SW Engineering CSC648/848 Spring 2023
SafetyHub

Section: 02
Team: 03
Diego FLORES - Team Lead - Database - dfloresflores@mail.sfsu.edu
Carmelo DE GUZMÁN - Front End Lead - cdeguzman@mail.sfsu.edu
Coline SEGURET - Github Master - cseguret@sfsu.edu
Sumith SHRESTHA - Front End - Sshrestha12@sfsu.edu
Adrian VAZQUEZ - Back End Lead - Database - avazquez10@mail.sfsu.edu
Isabel FALCON - Back End - ifalcon@mail.sfsu.edu

Milestone 4
04/28/2023

Revision History Table

Description	Date Submitted	Date Revised
Initial Document Outline	04/20/23	05/06/2023
Product Summary	04/27/23	05/06/2023
Usability Test Plan	04/23/23	05/09/2023
QA Test Plan	04/23/23	05/06/23
Code Review	04/21/23	
Self-Check on Best Practices For Security	04/27/23	
Self-Check on Non-functional Requirements	04/27/23	05/06/2023
Document Submission	04/28/23	

1.Product Summary:

- **SafetyHub:**

To effectively market and sell our app, we would first need to identify our target audience, which in this case, would be the entire population of California.

One strategy would be to make use of social media platforms such as Facebook, Twitter, and Instagram to reach out to California citizens. We would create targeted ads that highlight the benefits of our app, such as real-time updates on natural disasters and public health emergencies. Additionally, we understand that in today's market, influencers can make or break a brand. We want to partner with Californian influencers to also reach out to younger possible users.

Another strategy would be to partner with relevant organizations, such as local emergency management agencies, community groups, and nonprofit organizations.

We would also explore traditional marketing methods such as radio and TV ads, billboards, and print media. These methods could help us reach a broader audience, including those who may not be active on social media.

Lastly, we would prioritize providing excellent customer service to our users. This would include regularly updating our app with the latest information, addressing user feedback promptly, and offering support via email and social media channels. Overall, our marketing strategy would focus on creating awareness of our app, building trust with our users, and providing value to the community.

- **Functional Requirements of our app:**

1. Users shall be able to receive notification alerts
   - Current and past alerts will be in a chatbox that the user can scroll through
2. Users shall have login authentication.
   - To be done via sessions so the user does not have to log in whenever they access a new page.
3. Users shall be able to search through categories or by text
   - The search will be "Fuzzy" using %Like
4. Users shall be able to view entities on the map
   - These entities will consist of covid, weather, etc.
5. System shall have a registration form where users can register
   - Shall contain fields such as name, email, etc. that will be stored in the database. Users that are registering can have the option to submit a doc to apply to be a director account.
6. System shall have authorization levels (user, admin, director)

- There will be three authorization levels: user, admin, and director. The county directors can post content (input data/metrics) whereas the user cannot. Admin accounts have the power to delete alerts, users, and upgrade account roles.

7. Users shall be able to filter by type of emergency
   - Examples of such emergencies include covid, weather, etc. to avoid cluttering the map
8. county directors shall be able to input metrics
   - As an input in the admin management dashboard
9. Users shall be able to change their account settings
   - Users can change the area they want to receive notifications on, and the type of alerts that are pushed to them.

- **Uniqueness:**

Our product is unique because we focus on user's safety. Facebook, Instagram, and TikTok, are apps that offer similar functionalities as we do, but they are primarily social media apps. We on the other hand are not, our UI is designed to be user friendly. We do not bombard the user's screen with icons, notifications, ads. We take advantage of the user's screen and only show what is important. Our product is intuitive and gets straight to the point.

Additionally, we plan on working alongside the local government to ensure that California citizens are the first to know about a tragic event, not news outlets or influencers. California citizens will get first hand information from the local government about safety hazards.

2. Usability Test Plan: Search

Test Objectives:

The search and filters are being tested to ensure when user's press search, the right results are being displayed and to ensure searching is easy to use by any user. The search function plus the filters is the main thing that users will interact with, thus, it needs to be easy to use and understand by users with different levels of technological intelligence.

Test Background and Setup:

System Setup:

The system setup consists of the user's device of their preference, this could be either in the form of a desktop/laptop computer or their mobile smartphone/tablet. There will be no special requirement or hardware that the user must have or buy in order to perform and complete these tests.

Starting Point:

The starting point of this usability test will be from our application's home page, in which the user/tester will need to click the "Map & Search" button to be taken to our map and search page. From this point onwards, the user/tester can carry out the test from start to finish.

Intended Users:

The intended users are citizens in California of varying ages and varying tech-literacy. When wanting to find current emergency alerts, Californians who have limited internet skills should still be able to find the emergencies within a county/city they're looking for.

URL of System: https://saferity.quartz.technology/

What is Being Measured:

This test is meant to test user satisfaction and evaluation. We want to ensure a user, of different tech-literacy, can operate the filters and find a city within California through different means (i.e. county name, city name, and zip code).

