SW Engineering CSC648/848 Summer 2019
Safe SF
Team 2
Milestone 4
August 4, 2019

Team: Local

Alex Wolski
Team Lead & Back End
awolski@mail.sfsu.edu

David Stillwagon (Ryan)
Back End Lead

Lidiya Jebessa
Back End

Sunny Srijan
Front End Lead

Evan Guan
Front End

Tristan Mclennan
Front End

Harshveer Saini (Harsh)
GitHub Master & Front End/Back End

Reversion Table:

Version	Date	Description
1.0	August 6	Submitted For Review
1.1	August 13	Revised based off of feedback

Product Summary

Title: SF Safe

Safe SF is a website that helps raise awareness of environmental hazards in San Francisco. Users can check for hazards in an area to help them make safer decisions. And the website informs city management on where hazards are so that they can be removed quickly and efficiently. The goal of **Safe SF** is to create a safer, cleaner, and more environmentally friendly San Francisco.

- Users will be able to search for reports in a search bar
- Users will be able to filter through their results using dropdown menus
- Clicking on one of the results will show more details about that report
- Users will be able to create new reports with details such as location, a description, and an image
- The website will display properly on the last two versions of three different browsers as well as on mobile

SF Safe differentiates itself from other similar environmental hazard reporting websites because it has filtering options to help users find reports in a specific area faster and easier. In addition, SF Safe has a map feature that lets users see where hazards are being reported and how many there are.

URL to the website:
http://ec2-34-220-99-220.us-west-2.compute.amazonaws.com/

Usability Test Plan

Test objective:

The final objective of the following test plan is to measure the effectiveness and efficiency of the proposed system developed for reporting environmental hazards and to record subjective feedback from the sample of people who are invited to use the system and use this feedback to understand what worked and what needs fixing, and what completely failed from the development point of view.

Test background:

The system is designed to work on leading browsers viz. Google chrome, Mozilla Firefox and Safari on Mac, the user would need an active internet connection to connect and interact with the system available at http://safesf.ddns.net:3000/

Although attempts have been made to make the design of the website to be intuitive, 'English' is the descriptive language used on the website and there are no localized versions available yet.

The Website has been developed to allow people of SF report environmental Hazards with ease if they come across one, and keeping in mind the diverse population in the Metropolitan. It would make sense if the audience for the Usability tests would be as diverse, from various ethnic backgrounds, and from age groups as diverse as possible but not younger than 12 year old.

Usability Tasks:

Visit http://safesf.ddns.net:3000/

- Try to search for already reported Hazards.

- View the detailed report of a reported Hazard

- Try to report a hazard with dummy data

Lickert Questionnaire:

Scale: (1) Strongly Disagree . . . (5) Strongly Agree, followed by comment boxes.

- It is easy to find reports that are relevant to me.

1 2 3 4 5

Strongly Disagree **Strongly Agree**

[Do you have any comments?]

- Detailed report information is easy to find.

1 2 3 4 5

[Do you have any comments?]

- The type of detailed report information is useful.

1 2 3 4 5

[Do you have any comments?]

- It is easy to report a hazard on the website.

1 2 3 4 5

[Do you have any comments?]

- The website is easy to use overall.

1 2 3 4 5

[Do you have any comments?]

QA Test Plan

Test Objective: The final objective of the following test plan is to verify that the developed system works as expected and adheres to all requirements.

Test Background: The system is developed to render on Mobile as well as desktop web, however for the purpose of this test pass we would only be validating it is functional on desktop web only.

Make sure one of the following browsers is installed on the system being used to test the system: Google Chrome, Mozilla Firefox, Safari on Mac.

Application URL: http://safesf.ddns.net:3000/

Test Plan

	Feature	Instruction	Expected Output	Actual Output	Result	
					Google Chrome Ver-75.0.3770.142	**Mozilla Firefox Ver- 68**
1	**Homepage**	Go to http://safesf.ddns.net:3000/	User should see the home page with the following components: - Map-View -Navbar (Login/Signup, Search-bar)	All page component were seen	PASS	PASS
2	**Search**	-Try to click on the search button with all the search criteria blank	User should be shown 10 most recent reports in the list card and their locations marked by pins on the map	14 reports are shown, and the pins are not marked on the map	FAIL	FAIL
3		Try to search by only category -Garbage	User should be returned 5 reports of only that category ordered by date (most recent first)	5 reports with the garbage category in order by most recent	PASS	PASS
4		Try to search by location (Balboa Park)	User should be returned with 6 reports from only the entered location (recent first)	The user is returned with the expected reports	PASS	PASS

