Xinran Wang Individual Report

Xinran Wang

*dept.school of ITe University of Aucklanddd*Auckland,New Zealando
xinran.wang@auckland.ac.nz

I. Introduction

This particular report mostly discusses the final project, which we created as a complete system that included the front and back ends of the blog website. Additionally, it covers the overall system architecture and how different parts interact and fit together to make the whole. We needed two weeks to complete the final project, and throughout that time, we had to deal with numerous difficulties, disputes, adjustments, and evaluations. In a later section of this study, I shall discuss these topics. The things I learned in class and other strategies I looked up online while working on this website will also be included in my contributions to my teams' front-end and back-end projects. Finally, I'll outline the lessons I took away from working in a team, including the advantages and disadvantages, and talk about how my team overcame these difficulties.

II. THE WHOLE BLOG WEBSITE SYSTEM

A. ER diagram and SQL database

When we first received the final project handout. The most important thing that we discussed is the ER diagram and SQL database because these two elements will become our project's fundation. First thing is we need to understand how is this blog website works and what funcationaties this blog website has. We came up with an agreement that this website can let users can post articles, comment on them, like them, and subscribe to other users. Also, the UI for this website must be friendly to users, which mean users can easily find the notification bar in the home page, the notificaiotn bar should display every action other user interact with this user including oher users subscribe this user, like this user's article and comment on this user. We will design the website based on our ER diagram and SQL database. In the ER diagram, we have 9 tables which are notification, user_status, subscription, like_article, comments, image, articles, users and finally avatars. The users table includes fields for user credentials (username, password), personal information (real name, date of birth, gender, country, description), a reference to an avatar image, an authentication token, and an admin flag.The articles table includes fields for the user who posted the article, the article title, the timestamp, the content of the article, and a cover URL for an image associated with the article.The image table appears to allow for additional images to be associated with an article.The comments table includes fields for the user who made the comment, the content of the comment, a parent ID (which would allow for nested comments), the article the comment was made on, and the timestamp of the comment.The like_article table is used to track which users have liked which articles.The subscription table appears to be used to track subscriptions from one user (from_id) to another user (to_id).The user_status table has fields for user ID, logout time, total followers, new followers, total likes, new likes, total comments, and new comments. This appears to be for tracking user engagement metrics.The notification table appears to allow for tracking various types of notifications: likes, subscriptions, comments, and articles. It has fields for tracking who the notification is from and to, which article it relates to, whether it is a like/subscription/comment/article notification, when the notification was made, and whether it has been read.

B. Brief overview of the system and architecture design.

Based on the ER diagram and SQL database, we have a detailed plan for how to construct this blog website. This system will have a front end and a back end. The front end of the website has a number of pages, such as the home page, which displays all of the articles, the top five articles, and a number of buttons, including those that lead to the login page, the sign-up page, the personal page, and the page for each of the articles. The login button takes us to the login page, where we must enter our username and password to log in before we may subscribe to other users and like or comment on their articles. We are only able to read articles before logging in properly. In order to register as a new user, we must enter our username, password, and password again, as well as our gender, nationality, date of birth, and description when we click the signup button. The system will display a notice stating that "the username has already been taken" if we enter an already-used username, and a message stating that "the passwords are not matched" if we enter a password that has previously been used. To make our system more user-friendly, we believe these functionalities should be displayed immediately rather than after user submission. After signing up or logging in, we can access our personal page by clicking the "ME" icon on the home page. We can upload articles, look over or remove followers, likes, and comments on the personal page. Additionally, we constructed an analytical centre that would show all of the remarks in a histogram model. This is just a quick summary of how our system works. We also have a lot of details, which I will discuss in more detail in my contribution section.

