SW Engineering CSC648/848 Fall 2022
Milestone 4
Section 2 - Team 6

Garage.io

http://ec2-44-203-93-29.compute-1.amazonaws.com/

Thomas (tmichel@sfsu.edu) **Team lead / Back-end lead / DBA**
Mohammad (malhabli@mail.sfsu.edu) **Front-end lead**
Joe (jsand@sfsu.edu) **Back-end Engineer**
Tyler (tfulinara@sfsu.edu) **Back-end Engineer**
Nathneal (ngebre@sfsu.edu) **Front-end Engineer / Github Master**
Young (yso@mail.sfsu.edu) **Front-end Engineer**
Jiaming (jzhao19@sfsu.edu) **Front-end Engineer**

Revision history

12/2/2022 Final Check and Review of M4 Document

Product Summary

Garage.io

- Unique Features
  - One Click Application
    - Students are able to apply for job postings at ease by being able to only to press one button to be able to send all of their information to the matching company
    - This is done by students uploading things such as cover letters and resume beforehand to their profile and it automatically being attached to their job post with the correct matching job information
  - Favorites List
    - Students are able to add their favorite job posts to a favorite list in their student profile
    - This is done by students simply going into their profile and adding any job posts they have currently to the their respective favorite list
  - Connectivity
    - Students are able to add matching social media profile to their student profile
    - This is done simply by hooking up their profile directly inside their student profile
    - Linkedin, Facebook, etc are included.
  - Tech Job Focused
    - We at Garage.io would to focus job finding and job searching to key high topics that students may find interest in
    - We achieve by using the use of our filters, where we have 9 main categories that students may use to filters out different job posts such as Artificial Intelligence

Link to Garage.io's :

http://ec2-54-82-238-201.compute-1.amazonaws.com/

Final Committed Functions:

1. Student:
   A. Students shall use the search bar to search for jobs without being authenticated
   B. Students shall be able to register their own account
   C. Students shall be able to log into their account
   D. Students shall be able to log out of their account
   E. Students shall be able to upload a picture for their profile
   F. Student shall enable/disable alerts for matching job interests
   G. Students shall have access to a "one-click" application for jobs, while recruiters shall be able to set boundaries to accept applicants from said applicants
   H. Students shall be able to display results by clicking
2. Company:
   A. Companies shall be able to register their own account
   B. Companies shall be able to log into their account
   C. Companies shall be able to log out of their account
   D. Companies shall be able to upload their company logo

E. Companies shall submit multiple job posts to the platform
F. Companies shall have the list of all their job posts available on the platform
G. Companies shall access student applications for a job post

3. Administrator:
    A. Administrators shall trigger alerts for matching job interests for every student that enables this option
    B. Administrators shall be able to log onto the admin dashboard
    C. Administrators shall be able log out of the admin dashboard

4.Search:
    A. Job posts shall be searchable by job area
    B. Job posts list shall be filtered by category and by job type

2. Usability Testing

Major Function to be test: **Search Function**

Test Objective:

Searching is one of the most basic functions within this web application, since it offers the customers a helpful tool to narrow down the results. Whether the searching function provides the user with convenience and whether it correctly responds to the user's input will make the first as well as the most important impression. Moreover, the validation for input is also necessary and should be tested before the final delivery. On one hand, we should restrict the length and content of user's input to avoid any inappropriate code injection. On the other hand, if the input is valid, we should display the items that approximately match the user's search text. All of the test objectives mentioned above are actually trying to collect feedback from users, in order to help our development team bring better user experience for both existing and potential customers.

Test background and setup:

System Setup:

From the user's perspective, the setup process is fast and simple. From the user's perspective we see that there would be no installation requirements. The user needs a computer with access to the internet, they will need to open their preferred browser (we have tested Opera, Chrome, edge, and Mozilla) so we recommend users use those browsers until we are able to try more browsers. The user will the n need to type in our URL and start working

Starting Point:

The starting point of usage of the search function is the home page, when the user accesses the provided link they will be led to the search page where they will find the search bar at the top where they will be able to type in what they are looking for. If what
the user enters is valid they will receive the search results otherwise they will be prompted to make valid inputs.

