E-math server-side system

1 FUNCTIONAL DESCRIPTION AND ANALYSIS

The server-side system responds to requests that are sent by e-books. The main functionalities that the system has to implement are authenticating users, saving solutions, saving additional book content and returning solutions as well as book content from the data storage of the server on request. In addition to the main functionality the system also implements an error log for unsuccessful login attempts and for logging attempts to store malicious content. All requests to the server are also logged into a separate traffic log. New students are also generated and saved by the system. A user is always a teacher, student or administrator.

The books that are connecting to the system always need to send the valid username and password combination of a user as well as a valid course id. A course id is valid if the user is either a teacher or student in the course. An admin does not have to send a course id. The client sends the request over http/ajax and the response is a string that contains an integer or a json object depending on the type of request. The type of request is recognized with help of the parameter 'type' that is sent with the client request. This server-side system and the client book together form a dumb server-smart client solution where the client takes care of the actual logic and visualization of the application. The co-operation between the e-book and the server-side system in a nutshell is visualized in picture 1.

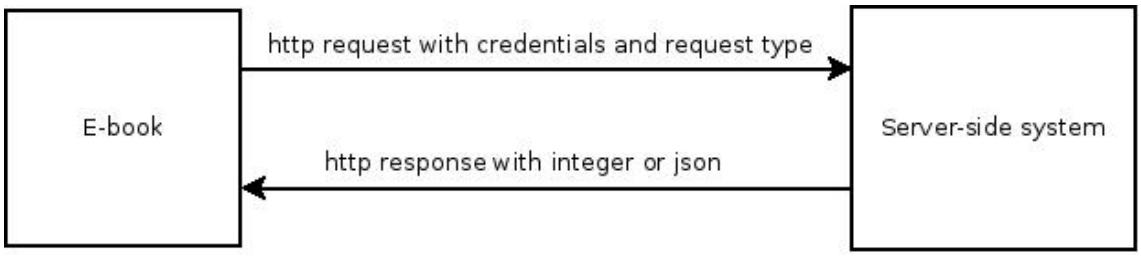

Figure 1: Client-side/Server-side co-operation in a nutshell.

2 DESCRIPTION OF THE SERVER-SIDE SYSTEM

The system is divided into layers to clarify its structure. Those layers are the web layer, service layer, domain object model (or model in short) and the persistence layer. The client handles the view layer. The connections between the different layers can be seen in figure 2.

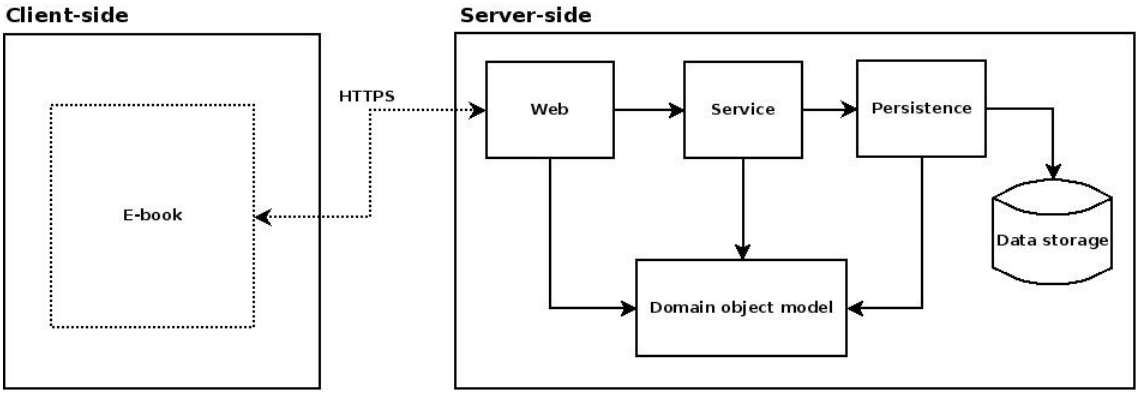

Figure 2: Server-side system layers.

2.1 Web layer

The Model and service layers do not know about the navigation logic of the system. That is the responsibility of the web layer. The idea of the web layer is to handle all web specific aspects of the system and in this way isolate the service layer from all web concerns. It deals directly with the http request and extracts parameters from the http request. The web layer is a thin layer that delegates actions to the service later based on http request parameters. The web layer also knows what rights a user needs to have to perform actions. The web layer of the server side system is defined by the package `org.emath.web`.

2.2 Service layer

The service layer `org.emath.service` defines a set of coarse-grained services that are used by the web layer. The interface `Service` defines the contract of the service layer and the class `ServiceImpl` implements the interface. The service layer knows what needs to be created, modified or retrieved to fulfil the service. This layer handles

objects of the model layer and uses the persistence layer to save, modify or retrieve objects from the data storage.

2.3 Domain object model layer

The model layer defines the business objects of the system. These objects come in two packages: `org.emath.model.book` and `org.emath.model.communication`. The package book holds all model objects that are related to some kind of e-book content whereas the communication package holds objects that are needed for other reasons like Logging or communication with the client.

