SW Engineering CSC 648/848-04

Team 04
Leiyi Gao: Team Leader
Justin Mao: Backend Leader
Yinyin Wu: Frontend Leader
Michael Han: Git Master
Nicholas Hamada: Scrum Master

Milestone 4: Testing

11/27/2022

Project Name: RateMyResume

CSC 648/848 Milestone 4: Testing

1) Product summary (e.g. how would you market and sell your product – max ¾ page)

Product Name
RateMyResume

Committed Functions
Registration – Users are able to register an account and the credentials will be saved in the MongoDB database.

Login – Registered users are able to login

View all the posts from the users – Users are able to view all the posts on the app.

Comment – Users can comment on posts.

Like – Users can like posts, and users can see the number of likes on each post.

After we implemented all these functions, we conducted JUnit tests on the following functions to satisfy the requirement:

Unique Features

1. Human feedback - Users can get their resumes reviewed by human eyes and get feedback to improve their resume
2. Free and private - Service is free and we don't sell users' data. Users should make sure private information are redacted from their resume since our app does not offer this feature.
3. Community sourced **-** As more users benefit from the app, we hope to see them return as reviewers to help others. Users who had a positive experience may also bring additional users to our app.

URL: **http://ratemyresume.ninja**

2) QA testing- max 2.5 pages

I. Unit Test

We wrote one test for each P1 feature and each helper function to our P1 features. Overall, we covered all of our P1 features with unit tests; we implemented tests for creating and logging in as a user, creating and retrieving a post from the database, commenting on a post, retrieving comments for a post, liking a post, and getting the number of likes in a post.

URL to unit test source directory

II. Integration Test

URL to report

3) Code Review:

1. URL to pull requests:
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/26
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/27
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/28
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/30
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/31
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/32
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/33
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/34
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/35
https://github.com/CSC-648-SFSU/csc-648-project-csc648-04-fa22-team04/pull/36

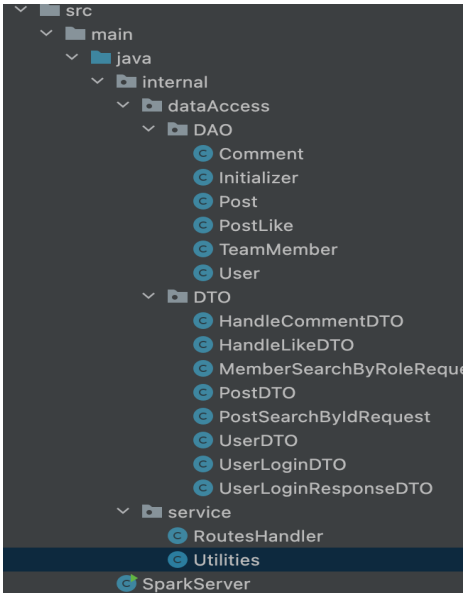
2. Coding Style

a. For the backend, we keep our codes organized inside of the Utilities file, where it contains all of our methods for major functions.

```java
/**
 * Uploads the file located at the filePath and fileName given
 * @param filePath The location of the file
 * @param fileName Acual file name
 * @param imgDb Mongo database reference object
 * @return Stored file's object id in MongoDB
 */
public static ObjectId upload(String filePath, String fileName, MongoDatabase imgDb) {
    ObjectId fileId = null;
    try {
        // Create a gridFSBucket
        GridFSBucket gridBucket = GridFSBuckets.create(imgDb);


        InputStream inStream = new FileInputStream(new File( pathname: filePath + "/" + fileName));

        // Create some customize options for the details that describes
        // the uploaded image
        GridFSUploadOptions uploadOptions = new GridFSUploadOptions().chunkSizeBytes(1024).metadata(new Document("t

        fileId = gridBucket.uploadFromStream(fileName, inStream, uploadOptions);

    } catch (Exception e) {
        e.printStackTrace();
    }
    return fileId;
}
```

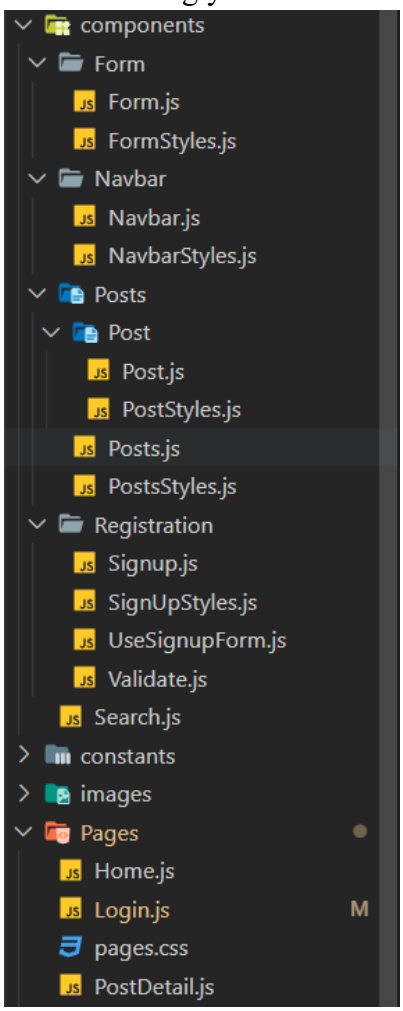
-As the example shown above, we will define what the method does first.
-Then we explain the parameters inside of the method.
-Lastly, we need to show what is the return for the method.

The way we enforce our Coding Style is to strictly keep our codes organized, in which we set up all the DTO and DAO classes, Utilities file which contains all the methods and RoutesHandler to call the methods for our endpoints. Example shown below:



In this way, we can keep everything on track, and fully follow our guideline for our coding style.

b. For the frontend, we use ESlint to check syntax and identify problems in the code, and enforce code styles following Airbnb React/JSX style guide and the help of Prettier plugin. After following the airbnb style guide, we fix filenames and indentation accordingly.




```
<Navbar></Navbar>
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/search" exact element={<Search />} />
  <Route path="/user/login" exact element={<Login />} />
  <Route path="/user/create/" exact element={<Signup />} />
  <Route path="/post/:id" element={<PostDetail />} />
</Routes>
```

For examples

- We use PascalCase for filenames, and for their instances, we use camelCase.
- We also follow the guideline to include a single space in self-closing tags

Lastly, we add appropriate documentation comments for functions

```
/**
 * Signup component for users to create an account
 * @param {string} username - username to register for an account
 * @param {string} password - password > 8 characters and at least 1 digit
 * @param {string} email - email must obey regular expressions rule
 * @returns user object created in database
 */
```

For this comment,

- begin with a short description of what the component does.
- include and explain what the parameters do
- include its return, whether is an object/component/test

`Search(keyword)`

function to filter posts using keyword

Parameters:

Name	Type	Description
keyword		anyword match in the description

Source: components/Search.js, line 17

Returns:

a lists of posts with input keyword

`Signup(username, password, email)`

Signup component for users to create an account

Parameters:

Name	Type	Description
username	string	username to register for an account
password	string	password > 8 characters and at least 1 digit
email	string	email must obey regular expressions rule

Source: components/Registration/Signup.js, line 23

Returns:

user object created in database

`validate(values)`

function to validate if user inputs are valid

Parameters:

Name	Type	Description
values	string	input value

Source: components/Registration/Validate.js, line 7

Returns:

error message if any

---

http://127.0.0.1:5500/jsdoc/global.html

Lastly, we store the information using jsdoc where we have access to variables and methods definitions.