hatchio 

[April 23, 2021]

SW Engineering CSC648/848 Spring 2021

Section 02 | Team 03

Milestone 04

Team Information

POSITION	NAME	EMAIL
Team Lead / Github Master	Roland Lee	mlee38@mail.sfsu.edu
Front End Lead	Jose Gonzalez	jgonzalez34@mail.sfsu.edu
Back End Lead	Lyra Solomon	lsolomon3@mail.sfsu.edu
Database Manager	Aaron Singh	asingh26@mail.sfsu.edu

History Table

Milestones	Date Submitted	Date Revised
01	2/23/2021	3/2/2021
02	3/9/2021	3/23/2021
04	4/23/2021	

Table of Contents

1) Product summary

Brief Description: *Hatchio* is a website that provides functionality for three unique users including: students, professors, and employers. *Hatchio* provides an interface for these three types of users to interact. The following are lists of functions that explain the interactivity between the three users further.

URL Product: http://3.141.216.125:3000

Unique about our product: Some of the things that make our product unique include the following: unique dashboards conditionally rendered based on the user type, very apt in response time, and clean interface.

Name of the Product: Hatchio

- Employers shall be able to filter through student profiles based on a criteria that can select talented students such as their gpa, major, and ratings based on professors.
- Employers shall be able to notify the student that they are interested in employing the student.
- Employer's dashboard will have the ability to post jobs on the market place. The job description will include things like the position, salary, benefits, and type of work.
- Students shall be able to filter through jobs based on a limited set of criteria such as Job and Position type for simplicity.
- Students shall be able to notify the Employer that he or she is interested in getting hired for the job; the employer will get notified on his dashboard.
- Students will have the ability to add as many projects,education or experience they would like to add on a public student profile; only students with education,and a profile page will be listed on the public view of the student profiles search.
- Students will get notified when they are rated by a professor.
- Professors, exclusively, will have the ability to rate the students based on a general criteria and have the ability to enter a recommendation.

2) Usability test plan

Test Objectives

_____The usability test plan is designed to test search for talent function for the web application, hatchio. The goals of usability testing include establishing a baseline of user performance, establishing and validating user experience with the current application's UI design (which is illustrated in Milestone 02), and identifying potential design concerns to be addressed in order to improve the efficiency, productivity, and end-user satisfaction. In addition, the usability test serves as an excellent debugging tool by determining design inconsistencies and usability problem areas within the user interface and content areas. Potential sources of error/threat may include:

- Navigation errors – failure to locate functions, excessive keystrokes to complete a function, failure to follow recommended screen flow.
- Presentation errors – failure to locate and properly act upon desired information in screens, selection errors due to labeling ambiguities.
- Control usage problems – improper toolbar or entry field usage.
- SQL security breach – SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution.
- Broken access control – Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data.
- Security Misconfiguration – Result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.

The main purpose of usability testing is to exercise the application under a controlled test environment with representative users. Data flow will be assessed in regards to effectiveness, performance, and efficiency.

Test Background and Setup

Hatchio application is built upon AWS EC2 t2.micro 1 vCPU 1GB RAM Ubuntu 20.04 with MySQL 8.0 database. By default, the database is loaded with ten student users with their major, GPA, rating total, student year, top strengths, and self-written bio. Possible roles involved in usability test are as follows. A tester may play multiple roles to identify whether the application's functions meet the intended user's expectation.

1. **Employer**
    a. A Business Recruitment employer who is specialized in finding the right candidates for the company. The expected result from searching for talent is recent undergraduate/graduate students based on GPA, top strengths (programming languages that the person is good at), major, and school year.
2. **Student**

   a. A recent undergraduate/graduate student at San Francisco State will expect to find his/her classmate's rating, top strengths, school year, major, and self-written bio to better understand his/her peers.

3. **Professor**
   a. A professor or teacher assistant(TA) who works at San Francisco State University will expect to find the student's current GPA (future candidates for side projects and researches), self-written bio (be able to understand the student better), and top strengths by accessing searching for talent function.

The usability test plan is designed to measure numbers of scenario completion, critical error, and required actions to complete each task.

