Reflective Report On Final Project

Kalvin (Xin) Wei
xwei040@aucklanduni.ac.nz

I. System design

As MVC pattern is a great examined practice in modern software development, we decide to apply it into our application at beginning. To figure out what we should have for each part of MVC becomes the first issue.

A. Architecture

- Model

We use relational database which is Mariadb to play the role of the data storage basis of our application. There are 3 essential database schemas *user, article, comment* and along with other 4 schemas *follow, likeArticle, likeComment* and *tag* in it.

Java beans. In order to facilitate data transfer within controllers and views, we don't employ the way that lets java bean classes simply map database schemas. For instance, *Article* class does not merely have fields in *article* table in the database. It also contains fields such as *user, tags, comments* which allows us to conveniently compare the current logged in user with author of the current showing article as well as authors of comments with only one *user* object and its many fields. In detail, we have: *Article, ArticleSummary, Comment, User, UserData, Error* to technically map database schemas but simultaneously quite well serving the data consumption in models and views.

DAOs. Also, we have DAOs to deal with the logics of CRUD actions from java code to data tables. What is worth mentioning is in this compound application rather than in labs we have several methods using rather complicated SQL statements to retrieve proper dataset back to controllers, more specifically, many inner join operations across different tables.

- View

We use JSP pages performing as views. Actually, this part is firstly considered and undertaken after the general structure is determined. We have 10 major JSP files to serve as user interface of our application, along with which, in order to mitigate code redundancy, we extract 6 web page components as separate JSP files to allow any major JSP to embed any number of them, such as comment blocks which would recur many times in article page.

- Controller

We have more than 30 servlet classes to catch browser requests and respond accordingly. We sort all the servlets into 3 types: (1) action servlets. These catch match request from JSP form submissions, do something, wrap data into request attributes or session attributes and then redirect to other page servlets. (2) page servlets. They respond to requests from pages or servlets, resemble data and then dispatch the requests to certain JSP pages. (3) ajax servlets, meaning that such servlets are invoked with ajax requests, and finally send json response to web pages which intend to asynchronously fetch data and refresh parts of a web page. Like in the process of checking a sign-up user if they input an already existent username which is a unique value in the data table.

- Utilities

We have several utility classes to make our application more flexible and robust.

AuthenticationUtils. It contains static methods to perform actions correlate with user authentication, login, logout logics. *authenticate* checks if a password matches one of a certain user. *checkLogInStatus* assumes the task to check if a user has logged in, if so, fill the current logged in user into a request attribute to allow it being retrieved for some usage.

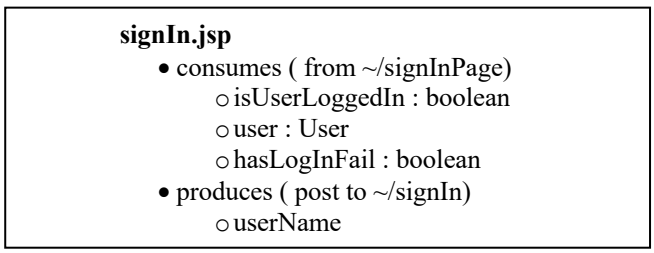
DBConnectionUtils. This is introduced from one of previous lab to render us separate database connection logic with the genuine transaction logics. We can easily configure database connection properties in *connection.properties* file.

JsonUtils. This also comes from our labs, allowing us to send a JSON object as a respond to an ajax request.

PasswordUtils. A utility class to greatly facilitate the hashing, salting procedures of users' passwords.

B. APIs

It contributes critically that we use prototyping tools to intuitively sketch the whole work flow of this blog application at first. Before starting with the solid coding work, we took advantage of the prototype, spent an entire day on figuring out how components should interact with each other. We made a table of all endpoints, which comprises JSP pages and servlets. How many JSP pages should we have to meet all requirements in the handout? To assemble and respond to the pages, how many and what kind of servlets should we have? What are entrance and exit of pages and servlets? What data do they consume and meanwhile what data do they produce? With answers to these questions, all things became clear for the following steps. Here we take sign in page as an example:



II. Work of myself

A. Front-end

- Prototyping

To quickly seize the key functionalities in the instruction and illustrate the entire main work flow as well as the basic UI of this application, I initiated to use prototyping tools to draw a blueprint of the system. I spent the first day to finish this, and it apparently helped much in discussion of APIs in the next day, since everything was just in front of our eyes as if the application was already there.

- Structuring all JSP pages

After we wrote down all parts of the first version APIs, I began to make drafts of JSP pages, giving them skeletons, using attributes values to fill the content of elements, setting request urls in forms actions to enable later servlets testing.

- Controls design and implementation

Throughout all the pages, except for the basic form controls provided by the browser, we design several customized controls to make user interaction more elegant and in the meantime level up UX and implement them all by my own. At places where users are required to upload an image for avatar or cover of an article, I don't coarsely place a file input in the form or under the image block but use some rather tricky approach to hide the file input and trigger the file selection window by clicking the image block itself, which renders much better UX and much more elegant page design.