C. Highlight targeted problems and solutions

We run into some challenges and issues when working in groups and pairs. For instance, we merged our codes into a fresh release branch every morning. However, there were numerous disputes while merging codes and pulling requests.When we originally tried to force code merging, several of our codes vanished. Additionally, I unintentionally pulled my coding to the main branch rather than the release branch when we pulled requests for our code together. We need the main branch to be the original branch for the final commit version, so this is a significant error. However, if we pull certain codes to the incorrect branch, we can easily repair this issue by going back to the previous version. The second issue we ran across when working together was that occasionally, we misread each other's intentions. For instance, I once talked with my group

leader on how to build the individual page. His idea is to put a "like" button to the top of each page, and when a user hits it, a list of all the articles that other users have previously "liked" will appear. This is a fantastic idea with a very user-friendly design, but I worry that the user may be confused by this feature because there is already a "likes" button in the section of the website that asks for personal information. All of the articles that this user likes will be connected by this "likes" button. My theory is that having two "likes" buttons on one web page is confusing since a third user won't understand the distinction or why they have two "likes" buttons. Then, by using the Instagram app, our group leader gave me a practical illustration of the "likes" and "liked" function. When I finally understood how the personal page should look, we were able to construct it.The third issue we ran into during the collaborative effort was that occasionally we couldn't decipher the codes of others.For instance, I had to leverage other people's code, such as the DAO file, when creating the personal page since I needed to gather the user's likes, comments, and follower information. However, when I used these codes, I discovered that the likes, comments, and following counts were off.Then, using the console.log function, which is not a very effective technique, I have to figure out which section is incorrect. The following day, after discussing this issue with my colleagues, they assisted me in debugging it, which reinforced the importance of teamwork and communication to me. We need to provide our problems and feedbacks during teamworking so that we can solve these problems together.

III. MY CONTRIBUTION TO THE GROUP

As a team member, it is my responsibility to design the fundamental features that underpin all blog websites. after we constructed the SQL database and ER diagram. Our group leader began organising the jobs, and I was given both front end and back end tasks. I'll discuss the front end and back end tasks I completed for the group project in this section of the report.

A. Front End

About the front end. Building the personal web page was my responsibility, and I completed the basic structures for all of the new navigations it contains, including the subscriptions web page, subscribers web page, comments web page, likes web page, edit profile web page, analytics centre web page, notification web page, and personal centre web page. The CSS portion of improving the appearance of these web pages was then assisted by my teammates.

First, for the personal page, my idea was using hyper links that we learned in COMPSCI 719 lecture to conect other pages which should be very easy, I can use the anchor to link other pages. However, the hard part for me is the “liked” button. Our requirement is when we click this “liked” button, our page should stay the same url but the content should be changed and displayed all the articles that user liked before. I was stucked by thinking how to implement this function. Then I reviewed the PPT slides and found out that I can insert {{#if}} statement and {{#each likedArticles}}in handlebars to display each articles that the user liked before. These statements can also be used when I navigate to other user’s personal page. Our logic is when I go to my personal page, I can review the analytic center and click the edit profile button. However, when I navigate to other’s personal page, I am not allowed to view their analytic page nor click their edit profil which means these buttons should be disappeared. In this case, I can also use the{{#if}} statement such as {{#if isMyAccount}}and put these methods inside this {{#if isMyAccount}} statements.

The second web page I want to talk about is analytic center page. Although my teamate has implemented all the functions but when I was trying to put them all together in a single web page. The thrid party plugin histogram app was not working. When I tested it several times and it didn’t work, I compare my code with my teamate’s code then I found out that there was some missing codes in the thrid party plugin histogram app. Then I learned how to put the plugin app in my handlebars. Although this is not difficult, but at that time I learned lessons about how to insert a histogram inside the handlebars by learning my teamate’s codes.

B. Back End

For the back end of this system. My duties are implementing the login system with hashing and salting password stored in the database, re-editing profile after sign up as a new account, designing the like and dislike function in the article page and building the subscriptions and subscribers function in the personal page.

First of all, for the login system with hashing and salting. This is the first task for me and I didn’t find much information in the lecture slides. So I searched online about this hashing and salting password. I found out that the different functions about hashing and salting:

- Hashing is when employed for data verification, hashing transforms plaintext data pieces into consistent ciphertext outputs.
- Salting is to prevent hackers from decoding sensitive data by searching for recurring words and phrases in it, salting adds random characters to data, such as passwords.[1].

