Exercise Series 1

Banking example: Presentation and application logic

27 April 2012

The objective of this exercise is to develop the presentation and application specific business logic of the bank example using Servlets and JSP. *All questions below have to be answered explicitly in your report*, with your own words. Pieces of text literally copied from the lectures notes or books do not qualify for marking. The report with answers to the questions below should clearly describe the solutions produced, making it unnecessary for the teachers to dive into the code. Clarity and readability will be important for the marking of the reports.

The results of this exercise have to be submitted to Blackboard as a *zip file* containing the report in PDF format, all the files of the code you have produced (Java classes, JSP files, description files, Ant scripts, etc.) and the WAR files of the web applications. The WAR files should be generated using Ant scripts.

1 Banking example: architecture

Figure 1 below shows the architecture of the banking system. The banking system has been structured according to the three tiers *presentation*, *business logic* and *resources*. The business logic has been split in two layers (application and banking layers), and the resources tier includes the database access. In the business logic we separated the parts that are present in any banking application, from the parts that are specific for the applications that have to be built. The banking system will have two applications, namely *cash dispensing* and *home banking*.

In this exercise you are asked to build the presentation and application-specific business logic for the cash dispensing and the home banking applications. The application-independent parts of the business logic (transaction processing and authentication) will be simulated by local mock-up objects, so that the implementations can be tested.

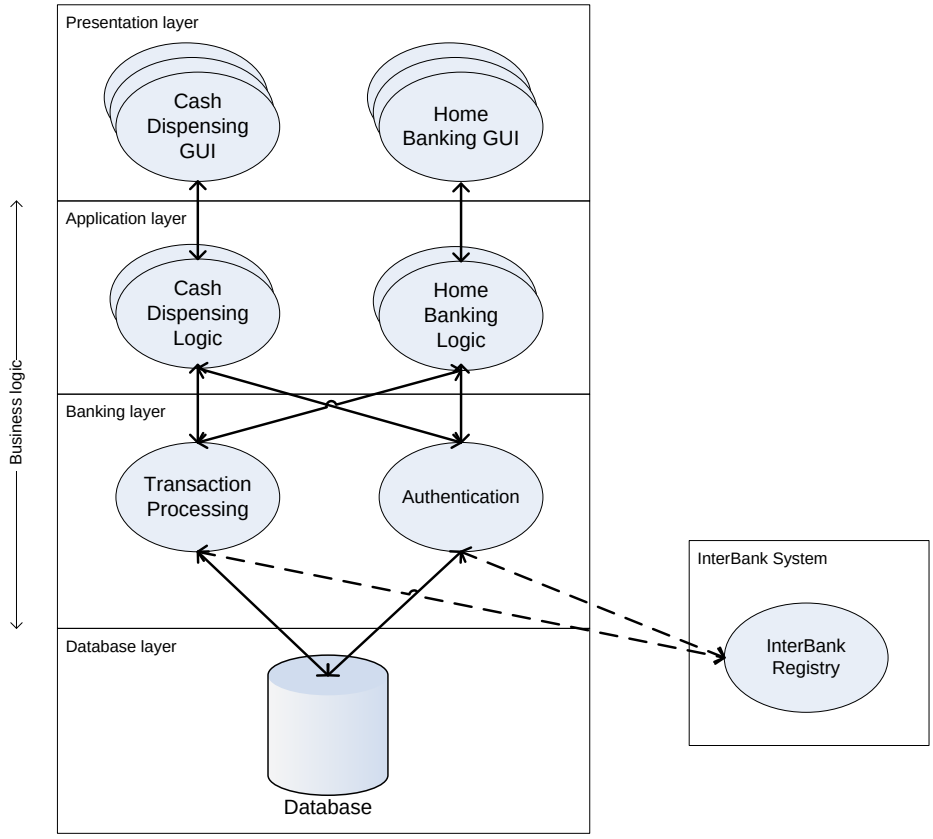

Figure 1. Banking example architecture.

2 Cash dispensing application

The cash dispensing application must support the following activities:

- *Authenticate the bank client*. Authentication is based on *bank account number*, some identifier carried by the bank card, called the *bank card id*, and a *PIN code*. In case authentication is successful, a *cash dispensing session* is established between the bank system and the bank client. During this session, the client may perform the other activities defined below;
- *Withdraw money*. This activity consists of a request to cash some amount of money from the bank account, which is either followed by the actual withdraw of the requested money or by a notification that the request is rejected for some reason (e.g., insufficient balance). The maximal amount of money that can be cashed during a session is € 500;
- *Deposit money*. This activity consists of informing the cash dispensing application the amount of money the client wants to deposit in his bank account. The maximal amount of money that can be deposited during a session is € 1.000;
- *Check account balance*. This activity consists of a request to check the account balance, after which the balance is displayed to the client;
- *Stop the cash dispensing session*. A session is terminated either explicitly by the bank client, or by the bank system after some time period expires in which the bank client has been inactive. The maximal inactivity period is 60 seconds.

