CSC 648/848 SW Engineering Fall 2020

SurgeHut

Section 02

Josue Carreon (Team Lead) jcarreo1@sfsu.edu

Pierre Antione (Front End Lead)

Diana Benavides (GitHub Master & Backend Assistant)

Kevin Chen (Back End Lead)

Harsimran Nandhra (Front End Assistant)

Milestone 4

Documentation & Version	Date Submitted	CTO/CEO Feedback	Revision Date
Milestone 4, Version 1	11/10/2020		

PRODUCT SUMMARY

In order to find information regarding the two large scale events that California had to deal with, people had to check a wide variety of sources to find the pertinent information. However, with SurgeHut the entire process can be streamlined and expedited. By providing a central platform where California residents can both access and update information (with proper verification), a significant amount of time would be saved and the users would get a very convenient, streamlined interface to work with.

Functions:

- Working COVID case map, with the ability to zoom and navigate and see cases per county.
- Working fire map and AQI map with the same functionality.
- User registration and login.
- Ability for verified users to update the map data for COVID, fires and AQI.
- Ability to navigate between pages.
- A notification system for alerts on COVID and wildfires for users.

USABILITY TEST PLAN

TEST OBJECTIVE:

The purpose of a usability test is to identify a user's interactions experience with the website's Search functionality. This will help to find weakness in the website's workflow. This is helpful to create a simpler and improved usage flow. It is to identify if the product will be useful or not for the users, before the final product is completed. Users will use the search functionality on the website, without any experience of the website, so this will let us know if the functions have been effective and efficient.

TEST BACKGROUND AND SETUP

System setup:

User tester will need a computer, an internet access to get access to the website and a browser: firefox or google chrome, URL address.

Starting point:

On the homepage of the website, users will look for a search button to find the information about different counties.

Intended users:

-Any one in California who need to get some information about COVID_19, testing areas, or COVID hotspots and also for wildfire information, and to get alerts, once they register in their county.

-Government officials, doctors, or scientists who would have to update information would be Admins.

-Anyone who has access to the Internet and has knowledge of browsing on the internet.

URL of the system: https://3.131.158.22/

What is to be Measured:

We are looking for user satisfaction when they are using the web pages. A user tester will be satisfied if they are searching for a certain county and they are easily able to obtain the information they wanted. A user would be satisfied if the website is simple, easy to access, and displays results and images without any delays and errors.

- To share their experience of the website, testers will be answering questionnaires on the Likert scale, post their testing**.**

Usability Task Description:

A user will open a browser and search the URL " https://3.131.158.22/". The homepage of "SURGEHUT" will appear. On the Homepage, general information about COVID and wildfire would be displayed but in order to find information about a certain county, they will need to enter the name of their county in the search section. The search section will show the names of the county that matches with the entered data.

To measure effectiveness: if users are able to easily find the search section with just a few characters typed, and does misspell still shows their search or not.

To measure efficiency: when a user enters data, how fast do they get the desired results, and no delays in displaying images and information.

Lickert subjective test:

1. Website, pictures and search loaded quickly

Strongly disagree	disagree	Neither agree or disagree	agree	Strongly agree

2. Website was easy to use

Strongly disagree	disagree	Neither agree or disagree	agree	Strongly agree

3. I was able to find the information easily, I was searching.

Strongly disagree	disagree	Neither agree or disagree	agree	Strongly agree

QA TEST PLAN

Test Objectives: For our QA Test we will be using the ability to enter COVID data into our database, the privileges that are included in the Admin account for the application and the ability to enter a search term in the search bar and receive and browse results from the database.

Test #:	Test 1: Enter Metrics
Test Title:	Enter metrics into application
Test Description:	This test will determine if it is possible to enter data into the application, save it in the database and then display that information back into the webpage. We will attempt to enter both COVID data and wildfire data into the form and see if it is outputted in the results page when searching for that specific county
Test Input:	Inputting 52 COVID cases and 2 fire cases to San Mateo County
Expected Correct Output:	Once validated by an admin, San Mateo County should display 52 COVID cases per 100k and 2 fire cases when searched or when hovered over on the map
Test Result:	PASS