2.4 Persistence layer

The persistence layer comes in three packages: `org.emath.persistence.hibernate`, `org.emath.persistence.book` and `org.emath.persistence.communication`. So all packages of the persistence layer are named “hibernate”. This is partly misleading because the server-side system does not rely very strongly on hibernate as the implementation method of the persistence layer. The interface `DatabaseFacade` defines the contract by which the persistence layer is used and currently the `HibernateImpl`-implementation of this interface is used to gain access to the data storage. If the implementation method is to be changed to JDBC for example then this can be achieved by implementing the Jdbc-version of the `DatabaseFaçade`-interface. The persistence layer is just a tool to be used by the service layer to access the database. The aim has been to let the service layer handle the business logic of the application.

2.5 Data storage

MySQL is currently used as the data storage method of the server-side system. The persistence layer handles all access to the data storage. The relations between the database tables will be presented separately for all main actions.

2.5.1 Courses and users

All tables that are used to create new courses are shown in figure 3.

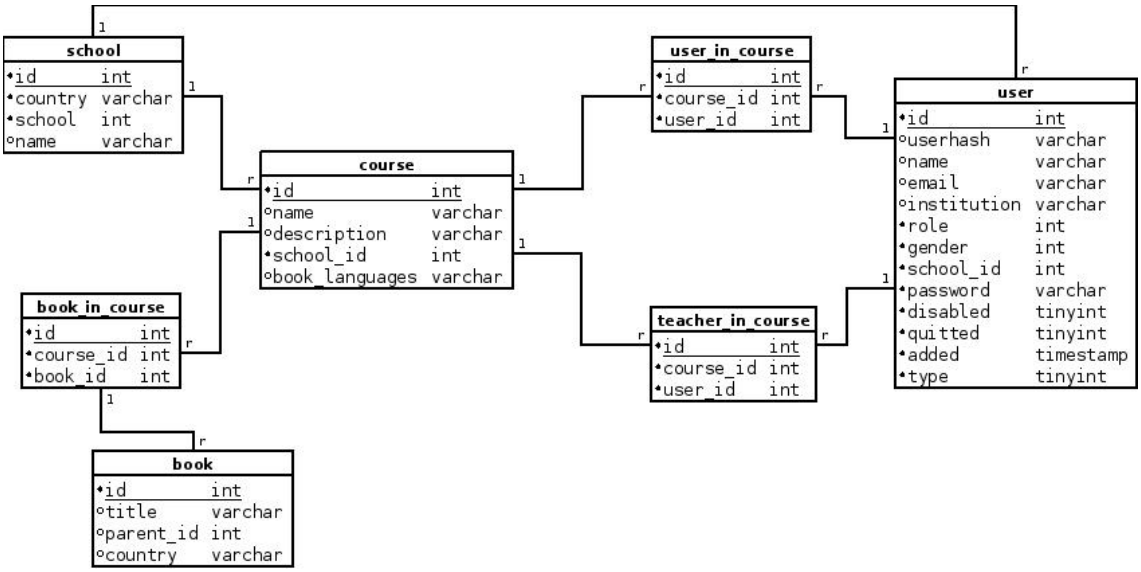

Figure 3: Course-related database tables.

Basically a course is held at a school, one or more books are used in a course and a course also has students and one or more teachers. This is all information that is needed to create a new course. When a user is validated, a username, password and course are needed. The username is `user.userhash`, the password is `user.password` and the course id is a 'C' followed by the `course.id`. The `user.role` indicates if the user is a teacher (1), student (2) or admin (3). A user is related to a school with the `user.school_id`. Setting the parameter `user.disabled` to anything else than 0 will disable the user. After that the user will not pass the user validation.

The type of user is indicated by `user.type`. These are full (1), demo (2) and without course management (3). The default value for this one is full. Demo is for dummy users that will not attend to any course and will also receive only dummy book updates. Without course management is used for teachers, which are not allowed to create students. The fields `user.name`, `user.email`, `user.institution` and `user.quitted` are for informational purposes only and they are not needed by the system.

2.5.2 Assignment solutions and the backup table

All tables that are related to solutions can be seen in figure 4.

