Software Architectures: Assignment 2

Arno Moonens
arno.moonens@vub.ac.be
Rolnr. 500513

Jens Nevens
jens.nevens@vub.ac.be
Rolnr. 500093

Design flaws

First of all, the data layer was changed. In this layer, both conversions from data-structures to SQL queries or XML text and vice-versa took place. This does not belong in this layer. The sole concern of the data-layer should be to represent the data internally. Converting data to SQL, or any other format such as JSON, should be done in the concrete implementation of the database layer. Transforming data to XML is only needed for the UI, so this is a concern of the UI layer and should consequently be implemented there.

In order to facilitate adding new implementations for the database layer, we included code to correctly initialize a type of database implementation based on configuration parameter values. This is dealt with by the Application Facade. We also added an abstract interface for the Database Facade. A new database layer should only implement this interface in order to be operable with the current application. Indeed, the new database layer is not obliged to use the same structure as the current database layer, i.e. there is no need for a Raw, Regular and User Database. This makes it very easy to add new database systems. Exactly the same changes can be done to allow multiple application layers.

As mentioned in the first paragraph, converting data from and to SQL or XML is not the responsibility of the data layer. So, in order to achieve a clean separation between the layers, these conversions should be moved to the appropriate layers (i.e. the database layer for SQL conversions and the UI layer for XML conversions). We have moved some of these conversions, but not all of them. Especially the SQL conversions were moved, in order to support the new JSON database.

Database layer diagrams

UML diagrams of the database layer and of the entire application can be seen in Figures 1 and 2, respectively.

Switching between implementations

A user can switch between two database implementation by using one additional configuration parameter for the Servlet. This parameter is defined in `/WebContent/WEB-INF/web.xml`, namely `db_type`. This parameter defines the type of database to be used.

In our application, the values of the `db_type` variable can be set to `sql`, when using the SQL database, and to `json`, for using the JSON database. This is easily extensible, e.g. one could give this parameter the value `no_sql` when using a NoSQL database like MongoDB. In order to further extend the application, one needs to provide a URL to the database of the specified type. This could be done by each time altering the `db_url` variable, or by specifying a specific variable for each database type. We have opted for the first option, since we believe this is slightly more user-friendly. For example, consider an application that supports up to 15 kinds of databases. Using the first method, this would mean 15 different parameters specifying the URL for each of them. In our case, we merely have to update the `db_url` parameter accordingly.

The above mentioned parameters are read by the `InternetFrondEnd` class. This class will read, among other parameter values, the type of the database and determine the database URL. The type and the URL are passed on to the `ApplicationFacade` class, where the correct instance of the `DatabaseFacade` is instantiated.

Facade pattern

The Facade pattern facilitates switching between different implementations of a certain layer, since it provides a unified interface for the different subsystems. Each database layer should implement what the Database Facade exposes to the outside, which is determined by the interface, making the implementations of these different database systems completely independent. This also makes the communication between the application and the database layer completely independent of the explicit implementations or underlying database system since the application layer only communicates with the Database Facade. In short, the Database Facade is the single point that takes care of the decision of which database system to use. This leads to low coupling between the layers, low coupling between the different database systems and high cohesion within the different database implementations.