Intended User:

There are two intended users. The user may be a returning customer with a search history, a new user who just signed up or an anonymous user who is browsing through the available jobs before deciding to sign up. Therefore the intended user is anyone who is interested in applying for the jobs listed weather or not they have an account already with garage.io

Url of system to be tested:

http://ec2-54-82-238-201.compute-1.amazonaws.com

Usability Task Description:

Task Number Task for Usability Testing
1 search for a job as an anonymous user For searching as a new user we will be looking at how quickly and accurately the users are able to navigate the website. We will look at what we need to highlight to make sure the essential parts are more clearly visible for first-time users
2 search for a job as a returning user For returning users we will be looking at what information that they stored was likely to make them want to return, we want to make getting an account worth it by saving essential progress made so that the customers will be encouraged to keep visiting the website
3 search for a job with several filters used We will be testing the search feature along with the filters to make sure we have not put up too many restrictions and that the users are able to come back with what they are looking for.

Lickert Subjective Test:

Search function is convenient to use
☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

The user interface is easy to use all functions
☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

Returning to the page is easy to pick up where you left off
☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

3. QA test plan

Test objectives:

We will be testing the overall user experience to check how they were able to navigate the page and find what they are looking for. We will see if their experience was matched to the quality that we assured as garage.io.

HW and SW Setup:

Server Host: AWS Free tier

Amazon Web Service - t2.micro: 1 vCPU, 1GiB RAM

Apac has been successfully set up, which is required for running PHP. PostgresSQL has been successfully set up, which is necessary for running the database. PHP has been successfully set up, which is the back-end programming language.

Web Browser:

- Chrome
- Firefox

Validation of input (Main Features to be Tested):

Student/Company Account Creation, Student/Company Account Updating Information, Student/Company Account Authentication, Admin have all correct Capabilities, Searching for Job Posts, Use of Filters to correctly and accurately filter job post results, One-Click Application,Use of Job Alerts and Job Notifications

Approximate matching:

Main Features are either pass or fail. The questionnaire questions are correctly matched to the functions that we prioritize here at garage.io.

Responsiveness:

We check responsiveness by checking if the function that we have provided correctly works as intended. We check if the student and company profiles are able to be created and updated. We check if searching a job through filters and the search bar are correctly implemented properly. We check if the job alerts and notifications are correctly being sent to the student as well as being able to be triggered by the admin.

Test Plan:

We plan to give out questions pass/fail that allow us to see whether or not our main functions that we promised are assured. Our main goal is to check if the quality of the website in totality matches up with the promises and deliverables that we assured as the start-up company Garage.io.

Questionnaire:

#1 Creating Profiles

Able to both create a Student and/or Company Profiles

☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#2 Updating Profiles **Able to Update both Student and/or Company Profiles**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#3 Authentication **Authentication is needed for applying for a job**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#4 Searching Input **Search bar is able to take any input and give out a search result**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#5 Searching **Students are able to search for job posting with use of the search bar**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#6 Filters **Students are able to use the filters on the left to be able to filter job post results**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#7 One-Click Application **Students are able to apply for jobs through one click**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

#8 Job Alerts **Job Alerts are received by students and triggered by Admins**		
☑ ~~Pass~~	☐ Fail	Google Chrome

☑ ~~Pass~~	☐ Fail	Firefox

4. Code Review:

Pulls Requests:

- Development Branch is created
    - It is used to merge all commits from different teams before being merged to the main branch
    - One must have correct approval to the development branch before a commit may occur
    - Github master will then merge the corresponding development branch to the main branch once milestone has been fully reached
    - Review and Approval must be sent out to the engineer team that the commit was used for (e.g Back-End Team or Front-End Team)
    - If commit was not approved then the one who made the commit will have to update or recommit the new information that matches up with the other member's work
    - Work is done locally and checked/approved through the github development branch

Code style:

- Offers uniformity to the code created by different engineers.
- Enables the creation of reusable code.
- Makes it easier to detect errors.
- Make code simpler, more readable, and easier to maintain.
- Designed to boost programmer efficiency and generate faster results.