- URL of the tested system: http://3.141.216.125:3000

Usability Task Description

Before the test

Before the usability test, all testers must acknowledge there are a total of ten student users that can be displayed based on different filters (Access :5000/ec2 to have a better understanding of the database). Besides, make sure the tester keeps track of the number of scenario completion, errors, and actions(including page redirect) needed to complete each task.

As for usability effective metrics, a tester needs to measure the percentage of test/use cases complemented while documenting the number of errors encountered during the test. Optionally, the tester can comment on specific features of searching for talent functionality, which serves as valuable feedback for the dev team to improve and customize the functionality according.

As for usability efficient metrics, a tester needs to measure the number of clicks that are required for each task and provides the average number of clicks based on the formula [total number of clicks/number of tasks completed]. Besides, the tester can put down any comment that can potentially improve the function's performance.

Lickert Subjective Test

I'm satisfied with how searching for talent displays the results.
Strongly disagree_ Disagree_ Neutral_ Agree_ Strongly agree_

The filters and search bar are readable and easy to use.
Strongly disagree_ Disagree_ Neutral_ Agree_ Strongly agree_

Overall, I'm satisfied with the results of searching for talent functionality.
Strongly disagree_ Disagree_ Neutral_ Agree_ Strongly agree_

- Lickert Test URL: https://forms.gle/wwFJNhqASNLYFbSP9

3) QA test plan

Test objectives

Some of test objectives are as follows but not limited to:

- Ensure the application meets functional and non-functional requirements under test environment
- Bugs/issues identification and fixed before going live.

HW and SW setups

_____Hatchio application is built upon AWS EC2 t2.micro 1 vCPU 1GB RAM Ubuntu 20.04 with MySQL 8.0 database. By default, the database is loaded with ten student users with their major, GPA, rating total, student year, top strengths, and self-written bio.

Features to be tested

1. User registration
2. Email verification
3. User authentication
4. Search for talent
5. Employers can post Jobs if they are signed in
6. Professors can rate Students if they are signed in
7. First time Students can insert the following: profile page, education, projects

QA Test plan

Google Chrome V89.0.4389.128 / Firefox v88.0

Test #	Test Title	Test Description	Test Input	Expected Output	Test Results
1	User Registration	Create a student user with existing email address	(select student icon), enter “test” for fname,lname,and schoolname. Enter“test1234” for passoword and enter an existing email address	Go to :5000/students and verify the account has been created.	PASS/PASS
2	Email Verification	Activate a new account by checking email inbox or spam.	access the email’s spam that you entered in Test #1 and complete email verification process which is provided by the application bot	Go to :5000/students and verify state variable has been set to “1” for the account you created in Test #1	PASS/PASS

3	User Authentication	Basic password and usertype check.	(select student from drop down list), enter elonmusk@mail.com and pass12345 for email and password	Redirect to Tom Bobby's student profile page.	PASS/PASS
4	Search for Talent	Display candidates based on filter option(s).	(select "Freshman" under Student Year), enter 3.2 and 4.0 under GPA and click on the "+" sign to add filter.	Display two students: Tom Bobby and Sunny Jar with 4 and 3.5 GPA, correspondingly.	PASS/PASS
5	Post Jobs	A employer can log in and will be able to access the tab "Post Jobs".	Log in as Square Space employer (see :5000/employers for user info), create a new job posting on Post Jobs page.	Under search jobs, you will see under jobs listing the inserted job. Please note, view and apply functions have not been implemented yet.	PASS/PASS
6	Rate Students	A professor who is logged in can go under student/search and be able to rate a student and upon viewing the profile, will see his/her rating.	Log in as Duc Ta professor and rate Tom Bobby on student search page.	Under student Tom Bobby profile, you will see Duc Ta's rating inserted.	PASS/PASS

7	Student Profile	A first time user who has signed up, will have forms on the dashboard available to put information in their profile and will have the corresponding information rendered.	Login as the new user that you created in Test#1. On dashboard page, enter desired location, self-intro, top qualities, and school year by clicking on profile page insert button under About Me, click on the "+" sign next to Projects and create a project that you want to attach to student profile.	Update student's About Me, Top Qualities, Location, and Projects based on test inputs.	FAIL/FAIL (Professo r under created project isn't displayed correctly)