In JavaScript, we can install the bcrypt package for hashing and salting password. After install bcrypt, we can input “var bcrypt = require('bcrypt');” and “const saltRounds = 10” which will salt our passwords for ten rounds. After this, we can write the function hashPassword to make sure our passwords are hashed and salted in the database.

My second task is make sure after sign up, the user can re editiing his/her profile including the gender, username, password, country, description and date of birth. This back end function is much easier than the first one. All I need to do is redirect this link back to the sign up page, however, I need to change the “sign up” title to “edit profile” so that the users can re-edit their profile and when they refresh the page, the new information will be displayed in the personal web page.

The third task is building the likes and dislikes functions. User are allowed to like and dislike both other articles and his/her own articles. The key point is each user can only like or dislike each article once. So the first thing I need to do is create a like-dao file which stored all the like and dislike functions in this file. I created two functions called addLike

and removeLike. For addLike, the function is : INSERT INTO like_article (user_id, article_id, datetime) VALUES (${userId}, ${articleId}, datetime("now","localtime")) and for removeLike function is : DELETE FROM like_article WHERE user_id = ${userId} AND article_id = ${articleId}which are very simple. I can then create routes in my route file to leverage these functions. For the article route handler for "Like":Processing "like" actions for a specific article is handled by the router.post("/articles/:id/like", verifyAuthenticated, async function (req, res)... function, where:id is the id of the article being liked. verifyA middleware function called authenticated determines if a user has been authenticated or not.The user id is obtained from the authorised user data via the formula const userId = res.locals.user.user_id.Gets the article id from the request arguments with const articleId = req.params.id.

The function then uses likeDao to determine whether the user has already liked the content or not.articleHasBeenLikedByUser(UserId, ArticleId). If not, likeDao is called by the function.addLike(userId, articleId) adds a user-generated like to the article.It then uses likeDao to obtain the total number of likes for that article. the client receives the result of getLikeCountByArticleId(articleId) as a JSON response.Last but not least, notifyDao creates a new notification that a like has been made on the author's article if the author is not the one who liked the post. Create a new notification with the parameters (res.locals.user.user_id, article.user_id, "like", articleId).

Regarding the disliked routes,a router. Similar but in the other approach is how the Post ("/articles/:id/unlike", verifyAuthenticated, async function (req, res)... function operates. It handles "unlike" actions related to a certain article. If a user liked the article previously (likeDao.The function likeDao is called when userHasLikedArticle(userId, articleId) returns true. To remove a like from an article, use the Remove Like (userId, articleId) function.

Following that, it obtains the most recent number of likes and returns it to the client as a JSON response. It doesn't produce a notification, in contrast to the "like" handler.

Last but not least, I need to implement the client side functions after the back end operations are finished. I used the addEventListener function in my js file to listen for the like and dislike buttons. A post request to the HTTP will be fetched by JavaScript when a user hits the "like" or "dislike" button to indicate whether they like or dislike the article. I can then use the response.json() method, which is used to parse a server's response body into JSON format. A JavaScript object is the Promise that this method returns as its result.

The function for subscriptions and subscribers was the last task I completed. Every user in the system has the ability to subscribe other users, as well as unsubscribe any users they choose. The like and dislike function and this task are really similar. I started by making a subscription-dao file to implement all the functions I would need to do this operation, including the createSubscription and deleteSubscription functions. SQL should read "INSERT INTO subscription (from_id, to_id, datetime) VALUES ($from_id, $to_id, datetime("now","localtime"))" for creating subscriptions, and "DELETE FROM subscription WHERE from_id = $from_id AND to_id = $to_id" for deleting subscriptions; Then, same as the like and dislike function, I need to create the routes for these functions. Similar to the like and dislike routes, the subscription and subscribers routes are very similar. First we have router.post("/subscribe/:id", verifyAuthenticated, async function (req, res) {...}:

At the "/subscribe/:id" endpoint, where ":id" denotes the user id to be subscribed to, this route watches for HTTP POST requests. The user making the request is verified using the middleware function verifyAuthenticated.