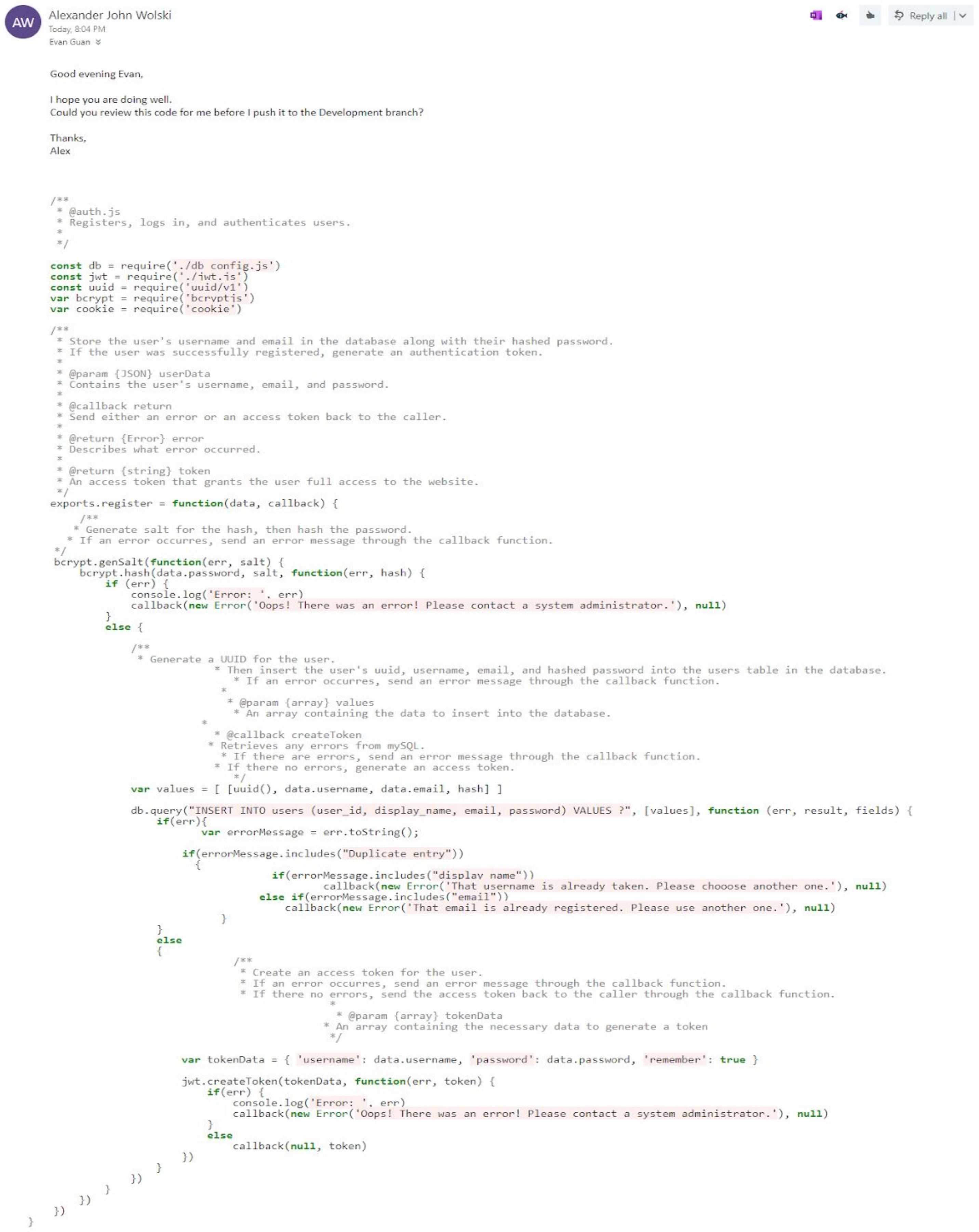
5	**Submit Reports Page**	Report and submit an issue including a short description without logging in first.	The user should first be directed to log in before their report can be submitted. Once logged in the user should see a web page that displays the information they entered into the form.	Report is submitted without logging in.	FAIL	FAIL
6		Submit a report without entering the mandatory fields	User should not be able to submit the report without filling out the mandatory fields	Verified that the user is not able to submit a report without filling the mandatory fields	PASS	PASS
7		Report and submit an issue after logging in first	The user should be redirected to a web page that displays the information they entered into the form.	Verified that the user was able to submit the report successfully and was redirected to the reports detail page	PASS	PASS
8	**Reports Detail Page**	Search for an existing report and click on its card	Verify that the user is redirected to the reports detail page of the report he clicked on	Verified that the user is redirected to the report detail page	PASS	PASS

Code Review

For Javascript syntax styling (on the frontend and backend), we are using the official javascript coding standards:
https://www.drupal.org/docs/develop/standards/javascript/javascript-api-documentation-and-comment-standards

As for comments, we are using the JavaScript commenting standards documentation:
https://www.drupal.org/docs/develop/standards/javascript/javascript-api-documentation-and-comment-standards

The email chain depicting one of our code reviews is shown on the next few pages.