4) Code Review:

A) **Coding Style**: Node.js Style Guide by Felix Geisendorfer licensed under the CC BY-SA 3.0. In order to enforce consistency, a configuration file was used in IDE. (code length has been adjusted from 80 characters per line to 120)

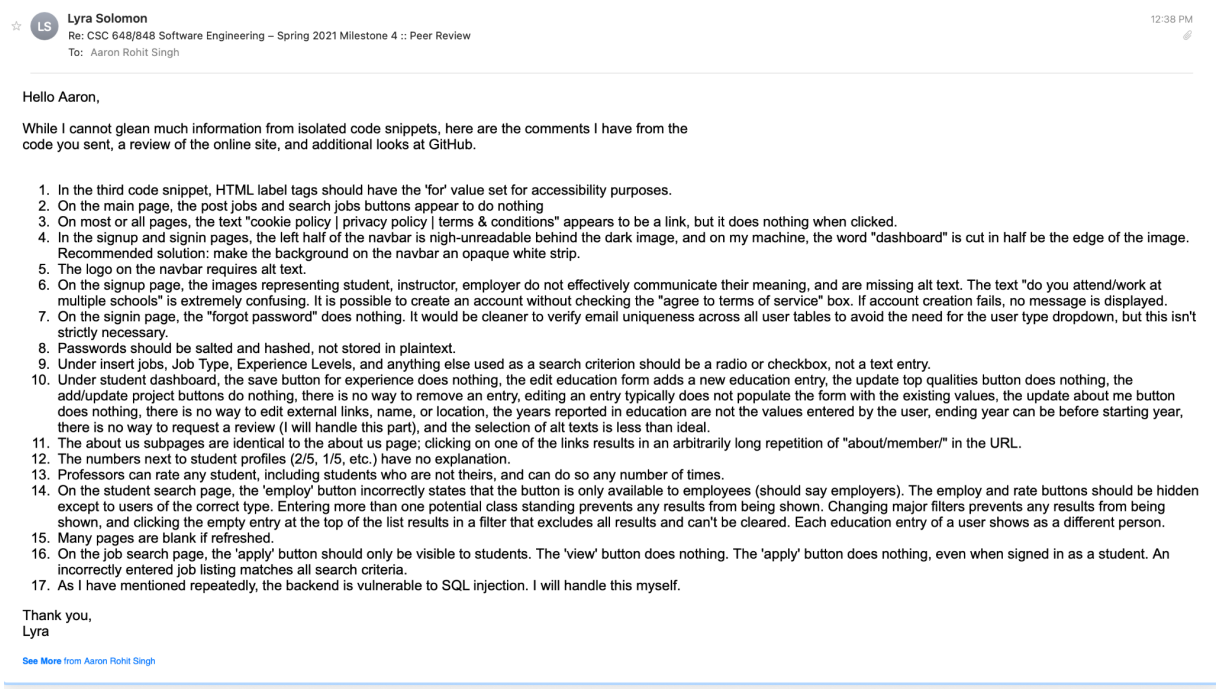
B) Usability testing includes the functionality and delivery of performance of the application. The following are critical functions in delivering the application including: filtering for search for talent, dashboard layouts for unique users (students, professors, employers) and registering users.

User Dashboard: This is our Dashboard component. Upon rendering after a user signs in, we access the locally saved cookie that was sent by the backend server to retrieve the user profile that is currently authenticated. The view layer is dynamically changed depending on the type of user and populates the corresponding data.

Registering: Students can have the ability to insert as many projects and educations. They are limited to one form for initially setting up there profile page.

Filter On load, the component will retrieve the response from the backend for the data to load. After loading, the filters on students are pure javascript that handles toggle what to display based on the filters. The Job filters are used with the help of the GET request to send back filtered results from mysql. Along with that are features including "Employ"(yet to be implemented) , "Rate[Student]" , and "Profile" to view the full profile of a student.