Const from_id = res.locals.user.user_id; in the callback function retrieves the user_id of the authenticated user from the local variable of the response object, and const to_id = req.params.id; retrieves the id of the user they intend to subscribe to from the request parameters.

The function then executes await subscription in an attempt to establish a new subscription record in the database.createSubscription(from_id, to_id); in the Dao API. If this action is successful, a JSON response with the status and a success message is returned to the client. By using await notifyDao.createnewNotification(from_id, to_id, "subscribe"), it also produces a new notice for the user who is being subscribed to.

The function detects errors during the subscription process and returns a JSON response to the client with the status and a failure message if one occurs.

Next, we haverouter.delete("/subscribe/:id", verifyAuthenticated, async function (req, res)...

Similar to the previous function, this route monitors the "/subscribe/:id" endpoint for HTTP DELETE requests. The function attempts to delete the subscription record by executing await subscription if the user is authenticated and a subscription exists between the authenticated user and the user supplied in the request parameters."Dao.deleteSubscription(from_id, to_id);" A JSON response with a status and a success message is given back to the client following a successful deletion. In the event of a failure, a JSON response including the failure message and the status is instead returned.

We now have the router.obtain ("/isSubscribed/:id", verifyAuthenticated, async function (req, res)"):

This route monitors the "/isSubscribed/:id" endpoint for HTTP GET requests. By invoking const isSubscribed = await subscription, it determines whether a subscription record between the authorised user and the user supplied in the request parameters exists.isSubscribed(from_id, to_id); in the Dao.

A JSON response containing the statuses "Already subscribed" with value 1, "Not yet subscribed" with value 0, and the result is provided back to the client, indicating whether or not a subscription exists. If a problem arises while the process is running, the function detects it and sends a JSON response with a failure message and a status of 0.

I started to work onI started working on the client side for the subscription and subscriber functions after finishing the back end tasks.In order to handle the functionality of subscribing to and unsubscribing from authors, as well as managing the navigation for viewing subscribers and subscriptions of a user, I wrote a client side JavaScript file and used jQuery. Const authorId = $('#subscribe-button'); and const subscribeButton = $('#subscribe-button') were

used.data("authorid");: These lines each define a constant for the author id and the subscribe button.The initial AJAX call determines whether the user is currently subscribed to the author. A GET request is sent to the URL /isSubscribed/$authorId.The text of the subscribe button changes to "Unsubscribe" if the user has subscribed (status === 1); otherwise, it remains "Subscribe".The click event for the subscribe button is then handled. A POST or DELETE request is sent to /subscribe/$authorId in order to subscribe to or unsubscribe from the author, depending on the button's current text. The button's wording changes from "Subscribe" to "Unsubscribe" depending on whether the operation was successful (status === 1). If not, a notice with an error is shown.

The $(document).on('click', 'a[href="/subscribers"]', function (event)... and $(document).on('click', 'a[href="/subscriptions"]', function (event)... functions stop users from clicking the "/subscribers" and "/subscriptions" links by default and instead direct them to the "/user/$userId"To unsubscribe from a subscriber, use the $(document).on('click', '.unsubscribe', function (event)... function. It stops the clicked link's class "unsubscribe" from doing its default action and sends a DELETE request to./subscribe/$to_id, where to_id is the data property of the clicked link. If the operation is successful (status === 1), the page is reloaded to reflect the changes. If not, an error message is displayed.

So, these are the contributions that I finished during group work time in both front end and back end.

IV. INDIVIDUAL PROJECT

After we finished the group project, I started to finish my own individual project which is a GUI program that connect to the server and website. This GUI program only allows admin user to login and check all the information that each user signs up in our website. Also, I can delete other user's account which will also be deleted in the database.