Usability Task Description:

1. Select one emergency filter with three different counties
2. Select two different emergency filters in one county
3. Find your county by zip code with all emergencies selected
4. Search up a city with one emergency filter
5. Find all security emergencies in "Orange County"
6. Find all health and weather emergencies in "San Francisco County"
7. Find your city on the map
8. Search all emergencies in the county of your choice

Likert Subjective Test:

Questions:	**Rating:** **1 → strongly disagree** **2→ disagree** **3 → neutral** **4 → agree** **5 → strongly agree**
Were you able to find emergencies in your county?	3 - Yes, I was able to find emergencies in my specified county. The county selector menu allowed me to do this and it was quite easy to use and navigate.
Could you find your zip code/city?	2 - I could find my city from the map and through the search box, although, I couldn't find my city by zip code.
Was it difficult to find a specific kind of emergency?	1 - no, it wasn't very difficult to find a specific kind of emergency. It helped that there were checkboxes where I can specify which emergencies I would like to see.
Were the emergency filters easy to use?	3 - yes, the emergency filters were easy to use.
Were the results easy to read?	3 - yes, the results were easy to read. I like that every county and emergency type are all color-coded depending on the severity of the emergency alert.
Did you understand what the map was showing?	1 - yes, I did understand what the map was showing.
Could you understand the results being presented?	1 - yes, I could understand the results being presented. I know that there are certain emergencies being advised in certain counties/cities throughout California.

3. QA Test Plan: Search

Test Objectives:

The search and filters are being tested to ensure that when the QA testers press the search button, the right results are being displayed and to ensure the searching function works and performs up to client specifications. The search function, checkbox filters, and map page is the main thing that QA testers will interact with, thus, it needs to be easy to use and be accurately evaluated for performance, efficiency, and overall usability.

HW and SW Setup:

Similar to the user's usability testing stage, the hardware and software setup will consist of the QA tester's device of choice (computer or mobile), and an internet browser to access and use our web application.

URL of System: https://saferity.quartz.technology/

Feature to be Tested:

The feature to be tested in the QA Testing stage is searching through the various counties listed and finding a specific emergency alert. This will consist of the QA tester selecting a county(s), selecting an emergency alert type(s), and then entering a city name or keyword into the search field text box to display that search result on the page.

QA Test Plan:

Test # and Description	Results (*PASS* or *FAIL*)
Test #1: Zip Code Search **Description:** Select the Weather and Fire checkboxes only and enter the zip code "94117" in the search bar. Verify that the results are only for Weather and Fire in the county of San Francisco.	**Browser:** Chrome (Desktop) **Input:** Zip Code ("94117") **Expected Output:** Show only alerts in QA tester's specified zip code **Result:** ***PASS*** **Browser:** Safari (iPad) **Input:** Zip Code ("94117") **Expected Output**: Show only alerts in QA tester's specified zip code Result: ***PASS***
Test #2: Invalid Input **Description:** In the search box, enter the color "red" and check all the emergency checkboxes. Verify that no results are returned.	**Browser:** Chrome (Desktop) **Input:** Emergency alert types (all *checked*), search keyword ("red") **Expected Output:** No alerts to be displayed in search results Result: ***PASS***

	Browser: Safari (iPad) Input: Emergency alert types (all *checked*), search keyword (“red”) **Expected Output:** No alerts to be displayed in search results Result: ***PASS***
Test 3: <u>Searching by Emergency</u> **Description:** In the search box, enter “flood” and verify that only results that contain the word “flood” in the title or description are shown.	**Browser:** Chrome (Desktop) **Input:** Weather checkbox (*checked*), search keyword (“flood”) **Expected Output:** Display all weather-related alerts containing “flood” keyword in title or description **Result:** ***PASS*** **Browser:** Safari (iPad) **Input:** Weather checkbox (*checked*), search keyword (“flood”) **Expected Output:** Display all weather-related alerts containing “flood” keyword in title or description **Result:** ***PASS***

4. <u>Code Review:</u>

a) Coding Style:
We use the airbnb coding style and a linter to be consistent on the typescript stack.

b) Peer Review:

Feature: Search
Submitted by: Adrian
Code Reviewed by: Carmelo

<u>Initial Review</u>
The following code is in application/frontend/src/components/pages/map/Sidebar.tsx

Carmelo De Guzman
To: Adrian Vazquez
Cc: Adrian Vazquez
Fri 4/21/2023 11:04 PM

Code Review Notes:

- Headers and In-line code comments are present
- Code syntax and logic are easy to read and follow
- Proper use of coding style is chosen and followed

TODO:

- Provide a short in-line code comment in the frontend search in Sidebar.tsx and the search function call in alerts.service.ts
- Even though you mentioned this, try to implement a search input validation for a 40-character word limit (possibly in the form of a reg-ex or something similar) in the backend side. I'll try to implement something similar from the frontend side

Overall, the backend code is structured very nicely and variables/functions are easy to read and trace back. Excellent Work!