Alexander John Wolski
Today, 8:04 PM
Evan Guan

Good evening Evan,

I hope you are doing well.
Could you review this code for me before I push it to the Development branch?

Thanks,
Alex

```
/**
 * @auth.js
 * Registers, logs in, and authenticates users.
 *
 */

const db = require('./db_config.js')
const jwt = require('./jwt.js')
const uuid = require('uuid/v1')
var bcrypt = require('bcryptjs')
var cookie = require('cookie')

/**
 * Store the user's username and email in the database along with their hashed password.
 * If the user was successfully registered, generate an authentication token.
 *
 * @param {JSON} userData
 * Contains the user's username, email, and password.
 *
 * @callback return
 * Send either an error or an access token back to the caller.
 *
 * @return {Error} error
 * Describes what error occurred.
 *
 * @return {string} token
 * An access token that grants the user full access to the website.
 */
exports.register = function(data, callback) {
     /**
    * Generate salt for the hash, then hash the password.
   * If an error occurres, send an error message through the callback function.
 */
 bcrypt.genSalt(function(err, salt) {
     bcrypt.hash(data.password, salt, function(err, hash) {
         if (err) {
             console.log('Error: ', err)
             callback(new Error('Oops! There was an error! Please contact a system administrator.'), null)
         }
         else {

             /**
              * Generate a UUID for the user.
                          * Then insert the user's uuid, username, email, and hashed password into the users table in the database.
                             * If an error occurres, send an error message through the callback function.
                           *
                            * @param {array} values
                             * An array containing the data to insert into the database.
                        *
                          * @callback createToken
                         * Retrieves any errors from mySQL.
                           * If there are errors, send an error message through the callback function.
                          * If there no errors, generate an access token.
                             */
             var values = [ [uuid(), data.username, data.email, hash] ]

             db.query("INSERT INTO users (user_id, display_name, email, password) VALUES ?", [values], function (err, result, fields) {
                 if(err){
                        var errorMessage = err.toString();

                     if(errorMessage.includes("Duplicate entry"))
                       {
                                   if(errorMessage.includes("display_name"))
                                           callback(new Error('That username is already taken. Please chooose another one.'), null)
                                 else if(errorMessage.includes("email"))
                                     callback(new Error('That email is already registered. Please use another one.'), null)
                           }
                 }
                 else
                 {
                             /**
                              * Create an access token for the user.
                              * If an error occurres, send an error message through the callback function.
                              * If there no errors, send the access token back to the caller through the callback function.
                                            *
                                             * @param {array} tokenData
                                           * An array containing the necessary data to generate a token
                                            */

                     var tokenData = { 'username': data.username, 'password': data.password, 'remember': true }

                     jwt.createToken(tokenData, function(err, token) {
                         if(err) {
                             console.log('Error: ', err)
                             callback(new Error('Oops! There was an error! Please contact a system administrator.'), null)
                         }
                         else
                             callback(null, token)
                     })
                 }
             })
         }
     })
 })
}
```

EG Evan Guan
Today, 8:26 PM

I do not see any glaring issues with this code. I would say go ahead with the push.

AW Alexander John Wolski
Today, 8:27 PM

Great! Thank you.

Self-check on best practices for security

Protected Data:

The major assets that we are protecting are secured in the database. This information is not accessible to the public since it is protected with a password, and this password is not stored on the front end. These assets include:

- **User data**

  The email, username, and password of registered users are kept private

- **Report, Category, and Location data**

  Information regarding reports, categories, and locations are sent to the front end to be displayed. To ensure that this information can't be replaced with inappropriate words, they are secured and inaccessible to the public.

- **Report Images**

  Malicious users may attempt to replace these images with inappropriate ones. So to ensure this doesn't happen, they are stored on our web server, which is secured with a password.

- **Passwords**

  In the case of a security breach, we ensure that the passwords of registered users aren't leaked by encrypting them. We are using the bcrypt library to generate a salted hash and hash the user's password. The hashed password is then stored in the database.

- **Input validation**

  And although we haven't implemented this yet, we will protect against SQL injections and server crashes by validating the search terms users enter. The search will be limited to 50 alphanumeric characters. To ensure that users don't bypass the frontend and send invalid search terms directly to the backend, we validate the search term on both the frontend and backend.

Self-check: Adherence to original Non-functional specs

List of non-functional specs:

- The website shall require registration and login to report hazards on the site.
  DONE
- Captcha authentication shall be used on the registration page.
  ON TRACK
- Management tools shall be used to manage the team.
  DONE
- Each page shall have official company logo in the upper left corner.
  ON TRACK
- The website shall use one concept per requirement
  ON TRACK
- The website shall use number for tracking
  DONE
- The website shall group by priorities
  ON TRACK