Test #:	Test 2
Test Title:	Test Admin Account Privileges
Test Description:	Ensure that Admin account has access to special privileges in the application, such as approving metrics entered and sending emails/alerts to users
Test Input:	Validating metrics entered by users by clicking ‘Validate’ button on Checking page
Expected Correct Output:	When validating metrics entered results should appear on details displayed by county on map
Test Result:	PASS

Test #:	Test 3
Test Title:	Search & Browse for Results
Test Description:	Testing if searched and viewing results is successful when entering a county to view or hovering over counties on the map
Test Input:	Enter a county name in the search bar
Expected Correct Output:	Expected to show specific county on map and display COVID and wildfire results
Test Result:	PASS

CODE REVIEW

Based on *React.js*, our code is organized in components and each component has a unique name that indicates its purpose. The name of each variable, class, and function are self-explanatory and the use of comments is implemented for explaining each functionality. Property management is cleaned up by dropping each property of an object to the next line, making it more readable. Dynamic rendering is also used in this web app to map out an array of data by using a unique key. *Next.js* allows us to use component-level CSS styling which supports the file naming convention as the following: **'[component].module.css'**. Proper indentation that corresponds to the logical structure of the function or program, is also implemented.

Below is the snapshot of the code review via-email:

code to send alert to user Inbox ×

Pierre Antoine <pierre.antoine@epitech.eu>
to me

// /pages/alert.js

```
import React from 'react';
import {ProtectRoute} from '../contexts/auth';

import Navbar from '../components/Navbar';
import {AlertRegisterForm, SendAlertForm, CancelAlertForm} from '../components/Form';

const db = require('../lib/db');
const escape = require('sql-template-strings');

// alert page
// data is the information of all county get from the County database

function Alert({data}) {
return (
<ProtectRoute>
<div>
<Navbar/>
<AlertRegisterForm
idData="County"
dataCounty={data}
/>
<SendAlertForm
idData="County2"
dataCounty={data}
/>
<CancelAlertForm
idData="County3"
dataCounty={data}
/>
</div>
</ProtectRoute>
);
}

// get all county information in the County database for this page

export async function getServerSideProps({req, query}) {
const county = await db.query(escape`
SELECT *
FROM County
`);

return {props: {data: county}};
}


export default Alert;
```

// components/Form.js

```
// Form that save a mail in the data base in case we need to send
// a email to a user when his county is concerned
// idData is the id of the dom element of the dataCounty
// dataCounty is an array of json where there is the name of all county

function AlertRegisterForm({idData, dataCounty}) {
const [mail, setMail] = useState('');
const [county, setCounty] = useState('');

return (
<div>
<p>Register your mail to receive alert from your county</p>
<TextInput
type={'text'}
value={mail}
placeHolder={'Enter your mail'}
onTextChange={setMail}
/>
<InputWithChoice
data={dataCounty}
idData={idData}
value={county}
placeHolder={'enter the name of your county'}
onTextChange={setCounty}
/>
<Button
onClick={ () => {
registerAlert(mail, county); // send the mail and county of the user in the Alert database
}}
>
register
</Button>
</div>
);
}
```

```
// Form that send a mail to all user in the Alert database and are in the county concerned
// idData is the id of the dom element of the dataCounty
// dataCounty is an array of json where there is the name of all county

function SendAlertForm({idData, dataCounty}) {
const [county, setCounty] = useState('');
const [level, setLevel] = useState('');

const {user} = useAuth(); // get information on the current user connected

return (
<div>
<p>Send a alert in a county</p>
<InputWithChoice
data={dataCounty}
idData={idData}
value={county}
placeHolder={'enter the name of the county'}
onTextChange={setCounty}
/>
<TextInput
type={'number'}
value={level}
placeHolder={'Enter the new level of alert'}
onTextChange={setLevel}
/>
<Button
onClick={ () => {
sendAlert(user, county, level); // send a email to all user in the county concerned and tell the alert level
}}
>
send
</Button>
</div>
);
}

// Form that send a mail to all user in the Alert database and are in the county concerned
// idData is the id of the dom element of the dataCounty
// dataCounty is an array of json where there is the name of all county

function CancelAlertForm({idData, dataCounty}) {
const [county, setCounty] = useState('');

const {user} = useAuth(); // get information on the current user connected

return (
<div>
<p>Cancel an alert in a county</p>
<InputWithChoice
data={dataCounty}
idData={idData}
value={county}
placeHolder={'enter the name of the county'}
onTextChange={setCounty}
/>
<Button
onClick={ () => {
cancelAlert(user, county); // send a email to all user in the county concerned
}}
>
send
</Button>
</div>
);
}
```