Below is a screenshot of the peer review and the screenshots provided. The peer review, Lyra, takes not only the functionalities used for testing, but also the overall progress of the entire website.



Lyra Solomon 12:38 PM
Re: CSC 648/848 Software Engineering – Spring 2021 Milestone 4 :: Peer Review
To: Aaron Rohit Singh

Hello Aaron,

While I cannot glean much information from isolated code snippets, here are the comments I have from the code you sent, a review of the online site, and additional looks at GitHub.

1. In the third code snippet, HTML label tags should have the 'for' value set for accessibility purposes.
2. On the main page, the post jobs and search jobs buttons appear to do nothing
3. On most or all pages, the text "cookie policy | privacy policy | terms & conditions" appears to be a link, but it does nothing when clicked.
4. In the signup and signin pages, the left half of the navbar is nigh-unreadable behind the dark image, and on my machine, the word "dashboard" is cut in half be the edge of the image. Recommended solution: make the background on the navbar an opaque white strip.
5. The logo on the navbar requires alt text.
6. On the signup page, the images representing student, instructor, employer do not effectively communicate their meaning, and are missing alt text. The text "do you attend/work at multiple schools" is extremely confusing. It is possible to create an account without checking the "agree to terms of service" box. If account creation fails, no message is displayed.
7. On the signin page, the "forgot password" does nothing. It would be cleaner to verify email uniqueness across all user tables to avoid the need for the user type dropdown, but this isn't strictly necessary.
8. Passwords should be salted and hashed, not stored in plaintext.
9. Under insert jobs, Job Type, Experience Levels, and anything else used as a search criterion should be a radio or checkbox, not a text entry.
10. Under student dashboard, the save button for experience does nothing, the edit education form adds a new education entry, the update top qualities button does nothing, the add/update project buttons do nothing, there is no way to remove an entry, editing an entry typically does not populate the form with the existing values, the update about me button does nothing, there is no way to edit external links, name, or location, the years reported in education are not the values entered by the user, ending year can be before starting year, there is no way to request a review (I will handle this part), and the selection of alt texts is less than ideal.
11. The about us subpages are identical to the about us page; clicking on one of the links results in an arbitrarily long repetition of "about/member/" in the URL.
12. The numbers next to student profiles (2/5, 1/5, etc.) have no explanation.
13. Professors can rate any student, including students who are not theirs, and can do so any number of times.
14. On the student search page, the 'employ' button incorrectly states that the button is only available to employees (should say employers). The employ and rate buttons should be hidden except to users of the correct type. Entering more than one potential class standing prevents any results from being shown. Changing major filters prevents any results from being shown, and clicking the empty entry at the top of the list results in a filter that excludes all results and can't be cleared. Each education entry of a user shows as a different person.
15. Many pages are blank if refreshed.
16. On the job search page, the 'apply' button should only be visible to students. The 'view' button does nothing. The 'apply' button does nothing, even when signed in as a student. An incorrectly entered job listing matches all search criteria.
17. As I have mentioned repeatedly, the backend is vulnerable to SQL injection. I will handle this myself.

