all the other appropriate tags for the entity. Finally, the integration layer contains a class which handles all interaction with the entity manager, and performs actions in the database. The class is also a stateless session bean (line 23) with a mandatory transaction (line 22).

- The server must handle transactions. EJB is used to implement container-managed transactions. More specifically, in the controller class, a transaction with type 'REQUIRES-NEW' is used (line 21), which indicates that a new transaction is to be started for the business method. Also, as mentioned previously, the class in the integration layer uses a 'MANDATORY' transaction type, which indicates that business method will execute as part of transaction, otherwise exception will be thrown.

- The conversion rates must be stored in a database. That is clear in the integration layer, where the class persists a new Converter object (line 33), with all the information on currency rates (Figure 4).

- The UI must be informative. This can be seen in Figure 5. The user is required to type the currency rates and then type the amount and select the desired currencies. When everything is filled and sent to the server, a respond is displayed with the results (Figure 6).

The final code of the Java application can be found here. Several comments are included there discussing about essential parts of the program.

#	CONVERTERID	RATEEURO	RATEGBP	RATEUSD
1	279	0.1	0.088	0.12
2	283	1.0	1.0	1.0
3	907	0.1	0.088	0.12
4	597	2.0	2.0	2.0
5	477	0.1	0.3	0.2
6	129	1.0	3.0	2.0
7	855	0.1	0.088	0.12
8	280	1.0	3.0	2.0
9	32	3.0	7.0	5.0

Figure 4: The database with info on currency rates

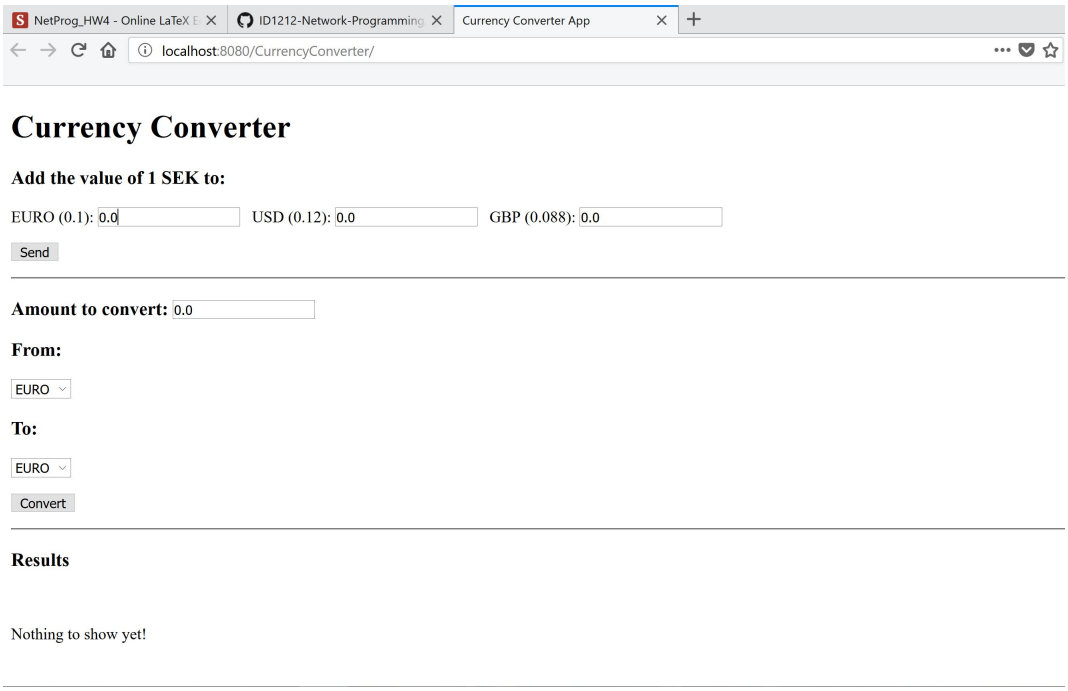

Figure 5: The UI in client's browser

5 Discussion

The main goals of the assignment were to learn about different frameworks for web-based applications and how to run and deploy such applications on an application server.

The requirements of the presented code can be summarized in converting between at least 4 different currencies, having a web browser interface for the client, using frameworks for all server layers, handling transactions from the server, storing the currency rates in a database, and finally having an informative UI. Any problems that occured during the execution were immediately identified as many 'throw exception' rules were implemented in the Java code.

If something was to be done differently, that would be again as in the previous three assignments, the testing part. It would have been beneficial to perform unit tests (e.g.

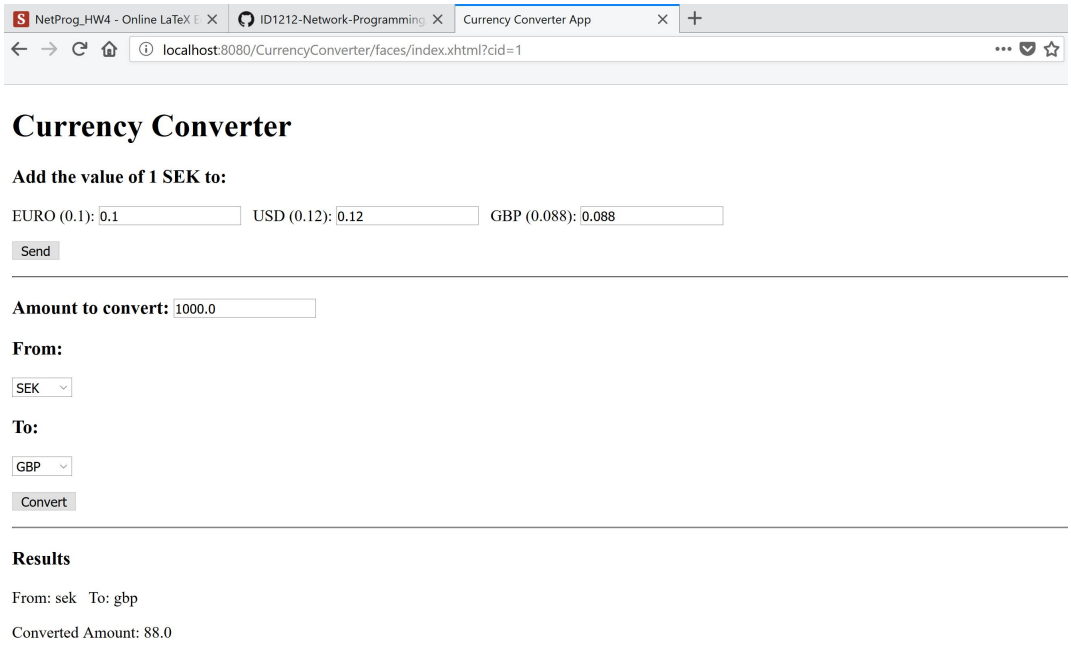

Figure 6: The results for the currency conversion

JUnit), so that to guaranty that everything performs fine without any issues.

6 Comments About the Assignment

This specific assignment was a really interesting topic and easy to implement. It took me about one day and a half to finish it (with several breaks in between).