Software Architectures

Assignment 2: Architectural Patterns

Assistants: Kennedy Kambona, Janwillem Swalens
Mail: {kkambona, jswalens}@vub.ac.be
Office: {10F730, 10F719}

Deadline: 20th March 2014, 23:59

The goal of this exercise is to partially refactor an existing web application that uses a flawed three-tier architecture. You will have to make modifications in one tier of the application, but you will see that the flawed architecture requires you to make changes in other tiers as well. You should fix the architecture, and report on the design flaws of the original approach.

Assignment

For this assignment you have to provide an implementation and a report. The rest of this document provides more details about the design of the given application, the implementation assignment, and what we expect to be in the report. Furthermore, in the last section of this document you will find a short description on how to set up and prepare Eclipse, Tomcat, and the given application.

Deadline 20th March 2014 at 23:59. The deadline is fixed and cannot be extended.

Deliverables You should write a report (in English), and provide your modified implementation of the project. The report and implementation should be handed in as a single ZIP file.
In Eclipse, export your project as an archive file by right-clicking the project, and selecting Export > Export... Then, select: General > Archive File. Include your report, as a PDF file, in this archive as well, by placing it in your project folder.
Submit the ZIP file on the Software Architectures course page in PointCarré, by clicking on *Assignments (Opdrachten)* > *Assignment 2*. The file should follow the naming schema `(Firstname-Lastname_)*2.zip`, for example:
`Kennedy-Kambona_Janwillem-Swalens_2.zip`.

Team work You are allowed to work alone, or in a team of two. Only one of you should submit the report on PointCarré, but be sure to mention both names in the report!

Grading The exercises will be graded and can become subject of an additional defense.

Application Design

This exercise session will focus on a relatively small application, which implements a (flawed) three-tier architecture.

The application is a web portal, which allows adding and retrieving information regarding articles, books, conferences, etc. Users need to log in before they can add or retrieve this information, and they can only perform operations that are associated with their role (such as administrator, regular user, etc.). Figure 1 shows an overview of the application's architecture.

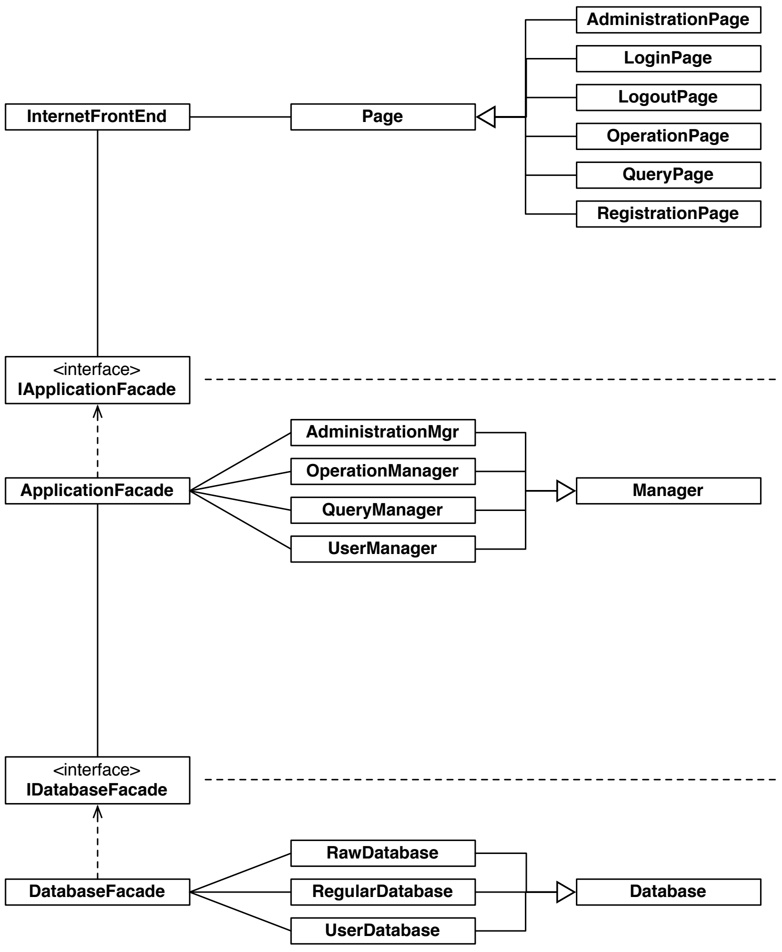

Figure 1: Application Architecture

Like most three-tier applications, there is a UI layer (top), an application layer (middle), and a database layer (bottom). The application's code follows this division: each of these layers is implemented in a separate package (ui, app, and db, respectively). The data package contains the application's data types.

Each layer is accessible to other layers through only one class, which acts as a façade (as in the design pattern). This facilitates replacing a certain layer's implementation with another, as other layers will continue to work as long as the façade's external interface remains unchanged.

The user interface layer is implemented by means of a Java Servlet (see the Internet-FrontEnd class). This Servlet will interpret the HTTP and POST requests sent to it and defer the requests to the appropriate Page class. Each page will handle this request using the application layer. Results are displayed by converting the application's data types to XML, and applying an XSL transformation on it.

The application layer is mainly responsible for session management; other requests are mostly forwarded to the data layer.