This is not unique. In default avatar selection case, I hide radio buttons and bind their *checked* status shift to clicks of image tags adjacent, which also a way most likewise websites employ.

It is never an easy work to write a WYSIWYG editor from scratch, so I turn to use an existent multimedia editor for our project. In order to let it best fit our purpose, I looked into its documentation, found a proper configuration to need our particular need.

- Page stylings

We strike a consensus among team members that all pages have a uniform style. So, we decided to introduce Bootstrap framework to make the styling work much swift. Yihao Wang made several modification and brought about a package in which coloring scheme, shadow and some other CSS properties are customized. I worked on styling of 4 pages, complete styling on edit profile page and edit article page, rough styling on article display page and my blog page.

B. Back-end

- Implementation of DAOs

On top of the class definition work of Yihao Wang, I implemented majority of methods in UserDAO, CommentDAO and ArticleDAO.

- Some servlets

I contributed to the servlet that correlates to posting articles.

III. TOPICS TAUGHT IN USE

I believe most of topics taught in courses this semester are utilized in this project, but here I want to list some prominent ones:

MVC design pattern. This is the most important topic used, since it defines how our task is deconstructed and how the project should be structured. If we do not have a clear and deep comprehension towards the rationale of MVC pattern, we might probably blend data manipulation code into servlet as well as password hashing and so forth. If we do like this, a task can hardly be appropriately divided and allocated to team members. Meanwhile our code will be less readable and hard to maintain.

Due to the complete understanding of the core flow of Git, we solidly employed Git as the collaborative base for

DAO. Thanks to the initiative of DAO, we don't need to apply data manipulation code directly here and there in servlets or any other places where data manipulation does not play the main role.

Responsive Design. With a few extra code, we allow our web pages display as expected on any size of screens. This weighs even more in the current mobile-dominated age. In our practice, we use Bootstrap to easily gain a responsiveness feature.

Servlet. Servlets are like engines in the web application. We have to use them to catch requests from browser and organize a result to feed back. By now, we have learnt and taken into practice two means of response. To return a web page, we use request dispatcher to redirect the request to another servlet or a JSP page. Otherwise we can use JSON utility class to feed back with a pure json object. The later is used to detect whether a username has been taken in the sign up page.

Refactoring. Refactoring is a very useful soft skill to escalate our code. IntelliJ provides quite advanced feature to allow use to refactor our code effortlessly. I often use refactor feature to refactor variable names, replace types and also automatically revise paths as url when moving a file to another directory. A more sophisticated feature is it can automatically give out hints on extracting replicate code into methods or replacing verbose statements with concise ones or more advanced coding styles.

Git. Git is one of the most popular version control systems. We heavily relied on Git in this project. Due to its powerful history tracking technology and sophisticated fork-commit-sync-merge work flow, team work is well organized, collaboration is perfectly practiced.

IV. TOPICS AND TECHS UNTAUGHT IN USE

UI prototyping. As mentioned above, at the very beginning, we applied a software popular in industry Axure to build up the most important pages of our project. There are plenty of build-in virtual web components served to let front-end designers to take drafts of the final effect of web pages. What is stunning is the components in this software can even react to mouse motions, screen tap and so forth user actions. This software is an extraordinary tool for analyzing customer demands in a visual way before system design and implementation stages.

Bootstrap. Bootstrap has been popular for years due to its inborn trait of responsiveness. Along with years' evolution, now it has almost all common components a normal modern web page may contain, such as navbar, pagination, popover, etc. Its 12 column grid layout system also exemplified a good responsive positioning system of blocks in a web page, powerful meanwhile easy to use.

V. ON TEAMWORK

I assume the most precious thing I gain from this project was not the project outcomes. It was the process of how we worked as a team especially our capability did not perfectly match with each other. There are still many things we can conclude from the project, though we did not experience too many difficulties working in our team.

teamwork. Benefiting from the good beginning of having a rather clear task division and allocation, almost at a certain

period during the project, we three always had relatively independent parts to work on. Therefore, we rarely had merge conflict issues when handing in a completed feature.

Scheduling is really important when working as a team, everyone should definitely stick to it. But sometimes one might meet an unsolvable impediment and can not hand in their work on time. Then, the whole picture should be prioritized rather than personal capability concerns. We should keep trust in teammates as well as ourselves. Asking for help is not shameful. It is also a process of learning and gaining.

When I first tried to deal with the code of parsing image files from a http request, I assumed that from the FileItem list, I was only supposed to retrieve non form fields, i.e. file items uploaded via a file input, so I innocently tried to retrieve other parameters using *request.getParameter* method without knowing that once we set the form's *enctype* attribute to *multipart/form-data*, we can not retrieve form fields by normal way any more. Only after I saw the code from Yihao Wang, I realized we should pay attention to details and carefully research before use and do not take something new for granted with the knowledge of old similar ones.