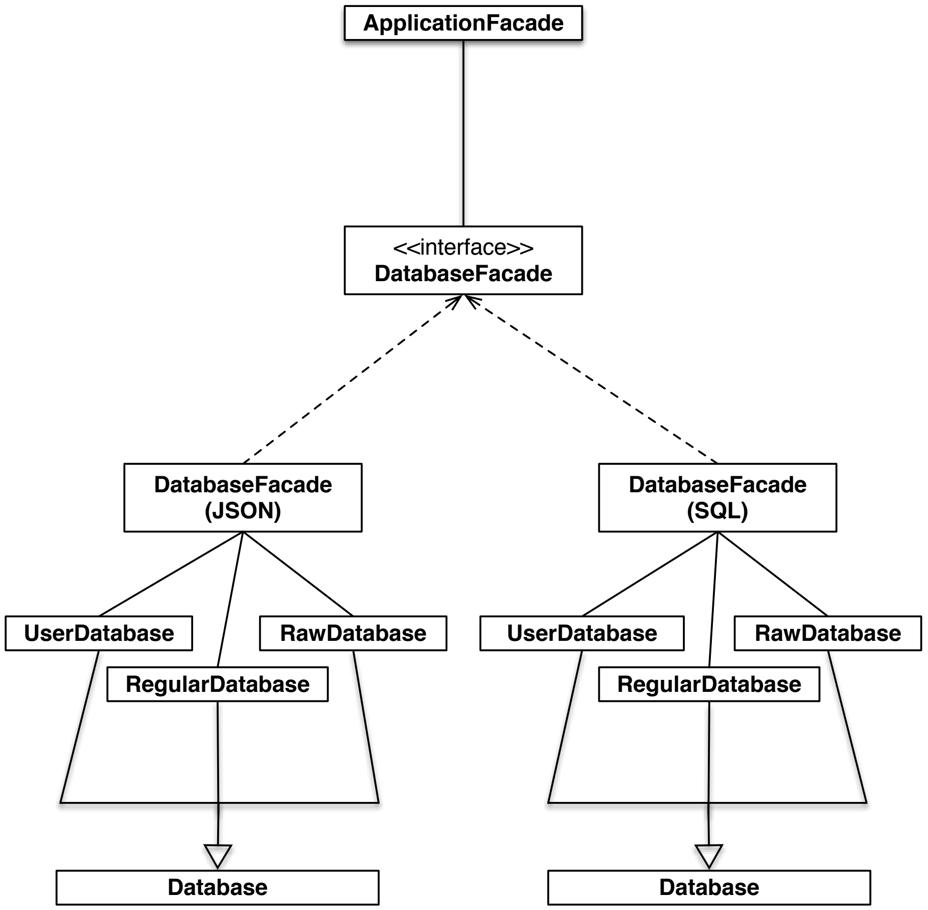

Figure 1: UML diagram of the Database layer

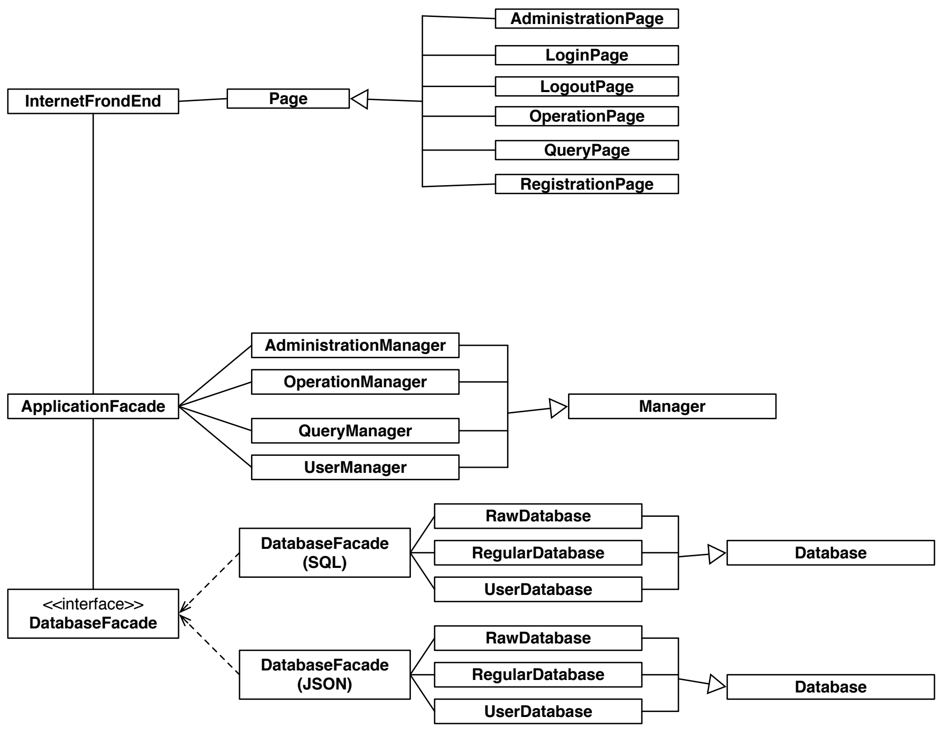

Figure 2: UML diagram of the entire application