3 Home banking application

The home banking application must support the following activities:

- *Authenticate the bank client*. Authentication is based on *username* and *password*. In case authentication is successful, a *home banking session* is established between the bank system and the bank client. During this session, the client may perform the other activities defined below;
- *Transfer money*. This activity consists of transferring an amount of money from the client's account to a target account. The transfer transaction should only be carried out if the client has enough balance in his account and the target account is valid. The information about the target account consists of the account number and client's name. The maximal amount of money that can be cashed during a session is € 5.000,00. For this exercise we assume that money transfer is only possible among clients of the same bank;
- *Check account balance*. This activity consists of a request to check the account balance, after which the balance is displayed to the client;
- *Stop the home banking session*. A session is terminated either explicitly by the bank client, or by the bank system after some time period expires in which the bank client has been inactive. The maximal inactivity period is 60 seconds.

4 Java Beans

The presentation and application layers should be implemented according to the MVC pattern, i.e., the functionality of the application should be distributed accordingly among the Servlets, HTML and JSP pages, and JavaBeans. This means that you have to design a data model for your JavaBeans design. Since these JavaBeans are internal to your implementation, we allow you some freedom to design this data model, but make sure it gives access only to information necessary to the applications, hiding information that is only relevant to the back-end system. The classes of this data model should represent the core concepts of the banking applications, namely, client, account and bank card. Both the cash dispensing and home banking applications are supposed to use your JavaBeans.

5 Provided interfaces

Local mock-up classes have to be implemented to simulate the access to the bank's back-end system (the Transaction Processing and Authentication services). JavaBeans for clients, bank cards, accounts and banks are generated and/or populated as a result of using these mock-up objects, which should implement the `Transaction` and `Authentication` interfaces shown in Figure 2.

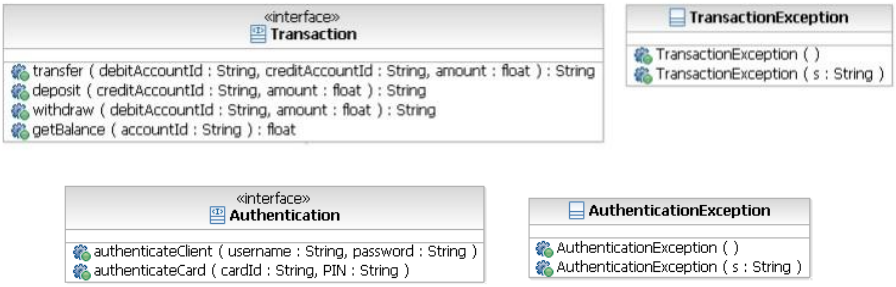

Figure 2. Mock-up interfaces.

Pages that can only be accessed after the login should implement a mechanism that checks whether the client has been successfully authenticated. This mechanism should prevent the access to these pages by directly entering the pages URLs in the browser, thus bypassing the login. This means that if someone directly enters the pages URLs instead of logging in, the browser should be redirected to the login page.

In case the user is successfully authenticated, the controller functionality redirects the client to main page of the application (home banking or cash dispensing) containing the corresponding options and a greeting to the client. From there the client can access the internal pages to proceed with the available activities.

In case the user is not successfully authenticated, the controller functionality returns to the client a page that displays a message with the login failure and asks for a new login attempt.

In both the cash dispensing and home banking applications, the user has the option to logout. When this option is selected the application should clean all the information pertaining to the current client's session and return to the login page.

Important notice

Master students are required to deploy the applications in the Google App Engine environment (see https://developers.google.com/appengine/downloads). You may decide to start with the local implementation and port that afterwards to the Google App Engine development tools, or start directly working with these tools. For Bachelor students the Google App Engine deployment is optional.

For all students it holds that the use of Ajax (asynchronous requests) with JavaScript yields a bonus point. This bonus point can only be earned if the report contains a section in which the students explain why their implementation complies with Ajax (see [1] and many other resources on the Internet).

Question 1

Draw UML class diagrams showing the classes you implemented, including the data model for the JavaBeans. Explain the relations among the classes. Explain the sequence of interaction between the client and the software components (e.g., JSPs, Java Beans, Servlets, etc.) and where these interactions take place (e.g., clients browser, server, Servlet container, etc.). Add diagrams to indicate how these software components can be assigned to concrete computers (end-user computers and servers). Explain which part of your implementation code corresponds to the model, the view and the controller. Draw UML diagram(s)

that show how your JSPs, Servlets and JavaBeans correspond to the elements of MVC pattern.

Question 2
Describe the strategy you applied in order to make the authenticated Client object available for the pages after the login. Make sure you answer the following questions in your report:

1. Which HTTP method did you use to exchange the login request between the login page and the Servlet and why?
2. How does the choice of HTTP method influence the security (possibility of eavesdropping) of the login process?
3. Explain how the pages in your system obtain the authenticated Client object to guarantee that only authenticated clients go through. In which part of the MVC pattern this authenticated Client object is created?

Question 3
Describe the approach you took in order to test your application. Make sure you answer the following questions in your report:

1. How did you test the Java Beans, the JSPs, the Servlet(s)?
2. How did you test the application as a whole?
3. How can you be sure the system works?

Describe your testing architecture (how test components interact with your implementation), your testing strategy (test cases), and the results produced in these tests.

Question 4
Explain why your implementation complies with Ajax [1]. Include in this explanation examples of interactions supported by your implementation.

References

[1] Paulson, L.D.; , "Building rich web applications with Ajax," Computer , vol.38, no.10, pp. 14- 17, Oct. 2005, doi: 10.1109/MC.2005.330