// api/api.js

```
import Axios from 'axios';

// create url for request

const urls = {
test: `/api/`,
development: 'http://localhost:3333/',
production: 'https://your-production-url.com/',
};

// create a Axios request

const api = Axios.create({
baseURL: urls.test,
headers: {
'Accept': 'application/json',
'Content-Type': 'application/json',
},
});

export default api;
```

// api/alert/alert.js

```
import api from '../api';

// save mail and county in the Alert database

async function registerAlert(mail, county) {
const result = await api.post('alert/create', {mail, county});
console.log(result);
}

// send an alert to all user in a county
// the parameter user is the information of the current user conected to allow this action

async function sendAlert(user, county, level) {
const result = await api.post('alert/send', {user, county, level});
console.log(result);
}

// cancel an alert to all user in a county
// the parameter user is the information of the current user conected to allow this action

async function cancelAlert(user, county) {
const result = await api.post('alert/cancel', {user, county});
console.log(result);
}

export {registerAlert, sendAlert, cancelAlert};
```

// pages/api/alert/create.js

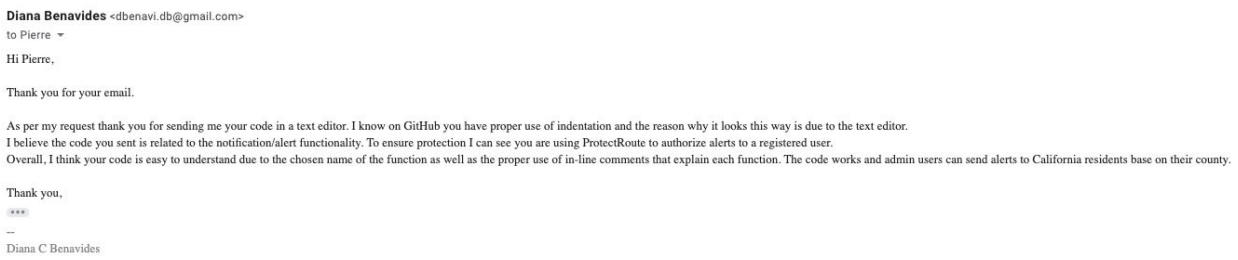
```
const db = require('../../../lib/db');
const escape = require('sql-template-strings');

// api that save the mail and county of a user in the Alert database

module.exports = async (req, res) => {
// sql request to save mail and county in Alert database
const data = await db.query(escape`
INSERT INTO Alert (mail, county)
VALUES (${req.body.mail}, ${req.body.county})
` );

res.status(200);
};
```

// pages/api/alert/cancel.js

```
const db = require('../../../lib/db');
const escape = require('sql-template-strings');

const nodemailer = require('nodemailer');

// create the list of contact for the mail to send
function createContact(mails) {
let result = '';

mails.forEach(function(data) {
result = result + ',' + data.mail;
});

return result;
}


module.exports = async (req, res) => {
// create the mail api
const transporter = nodemailer.createTransport({
host: 'smtp-mail.outlook.com', // hostname
secureConnection: false, // TLS requires secureConnection to be false
port: 587, // port for secure SMTP
tls: {
ciphers: 'SSLv3',
},
auth: {
user: 'surgehut@outlook.fr',
pass: 'ITigreblanc',
},
});

// sql request to update the county's level of alert in County database
const county = await db.query(escape`
UPDATE County C
SET C.evacuation_level = 0
WHERE C.name = ${req.body.county}
` );

// sql request to get all email of a county in Alert databse
const mails = await db.query(escape`
SELECT A.mail
FROM Alert A
WHERE A.county = ${req.body.county}
` );

// create list of receiver
const contact = createContact(mails);

// create the mail to send
const mailOptions = {
from: '"SurgeHut" <surgehut@outlook.fr>', // sender address (who sends)
to: contact, // list of receivers (who receives)
subject: `Alert surgeHut ${req.body.county}`, // Subject line
text: `The level of evacuation has return to normal in ${req.body.county} `,
};

// send the mail
transporter.sendMail(mailOptions, function(error, info) {
if (error) {
console.log(error);
} else {
console.log('Email sent: ' + info.response);
}
});

res.status(200);
};
```