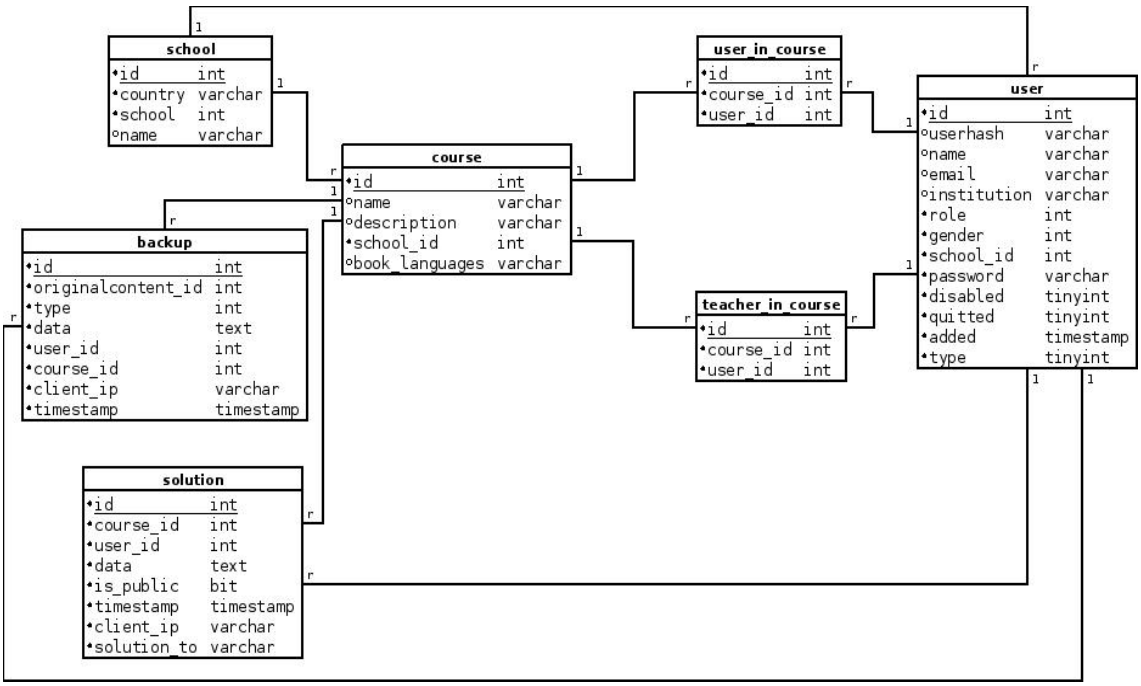

Figure 4: Tables that are related to solutions.

A solution is related to a course and to a user. If `solution.is_public` is 0, then the solution is only visible to the student who made the solution. If `solution.is_public` is 1, then the solution is also submitted to the teacher. `solution.solution_to` holds the assignments title-attribute of the assignment to which the solution is. The backup table is also visible, because all versions of a solution except for the newest are in the backup table. Only the newest solution for any given solution and student is in the solution table. This is also true for added content. When a solution is updated, the earlier version of the solution is first inserted into the backup table before overwriting the solution that is in the solution table. `backup.originalcontent_id` is the database id that the backed up element originally had when it was created into the solution/added content table. The `backup.type` indicates the type of content of the element. The type can be added content (0), solution (1) or comment (2).

2.5.3 Added content

Added contents work like solutions and that's why it makes no sense to repeat figure 4 here. The backups of added content are also saved in the backup table as mentioned earlier. The addedcontent table is visible in figure 5.

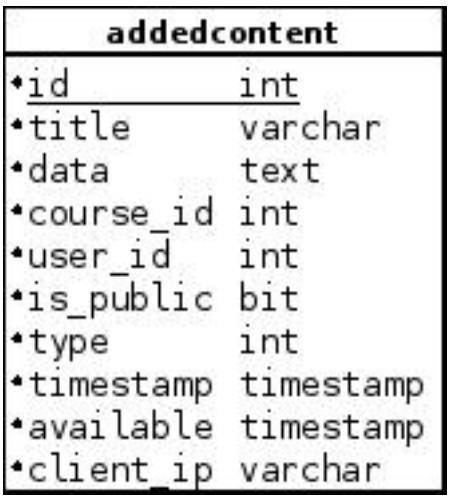

Figure 5: addedcontent table.

The `addedcontent.is_public` is used to define if the added content is only visible in the teachers book (0) or if students can also see the added content (1). The type indicates the type of content. That can be example (0), sd markings (1), home assignment (2), model solution (3), assignment (4), text (5) or theory (6). The attribute `addedcontent.available` is used to control when the added content will be visible to others than the teacher himself/herself if when `addedcontent.is_public` is 1.

3 SOLUTION PRINCIPLE

When an http request arrives at the web layer then the user credentials are extracted from it. The validation of username and password are delegated to the service layer method `isValidUser`. If the information is invalid then a `LogEntry` is created and the persistence layer adds the `LogEntry` to the data storage. If the user exists then the `isUserInCourse` service is called from the service layer next. The user needs to be either a teacher or a student in the course. An admin does not have to send a course id at all. If the authentication process finishes successful then the type of request is parsed, checked and after that type-specific variables may be parsed from the request. When all necessary variables have been parsed then the actual fulfilment of the service is delegated to the service layer, which will delegate the persistence layer if it needs to access the data storage.

4 TESTING

Almost all of the business logic of the system is in the service layer. Because of this the web layer and the persistence layer are very thin. Therefore there are jUnit/EasyMock tests for the service layer. EasyMock is used to mock the persistence layer of the system. In this way it is not necessary to control data storage content for testing purposes and also the service layer can be tested without the persistence layer. When the service layer is properly tested then any upcoming problems should either be in the web layer or the persistence layer. Because both of these layers are very thin, it should not be very difficult to spot problems in these layers. The tests for the service layer are in the package `org.emath.test.service`.