Thank you,
Lyra

See More from Aaron Rohit Singh

StudentSearch.js

frontend > src > views > js > search > student > StudentSearch.js > StudentSearch

```
]); // holds filtered students (avoids having to fetch from DB again)
const [activeFilters, setActiveFilters] = useState([]); // Active filters
const [schoolYear, setSchoolYear] = useState([]); // Active school year(s)
const [strength, setStrength] = useState("");
const [gpa, setGPA] = useState({ min: 0, max: 4 });
const [rating, setRating] = useState({ min: 0, max: 5 });
const [keyword, setKeyword] = useState(""); // Search Bar Keyword storing

// fetch students from DB && initialize the results
useEffect(() => {
  API_FETCH_STUDENTS(setdbStudents);
  API_FETCH_STUDENTS(setResults);
}, []);

//filter dbStudents based on the keyword entered
const keywordFilterHandler = (e) => {
  e.preventDefault(); // prevent refresh
  setResults(dbStudents); // resets results to have the entire students DB

  // filters results based on the input (ONLY if input is not blank)
  if (keyword !== "") {
    setResults(
      results.filter((student) => {
        return (
          student.first_name.toLowerCase().indexOf(keyword.toLowerCase()) !==
            -1 ||
          student.last_name.toLowerCase().indexOf(keyword.toLowerCase()) !==
            -1 ||
          student.study_major.toLowerCase().indexOf(keyword.toLowerCase()) !==
            -1 ||
          student.school.toLowerCase().indexOf(keyword.toLowerCase()) !==
            -1 ||
          student.school_grade_level
            .toLowerCase()
            .indexOf(keyword.toLowerCase()) !== -1
        );
      })
    );
  }
};

// Updates school year(s) list
const addSchoolYearHandler = (e) => {
  // School Year FILTER
  if (e.target.checked) {
    setSchoolYear((activeFilters) => [...schoolYear, e.target.value]);
    // Updates results (adds school year)
    setResults(
      results.filter((student) => {
        return (
          student.school_grade_level
            .toLowerCase()
            .indexOf(e.target.value.toLowerCase()) !== -1
        );
      })
    );
  } else {
    setSchoolYear(schoolYear.filter((filter) => filter !== e.target.value));
    // Updates results (removes school year)
    setResults(dbStudents);
  }
};

// Updates active filters
const addFilterHandler = (e) => {
```

milestone03 ⊗ 0 ⚠ 0 -- NORMAL --

```
156  //POP UP Student Projects
157  const POPUP_STUDENT_PROJECTS = (Student_ID) => {
158    return (
159      <div>
160        <StyledPopup trigger={<button> Insert Projects</button>}>
161          <Formik
162            initialValues={{
163              Student_ID: Student_ID,
164              project_name: "Project Name",
165              summary: "Summary",
166              professor: "professors name",
167              arr_tools_used: "c++,java,etcs",
168              links_website: "links_webiste",
169              arr_collaborators_arr: "collarborates",
170            }}
171            onSubmit={async (values) => {
172              console.log(values);
173              const response = await API_STUDENT_INSERT_PROJECTS(values);
174              console.log(response);
175              if (response == 400) {
176                console.log("error");
177              }
178              if (response == 200) {
179                window.location.reload();
180                console.log("success");
181              }
182              return;
183            }}
184          >
185            <Form>
186              <label> Projecct Name</label>
187              <Field id="project_name" name="project_name" />
188              <label> Summary</label>
189              <Field id="summary" name="summary" />
190              <label> Professors</label>
191              <Field id="professor" name="professor" />
192              <label> Array of Tools</label>
193              <Field id="arr_tools_used" name="arr_tools_used" />
194              <label> Links & Websites</label>
195              <Field id="links_website" name="links_website" />
196
197              <button type="submit">Submit</button>
198            </Form>
199          </Formik>
200        </StyledPopup>
201      </div>
202    );
203  };
```

Dashboard.js ×

rontend > src > views > js > user > Dashboard.js > Dashboard

You, 2 days ago | 2 authors (You and others)

```
/**
 * File: Dashboard.js
 * Purpose: Serve Unique Dashboard for varying users
 * Functionality IE: Render dynamically based on logged in User.
 * Authors:
 * Aaron implementing Cookies & Condition
 * Jose: Profile Components for Student | Professor | Employer
 */
import { useState, useEffect } from "react";  2.9K (gzipped: 1.3K)
import { useCookies } from "react-cookie";  4.6K (gzipped: 1.8K)
// CSS
import "../../css/Help.css";
// Profile Components
import StudentProfile from "../profiles/student/StudentProfile";
import ProfessorProfile from "../profiles/professor/ProfessorProfile";
import CompanyProfile from "../profiles/company/CompanyProfile";
//API
import API_USER_GET_PROFILE from "../../../models/user_profile";
const Dashboard = () => {
  //Get Profile from Database      You, 2 days ago • code_credentials&doing_m4_c
  useEffect(() => {
    API_USER_GET_PROFILE(cookie.Type_User, cookie.ID_OF_USER, setuserProfile);
  }, []);
  //Current User
  const [cookie] = useCookies(["Type_User", "ID_OF_USER", "First_Name"]);
  //User Profile Return
  const [userProfile, setuserProfile] = useState([
    [{ null: "null" }, { null: "null" }], //General Information
    [{ null: "null" }, { null: "null" }], //Education
    [{ null: "null" }, { null: "null" }], //Ratings
    [{ null: "null" }, { null: "null" }], //Job Listings
    [{ null: "null" }, { null: "null" }],
  ]);

  if (cookie.Type_User === "student") {
    return (
      <>
        <StudentProfile {...userProfile} />
      </>
    );
  } else if (cookie.Type_User === "professor") {
    return (
      <>
        <ProfessorProfile {...userProfile} />
      </>
    );
  } else if (cookie.Type_User === "employer") {
    return (
      <>
        <CompanyProfile {...userProfile} />
```