The database layer currently uses JDBC to connect to a HyperSQL database. The application's data types are created using SQL SELECT queries; new and modified data are saved using SQL INSERT or UPDATE queries.

Assignment

The goal of n-tier architectures is to decouple the different layers of an application, thus enabling the evolution of each layer independent from the other layers.

The goal of this assignment is to replace the database layer with a new implementation. More specifically, you have to replace the current database layer by a flat file database (comma or tab separated: CSV files). If you deem it necessary, you are allowed to use an existing library to read/write CSV files.

The resulting application should have as low coupling between the layers as possible, and adding yet another implementation for a layer should be as easy as possible.

Remark: Actually, this implementation is "flawed" in two different ways. First, it is not an actual n-tier application, to keep things simple the different layers do not use separate processes. Secondly, the separation between the layers is not sufficient. You should fix only the second problem.

Required Functionality Your new layer does not need to implement all of the original layer's functionality. Only the following use cases need to be supported:

- Creating a new "free subscription" user
- Creating a new "cheap subscription" user
- Creating a new "expensive subscription" user
- Log in with the new users
- All other operations do not need to be implemented, but can raise an exception.
- Enable a developer to change between database layers using a configuration setting. Layer-specific settings should be in the configuration file.

- The database layer interface should be generic, and not expose the specific requirements of the layer.
- Identify the remaining design deficiencies and include a discussion in the report.

The implementation of the original database layer needs to remain in the project but needs to be move to a new `db.sql` package. The package for the new database layer should be named `db.flatfile`.

There is at least one other known design deficiency with regard to the layering/decoupling. You do not need to fix the problems which are not explicitly asked for in the assignment. However, a strategy to fix those issues needs to be included in the report.

Suggestion In the source archive, you will find some rudimentary test cases, which are neither complete nor actually tested. A typical case of unmaintained tests. Thus, it is suggested to write additional JUnit tests to make sure your new layer is implemented as required.

However, *Software Architectures* is not a programming course, i. e., you do not have to implement test cases to fulfill this assignment. It is only suggested to do so, since our experiences shows that tests cases can help with designing a cleanly separated architecture.

Report

For the report, briefly answer the following questions:

- Which design flaws have you discovered in the implementation of the three-tier architecture?
  - For example, where did you need to change another layer in order to get your new implementation to work?
  - How did you improve the architecture to facilitate adding new implementations for a layer in the future?
  - Which changes would need to be done or have been done to achieve a clean separation between the layers? Clearly identify the problem, and describe how the code could be refactored.
- How does your configuration solution work, i.e., how can a user switch between layer implementations?

Preparing Eclipse and Tomcat

The assignment was designed using Eclipse and Apache Tomcat 6.0.39.

What follows is a step by step description on how to set up Eclipse and Tomcat for your platform.

Preparing Eclipse

Either use **Eclipse for Java EE Developers** (recommended):

1. Download "Eclipse for Java EE Developers" from `http://www.eclipse.org/downloads/`.
2. Extract the downloaded file to a path of your choice and start Eclipse. It is recommended to create an empty workspace for these assignments.

Or, use **Eclipse for RCP and RAP Developers** from the previous assignment:

1. In Eclipse, go to Help > Install New Software...
2. Select the Kepler software site in the menu after "Work with:".
3. From the list, select the following packages in the section "Web, XML, Java EE and OSGi Enterprise Development":
   - Java EE Developer Tools
   - JST Server Adapters
   - JST Server Adapters Extensions
   - JST Server UI
4. Press Next > and install these packages.
5. Restart Eclipse when asked.

Preparing Tomcat

Tomcat is the web server on which the web application will run. To install it:

1. Download Tomcat 6.0.39 **Core** zip (**binary distribution**) from `https://tomcat.apache.org/download-60.cgi`.
2. Extract the archive into a folder of your choice.
3. In Eclipse, set up the Tomcat integration by going to the preferences: Eclipse > Preferences or Window > Preferences.
4. Choose Server > Runtime Environments, and click Add...
5. Select Apache > Apache Tomcat v6.0 and then Next >
6. As Tomcat installation directory choose the folder you extracted the Tomcat archive to. The folder you need to select contains folders such as `bin`, `conf`, `lib`, and `webapps`.

Preparing the Web application

Finally, configure the web application:

1. Download the `web_portal.zip` from PointCarré
2. In Eclipse choose File > Import... Then choose General > Existing Projects into Workspace (*not* Archive File). On the next page, choose Archive file: `web_portal.zip`.
3. Now you will need to fix the Web application configuration to fit your local machine.
4. First you might need to fix the referenced JRE. Right click on the project and select Properties... Go to Java Build Path, open the tab Libraries. If there is an error indicated, double click on JRE System Library and choose for instance the Workspace default JRE.
5. Fix the path in `/web_portal/WebContent/WEB-INF/web.xml` it needs to point to the `web_portal.cfg` in your Eclipse workspace. On Windows systems, a backslash needs to be escaped by another one. On Unix systems, you have to use forward slashes.
6. Fix the paths in `/web_portal/web_portal.cfg` as well.
7. Now your setup should be ready to allow you to debug the application with Tomcat by right clicking on the project and selecting Debug As > Debug on Server.