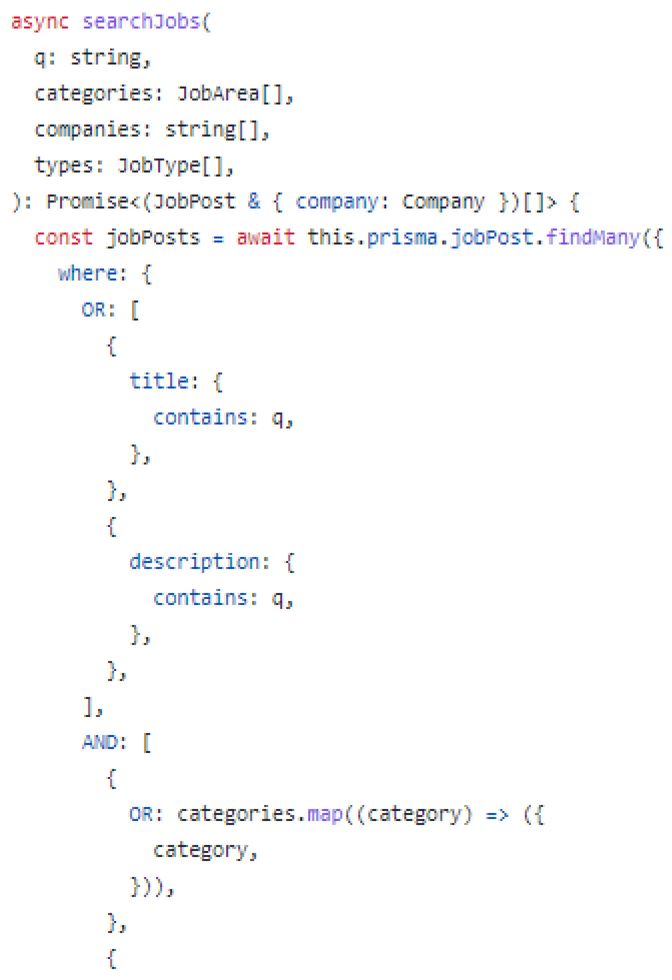
Our Specific Code Style:

- At start of development phase, there will be a starting point of code that shows how the code style will look like
- This will allows members to use the starting point as a reference to how they should be coding
- This will allow team members from both back-end and front-end to know before actually developing what coding style the team will use.
- For any frameworks, such as Nest.js, we use the provided formatting that the framework gives us
- We don't have a specific coding style, our main focus is simply sticking to one custom coding style and staying consistent with it throughout the development phase.

Back-End Formatting:

- Coding Style used for all the back-end pages that we have
- This is the Search Functions that was implemented and designed by the Back-End Team

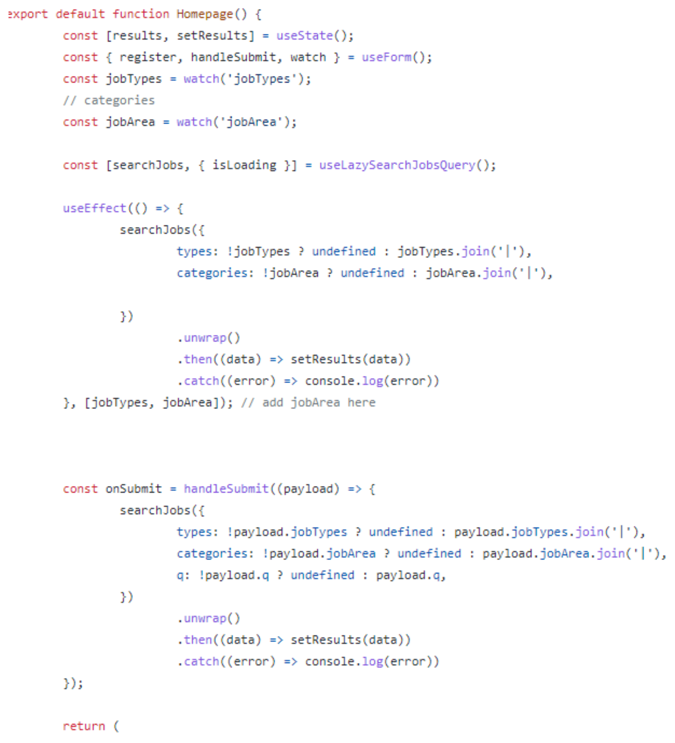
```
@Controller('api/search')
export class SearchController {
  constructor(private readonly searchService: SearchService) {}

  // EXAMPLE: /api/search?q=terms&categories=UI_UX|AR_VR&companies=Glowme&types=INTERSHIP
  @Get('jobs')
  async searchJobPosts(
    @Query() { q, categories, companies, types }: SearchJobsParams,
  ): Promise<(JobPost & { company: Company })[]> {
    categories = ((categories ?? []) as string[]).filter(
      (f) => f !== '',
    ) as JobArea[];
    types = ((types ?? []) as string[]).filter((f) => f !== '') as JobType[];

    return this.searchService.searchJobs(
      q ?? '',
      categories ?? [],
      companies ?? [],
      types ?? [],
    );
  }
}
```



```
async searchJobs(
  q: string,
  categories: JobArea[],
  companies: string[],
  types: JobType[],
): Promise<(JobPost & { company: Company })[]> {
  const jobPosts = await this.prisma.jobPost.findMany({
    where: {
      OR: [
        {
          title: {
            contains: q,
          },
        },
        {
          description: {
            contains: q,
          },
        },
      ],
      AND: [
        {
          OR: categories.map((category) => ({
            category,
          })),
        },
        {
```