THIS IS WHAT WE HAVE RIGHT NOW

-Populated Database with 10 Students(Student Profile Page, Education, Projects, Ratings), 4 Professors, 4 Employers, and 1 Admin.

-Conditional User Dashboard and page rendering w/ proper information fetch

-Student Registration Forms on dashboard.

-Insert Jobs for Employers Only

-Rate students for Professors Only

THIS IS WHAT WE SORT OF HAVE RIGHT NOW

-Student filters page should let the user know when to hit the search bar to refresh when it gets buggy

THIS IS WHAT WE WOULD IDEALLY LIKE HAVE SOON

-Sign up forms forms Professor | Employer

-Notifications for all students and employers , (Student: get notified when they are rated,Employer: Get notified when student clicks on ‘employ’).

5) Self-check on best practices for security

List of major assets we're protecting

- User credentials with encryption for sending the network request from the client to the backend; storing encrypted password in the database to prevent insiders from viewing.
- Preloaded users in the database for demo testing are not going to be encrypted and will have an exception case in the backend to allow insecure access for users but all new users will be secure.
- Overlapping accessibility of users. For example, employers should be able to access post jobs but students should not.
- Post requests for student dashboard profile insertion are only processed if the input fields are correctly formatted.
- Sign in fields must both be complete in order for the verification to be processed.
- Field inputs for any strings are correctly matched with the length of the storage allocated on the database column attribute.

6) Self-check: Adherence to original Non-functional specs – performed by team leads

1. The 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. The application shall be optimized for standard desktop/laptop browsers, e.g., it must render correctly on the two latest versions of two major browsers. **ON TRACK**
3. Selected application functions must render well on mobile devices. **ISSUE, our frontend team is still working on it, but not expecting to be fully implemented when final delivery is due.**
4. Data shall be stored in the team's chosen database technology on the team's deployment server. **DONE**
5. No more than 100 concurrent users shall be accessing the application at any time **NO ACTION REQUIRED**
6. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. **ON TRACK, as for now, new user's password is protected through password hashing. (but that's all we have for privacy features)**
7. The language used shall be English. **IN COMPLIANCE; BUT DEVELOPMENT ONGOING. NO ISSUES EXPECTED.**
8. The application shall be very easy to use and intuitive. **ON TRACK**
9. Google maps and analytics shall be added. **ISSUE, dropping this feature with professor's consent.**
10. No email clients shall be allowed. You shall use webmail. **DONE**
11. Pay functionality, if any (e.g., paying for goods and services) shall not be implemented nor simulated in UI. **NO ACTION REQUIRED**
12. Site security: basic best practices shall be applied (as covered in the class) **IN PROGRESS**
13. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development. **ON TRACK**
14. The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Spring 2021. For Demonstration Only" at the top of the WWW page. (Important so as not to confuse this with a real application). **ON TRACK**
15. The project shall comply with WCAG 2.1 **IN PROGRESS**
16. The website shall respond quickly to user input and shall not cause lag in any portion of the user's system. If the client must wait for the server's response, this shall be communicated to the user so they do not suspect a bug. **IN COMPLIANCE; BUT DEVELOPMENT ONGOING**