A. API

I wish to concentrate on linking the API in my Java Swing programme first. As a result, I added an API class to my Java web package. I must ensure that the HTTP queries sent to a remote API are contained within this API class. BASE_URL indicates that the server is located at "http://localhost:3000".I employed the Singleton Pattern in this class to ensure that only one instance of it could be produced. The getInstance() method can be used to obtain this instance. For CookieManager and HttpClient: Java 11 added the HttpClient class, which is used to send HTTP requests. It takes the place of the more dated HttpURLConnection class. This class's HttpClient is set up to utilise HTTP/1.1, have a 10-second connection timeout, ignore redirection, and manage cookies with the cookieManager.Additionally, I have the logoutUser() and loginUser(String username, String password) methods, which are used to log a user in and out, respectively. The login method makes a POST request with the user's username and password in the body to the "/api/login" endpoint. If the response status code is 204, a new User object is created and returned with the supplied username, password, and isAdmin property set to true. The GET request for the "/api/logout" endpoint is sent by the logout method, which returns true if the response status code is 204. Additionally, to obtain all users, use the getAllUsers(User requestor) function. Only when the requestor is an admin does it send a POST request to the "/api/user" endpoint. If the response status code is 200, it transforms the response body from JSON to a list of User objects using ObjectMapper before returning it.Last but not least, a user can be deleted using the deleteUser(String id, User requestor) method. Only if the requestor is an admin does it send a DELETE request to the "/api/user/id" endpoint. If the response status code is 204, it returns true. If not, it returns false and 401 code and prints an error message.

B. User, UserService,UserTableModel and UserAdapter

I generated the blog package and the adapter package after finishing the API class. There are three classes in the blog package: the User class, the UserService class, and the UserTableModel class. The UserAdapter class is contained in the adapter package. Regarding the User class (Blog package): A user and all of their characteristics are represented by this model class. It contains a variety of user details, including user id, username, password, true name, birthdate, gender, country, description, and whether the user is an administrator. Additionally, it includes some straightforward getter and setter methods for these attributes. For the adapter package's UserAdapter class: This class serves as an adaptor, mostly wrapping the User class to make it easier to utilise in other contexts. In order to make it simpler for other code to get user information, it primarily offers certain additional methods, such as getFullName() (which returns a mix of the username and full name). This is a service class for the UserService class (Blog package), which basically manages user-related tasks including logging in, logging out, getting all users, and deleting a user. Internally, it makes use of the API class, which offers HTTP request functionality for server communication.A model class for showing user information in a graphical user interface (GUI) is the UserTableModel class (Blog package). It defines a table model that may be utilised in a JTable component by extending the Java Swing library's abstract AbstractTableModel class. The UserTableModel class has methods for obtaining each cell's value as well as the number of columns, rows, and column names. A list of UserAdapter objects is stored in each instance of this class for use in presenting the table's data for each user.In summary, these classes are related in that the UserService gets user data from the server, puts it into User objects, and then uses the UserAdapter to wrap those objects. The UserTableModel receives these adapter objects after which the GUI displays them.

C. BlogApp

Finally, the last step is create a BlogApp class in the UI package.This class, BlogApp, represents the graphical user interface (GUI) of a blog application and extends the JFrame class from the Java Swing library.This class has several components:

- Username field and password field: Text fields for entering a username and password.
- Login button,logout button, and delete user button: Buttons for logging in, logging out, and deleting a user.

- User data table: A JTable for displaying user data.

The BlogApp class also keeps a userService of type UserService to handle user-related tasks and a currentUser of type User to represent the currently authorised user.These elements are initialised and positioned in the relevant areas of the frame in the constructor BlogApp(). When the loginButton is clicked, an action listener that is connected to it authenticates the user. All users are retrieved from the server and shown in the userDataTable if login is successful and the authenticated user is an admin. An error message is sent if the authentication is unsuccessful or the user is not an admin.Additionally, the logoutButton contains an action listener that, when activated, logs the current user out and clears the user data from the userDataTable.