// pages/api/alert/send.js

```
const db = require('../../../lib/db');
const escape = require('sql-template-strings');

const nodemailer = require('nodemailer');

// create the list of contact for the mail to send
function createContact(mails) {
let result = '';

mails.forEach(function(data) {
result = result + ',' + data.mail;
});

return result;
}


module.exports = async (req, res) => {
// create the mail api
const transporter = nodemailer.createTransport({
host: 'smtp-mail.outlook.com' , // hostname
secureConnection: false, // TLS requires secureConnection to be false
port: 587, // port for secure SMTP
tls: {
ciphers: 'SSLv3',
},
auth: {
user: 'surgehut@outlook.fr',
pass: 'ITigreblanc',
},
});

// sql request to update the county's level of alert in County database
const county = await db.query(escape`
UPDATE County C
SET C.evacuation_level = ${req.body.level}
WHERE C.name = ${req.body.county}
` );

// sql request to get all email of a county in Alert databse
const mails = await db.query(escape`
SELECT A.mail
FROM Alert A
WHERE A.county = ${req.body.county}
` );

// create list of receiver
const contact = createContact(mails);

// create the mail to send
const mailOptions = {
from: '"SurgeHut" <surgehut@outlook.fr>', // sender address (who sends)
to: contact, // list of receivers (who receives)
subject: `Alert surgeHut ${req.body.county}`, // Subject line
text: `The level of evacuation has been up to ${req.body.level} in ${req.body.county} `,
};

// send the mail
transporter.sendMail(mailOptions, function(error, info) {
if (error) {
console.log(error);
} else {
console.log('Email sent: ' + info.response);
}
});
res.status(200);
};
```



Diana Benavides <dbenavi.db@gmail.com>
to Pierre

Hi Pierre,

Thank you for your email.

As per my request thank you for sending me your code in a text editor. I know on GitHub you have proper use of indentation and the reason why it looks this way is due to the text editor.
I believe the code you sent is related to the notification/alert functionality. To ensure protection I can see you are using ProtectRoute to authorize alerts to a registered user.
Overall, I think your code is easy to understand due to the chosen name of the function as well as the proper use of in-line comments that explain each function. The code works and admin users can send alerts to California residents base on their county.

Thank you,

--
Diana C Benavides

Major Assets:

Credentials:

- Public users: username, email, password, location
- Fire and health director's authentication
- Administration authorization

Data:

- Fire and COVID-19 data
- Data encryption
- Data validation

To ensure security protection for our assets, we ask registered users and administration users to identify themselves by providing username and password. Admin users and government officials will have a multi-factor authorization sign in policy to validate their identity. We're using Next.js and React.js to validate registration and login forms as well for wildfire and coronavirus data. Each search input is also validated to check data accuracy before allocating to storage. This is done by confirming each input token using cookies. We have secured our database by transporting data from server to browser using HTTPS transit encryption. Our database is password protected and only developers have access to the database.

SELF CHECK: ADHERENCE TO ORIGINAL NON-FUNCTIONAL SPECS

- System shall respond visually within 5 seconds.
    - DONE
- Every image on the WWW shall be royalty free.
    - DONE
- Application shall be developed using responsive UI implementation.
    - DONE
- Application shall not use any mail clients.
    - DONE
- No cost for services transaction shall be addressed, nor simulated in user interface.
    - DONE
- Application should use email confirmation to verify a proper email address from the public.
    - DONE
- Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0.
    - DONE
- Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers.
    - DONE
- Selected application functions must render well on mobile devices.
    - ON TRACK
- Data shall be stored in the team's chosen database technology on the team's deployment server.
    - DONE

- No more than 1000 concurrent users shall be accessing the application at any time.
    - DONE
- Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users.
    - ON TRACK
- The language used shall be English.
    - DONE
- Application shall be very easy to use and intuitive.
    - ON TRACK
- Google maps and analytics shall be added.
    - ON TRACK
- Site security: basic best practices shall be applied.
    - ON TRACK
- Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development.
    - ON TRACK
- The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Fall 2020. For Demonstration Only” at the top of the WWW page.
    - DONE