Front-End Formatting:

- Coding Style used for all the front-end pages that we have
- This is the Search Functions that was implemented and designed by the Front-End Team



```
export default function Homepage() {
        const [results, setResults] = useState();
        const { register, handleSubmit, watch } = useForm();
        const jobTypes = watch('jobTypes');
        // categories
        const jobArea = watch('jobArea');

        const [searchJobs, { isLoading }] = useLazySearchJobsQuery();

        useEffect(() => {
                searchJobs({
                        types: !jobTypes ? undefined : jobTypes.join('|'),
                        categories: !jobArea ? undefined : jobArea.join('|'),

                })
                        .unwrap()
                        .then((data) => setResults(data))
                        .catch((error) => console.log(error))
        }, [jobTypes, jobArea]); // add jobArea here



        const onSubmit = handleSubmit((payload) => {
                searchJobs({
                        types: !payload.jobTypes ? undefined : payload.jobTypes.join('|'),
                        categories: !payload.jobArea ? undefined : payload.jobArea.join('|'),
                        q: !payload.q ? undefined : payload.q,
                })
                        .unwrap()
                        .then((data) => setResults(data))
                        .catch((error) => console.log(error))
        });

        return (
```

Feedback:

- This is done through the approvals and review of any and all commits send to the development branch
- A team member will commit to the development branch with a review request done that is reviewed by specific team members that were working on that issue.
- All commits will also be sent to the team lead to check progress done.

5. Security

Authentication:

- Students/Company/Admin profiles are protected through the use of JWT Tokens
    - These JWT Tokens are used from the JSON Web Tokens open source standard
    - They allows us to use guards in order to protect our data when we create or update different profiles
    - They are signed and verified, which are attached to guards that then allows our implementations to grab and therefore use within the website.
    - Passwords are hashed through the use of BCrypt for further protection
    - Passwords are excluded from guards, therefore implementations and functions can not use the user's password

Register:

- Input data Validation - Blowfish Algorithm
    - We make sure every field is filled and the passwords minimum length is at least 8 characters
    - We send through JWT payloads which send the information from the user input to a validation function that checks against the key and UnauthorizedException() to see if it can be validated.

Text Fields:

- All of the text fields for registering and updating profiles are protected through the validation sequences.
    - By protecting the text fields with the use of JWT Payloads and JWT Tokens, input is not taken if it does not match the corresponding key and will spit on unauthorizedException() error.

  - By giving out this error, we eliminate the possible attack of SQL Injection or side-scripting as no other inputs are allowed.
- The only other text fields found in our websites is the search bar itself
  - This is protected by the specific algorithm that we use.
  - The main way it is protected is by we created a specific algorithm that only takes in information and input that searches
  - No other data input is allowed to affect any other part of the website
  - We did this in order to limit what capabilities the search bar had on our website.

3. Self-Check: Adherence to original Non-Function Specs (Done by Thomas - Team Lead)

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by class CTO): DONE

2. Application shall be optimized for standard desktop/laptop browsers e.g., must render correctly on the two latest versions of two major browsers: DONE

3. Selected application functions must render well on mobile devices: DONE

4. Data shall be stored in the team's chosen database technology on the team's deployment server: DONE

5. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users : DONE

6. The language used shall be English: DONE

7. Application shall be very easy to use and intuitive: DONE 8. Google maps and analytics shall be added: DONE

9. No email clients shall be allowed. You shall use webmail: DONE 10. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI: DONE

11.Site security: basic best practices shall be applied (as covered in the class): DONE

12. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development: DONE **13.** The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Fall 2022. For Demonstration Only" at the top of the WWW page. (Important so not to confuse this with a real application). : DONE