Attached below are screenshots of my code-comments:
Note: disregard the red error squiggly lines, as though are related my system not having Prisma installed

Front-end: The following code is in application/frontend/src/components/pages/map/Sidebar.tsx

```
onClick={async () => { // INSERT brief code comment on the purpose of
                       // this code block, basically the logic here.
                         // Somewhere along the lines of, if user clicks this
                           // checkmark..., then serch function should do this...
  const { checkboxes, input, multiselect } = searchOptions;
  const { WEATHER, SECURITY, FIRE, HEALTH } = checkboxes;
  const types = [];
```

The following code is in: application/backend/src/alerts/alerts.service.ts

Code Review: Search

The calling function:

```
async searchAlerts(searchTerm: string, type?: string, counties?: string) {
  const searchResults = {}; // CHECK: In-line code comments -> PARTIAL -
                                                  // Could add 1 or 2 lines more of code comments
                                                    // in this code block
                                                      // Maybe the purpose or logic
                                                        // of calling search function?
```

Final Review

The following code is in application/frontend/src/components/pages/map/Sidebar.tsx

Carmelo De Guzman
To: Adrian Vazquez
Cc: Adrian Vazquez
Sat 4/22/2023 8:39 PM

Code Review Notes:

- Provided in-line code comments to frontend search in Sidebar.tsx and the search function call in alerts.service.ts
- Implemented search input field validation from the backend side with 40 alphanumeric string length, in the form of a class function

Excellent work on getting the input validation implemented! Everything in the code blocks for searching is now sufficiently coded and commented.

It was a pleasure peer reviewing this code with you!

Attached below are screenshots of my code-comments:

Front-end: The following code is in application/frontend/src/components/pages/map/Sidebar.tsx

```
// If the user clicks the search button
onClick={async () => {    // CHECK: In-line code comments -> GOOD
```

The following code is in: application/backend/src/alerts/alerts.service.ts

Code Review: Search

The calling function now using a Validation Class:

```
/*
  Use case: Is triggered when the front-end client makes a request for alerts.
  Returns:  Alerts specified by the user
*/
async searchAlerts(searchTerm: string, type?: string, counties?: string) {

                                    // CHECK: In-line code comments -> GOOD

  const searchResults = {}; // to be populated by one of the search functions
  // Convert comma delimited user input strings into arrays
  const types = type
    ? type.split(',')
    : ['covid', 'fire', 'security', 'weather'];

  const countiesArr = counties && counties !== "Option 1" ? counties.split(',') : [];
```

The validation is used in searchAlerts() shown below

```
                                    // CHECK: In-line code comments -> GOOD
  // If no search term, perform the search ignoring the search term
  if (!searchTerm) {
    searchPromises = this.createSearchPromisesAll(
      types,
      prismaModels,
      searchResults,
      countiesArr,
    );
  } else {      // CHECK: Implemented Search Input Validation -> YES
    // if search term is alphanumeric and at most 40 characters
    // perform the search, else throw an error
    const errors = await validate(new SearchInput(searchTerm))
    if (errors.length > 0) {
      throw new BadRequestException(errors[0].constraints);
    }
    searchPromises = this.createSearchPromises(
      types,
      prismaModels,
      searchTerm,
      searchResults,
      countiesArr,
    );
  }
```

The Search Input validation class shown below

The Validation Class function:

```
                    // CHECK: In-line code comments -> GOOD
                    // CHECK: Implemented Search Input Validation -> YES
// Class used to validate the search term input in searchAlerts()
class SearchInput {
  constructor(query: string) {
    this.query = query;
  }

  @IsAlphanumeric(undefined, {
    message: 'Query must contain only letters and numbers',
  })
  @MaxLength(40, { message: 'Query must not exceed 40 characters' })
  query: string;
}
```

--
Carmelo De Guzman

...

Reply Reply all Forward

5. Self-Check on Best Practices for Security:

Major assets being protected:

- **Search Bar:**
  As far as protecting our application from SQL injections, we have taken into consideration that the ORM that we are using is injection safe.

  Prisma provides us with an API that allows us to make common database operations. This API ensures SQL injection safety by:

  Parameterized queries: Prisma uses parameterized queries, which means that user input is never directly concatenated into a SQL statement.

  Input validation: Prisma automatically validates user input before it is sent to the database. This includes checking for malicious characters and other forms of input that could be used in a SQL injection attack.

  Query builders: Prisma uses query builders to construct SQL statements, which ensures that queries are generated correctly and prevents developers from making mistakes that could lead to SQL injection vulnerabilities.

  Type-safe APIs: Prisma's APIs are type-safe, meaning developers cannot accidentally pass incorrect data types to the database. This eliminates the risk of SQL injection attacks that result from type mismatch vulnerabilities.

  Meaning, our search bar is completely protected from SQL injections.

- **Authentication of Routes:**
  User's need to be authenticated in order to access the posting of alerts feature, and the admin dashboard. This means that user's that are not admins or county directors, do not access those features, preventing them from making any unwanted change in our app.
- **Protecting passwords:**
  Additionally, we are using the module bcrypt to hash user's passwords and ensure their accounts are safe.

6. Self-Check on Non-functional Requirements:

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 (this is a plus). **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, Spring 2023. For Demonstration Only" at the top of the WWW page. (Important so not to confuse this with a real application). **DONE**