When clicked, the deleteUserButton's action listener deletes the selected user from the server. The table is updated to reflect the deletion if it is successful. A notice is displayed if the deletion fails. A helper function called convertToUserAdapterList transforms a list of User objects into a list of UserAdapter objects. In addition to logging out the current user, the logoutAndShowErrorMessage method also shows an error message.The User and UserService classes are used to manage user-related data and actions, the UserAdapter class is used to adapt the User class for the GUI, and the UserTableModel class is used to show user data in the userDataTable. These classes have a relationship with the code in the preceding section.

TOPICS TAUGHT IN CLASS THAT I USED

A. Node

For the Node part, I used both front end and back end technologies that we learnt in class. For the front end part, most of my work contains handlebars, client side JavaScript and css modeling. For the back end part, I used DAO to collect the data stored in the database, then I used routes handlers to connect my data to the server and finally use handlebars to display these data. These are the methods that I used when I was building the like & dislike functions and subscriptions and subscribers functions. Also, when I was implementing the login function, I used the auth-middleware functions such as addUserToLocals() which retrieves the current user from an auth token in cookies and adds it to res.locals. The second function is verifyAuthenticated() which checks if the user is authenticated. If so, it continues the request. If not, it redirects the client to the login page.

B. Java

For the Java part, first thing that I used is the Java Swing package which is required when building a GUI progam. Then I used design pattern such as adapter. The UserAdapter class provides methods for accessing the User data in a way that's suitable for my GUI. For instance, the getFullName() method concatenates the username and real name of the user, which may be useful for displaying in certain parts of the GUI.

TOPICS NOT TAUGHT IN CLASS

A. Node

For the Node part, there was some topics that I searched online when I was finishing the login system which is hashing and salting. .

B. *Java*

For the Java part, the first topic is the API which I didn't learn at class. I searched online to learn how can we connect the API from server side to client side.

The second topic that I didn't learn is the JSON in Java, when @JsonIgnoreProperties(ignoreUnknown = true) is used: This annotation instructs Jackson to disregard any JSON attributes during deserialization that are not mapped to fields in the User class. This helps to avoid problems during the deserialization process when the JSON may have more properties than the Java class.

The annotation @JsonProperty("property_name") displays the JSON property name. For instance, Jackson is instructed to map the user_id JSON property to the userId field in the User class via the syntax @JsonProperty("user_id"). When Java field names conflict with JSON property names, this technique is used to handle the situation.

LESSONS I LEARNT FROM TEAM

I picked up a lot of helpful knowledge from my teammates throughout collaboration and pair programming sessions. My first lesson was to share ideas with others. When I was creating the functions, I would occasionally question my teammates about how to utilise them and in what circumstances. They would then offer their suggestions, which would often serve as sources of inspiration for me. The second thing I picked up was effective code debugging techniques.Initially, I struggled to debug the code because I lacked much experience. However, when we tested my code, my teammates assisted me in finding bugs. Then I discovered how to debug my method using console.log.

There were also some difficulities when we did the teamwork. At first day we didn't talk to much, everyone was just doing their codes. Then we found some comflicts and some of us did the same function twice, which is not very effective. Then we used the app Trello to distribute tasks. The second challenge is after we merged our codes together, the web page css had conflicts and some of our functions disappeared because of the css model. Also, when we were testing our system, if we input the username or the article title too long, the text would break the css model and overflew the webpage. Then we adjusted our css for two days and limited the input string so that the article title or username wouldn't ran out of the screen.

Conclusion

I learnt a lot of essential information and technologies because this was my first experience pair programming and collaborating with others. Despite the difficulties and hurdles I faced, my teammates' support in helping me overcome them together was greatly appreciated. Because of the extensive coding experience I gained from this group project and the lessons I learned about teamwork, I now feel more at ease working with others.

REFERENCES

[1] A. BURCHFIEL , "Hashing vs Salting: How do these functions work?"JUN 23, 2022