SW Engineering CSC648/848 Spring 2021

Hermes

Team 03

Milestone 4

May 7, 2021

Roberto Herman - Team Lead (email: rhermanvaldivia@mail.sfsu.edu)
Alex Pena - Backend
Amit Joshi - GitHub Master (email: ajoshi2@mail.sfsu.edu)
Angela Avina Garcia - Front End
Jacob Sebastian - Backend Lead (email: jsebastian@mail.sfsu.edu)
Jarvis Yu - Front End Lead

History Table

Date Submitted	**Date Revised**
05/07/21	05/12/21

1. Product Summary

Name of product: Hermes

What is unique about our product:
Hermes provides a variety of food choices for SFSU students and faculty allowing them to browse through menu items and restaurants. Hermes is a food delivery app that delivers to SFSU students and faculty at any SFSU campus location. Hermes avoids the long dreading lines and prevents the late night MUNI rides to get a warm meal. Hermes' goal is to deliver food at any location within the SFSU campus as fast as possible. With the online food delivery industry growing we want to provide an app that is convenient and easy to use for SFSU students and faculty. Since all food deliveries are being delivered to the SFSU campus it will reduce the trips that the delivery driver would make. It decreases the wait time for delivery and lowers the operation costs for restaurants and drivers. Hermes' app is very accessible to SFSU students and faculty along with restaurant owners and drivers. Its registration process is simple and straightforward where they are guided step by step through the registration. Restaurant owners can

register their menu items and are able to gain exposure for their restaurant. Hermes is uniquely just for SFSU students and faculty allowing only deliveries made to the SFSU campus. Hermes gives detailed instructions to delivery drivers providing drivers with a SFSU campus map. Hermes provides an environment that benefits both students and restaurant owners creating an easy way to purchase and deliver food.

Committed Priority 1 functions:
All Users

1. All users shall be able to register an account as well as modify their account information once registered.

Delivery Drivers

2. Delivery drivers shall be able to see orders to deliver and check them off as they are delivered.
3. Delivery drivers shall have access to the campus map.

Restaurant Owners

4. A restaurant's location shall be displayed on a map.
5. Restaurant owners shall be able to add restaurant information along with image and all its data, as well as its menu.

SFSU customers, staff and faculty

6. SFSU customers, staff, and faculty shall be able to browse through restaurants.
7. The application shall have restaurant search features and filtering functionality.
8. Customers shall be able to add items to a shopping cart and see their itemized list and total price.
9. Customers shall be able to place an order from one restaurant at a time.

URL: http://18.236.138.191/

2. **Usability Test Plan**

Test objectives:

Test the search functions to ensure users have a smooth user experience and are able to access all the features in an efficient manner.

Test background and setup:

- **System setup:** Run the client using 'npm start' command and run the server using 'npm run start:dev' command. This should open up the browser window with the app running.
- **starting point:** Any page on the website will have the search functionality.
- **intended users:** SFSU customers, staff, and faculty
- **URL of the system to be tested:** http://18.236.138.191/

Usability Task description:

With the app running, the tester should be able to find the different restaurants available using the search bar. Furthermore, they should test the functionality which should let the user find restaurants by searching for a particular cuisine and also ensure the user can search using keywords that would match them to restaurants that have the searched word in their description.

Evaluation of Effectiveness:

To evaluate the effectiveness of the search functionality in the app, the tester should search for restaurants by name, cuisines by title, and search using keywords. These searches should return the output we want and display all the results that meet the criteria. Using the data from these tests, we can extract feedback on whether the search functionality made the user experience better and served its purpose of simplifying the process of finding the best matched restaurants. Furthermore, we can also measure the percentage completion of this functionality based on how many respondents got the results they expected while also getting the count of errors they encountered during the evaluation.

Evaluation of efficiency:

To evaluate the efficiency of the search feature in the app, the tester can use the search bar to search for restaurants by name, cuisines by title, and search using keywords. The tester should then ensure that the time required for a result to be displayed is within the expected range and does not keep the user waiting for longer than is acceptable. They should also ensure that the user gets the best results with the least possible effort, for example, in terms of number of clicks or number of pages to be loaded. The tester also needs to ensure that there are sufficient instructions easily available to users who may have questions about navigating the application.

Evaluation of user satisfaction:

Statement	Strongly disagree	Disagree	Neither agree or disagree	Agree	Strongly agree
It was easy to find the search bar from all the pages.					
The search bar takes a wide range of input types which makes it very easy to find what I need.					

The search bar returned the correct output very fast.					
Any comments you may have to improve the search functionality is much appreciated.					

3. **QA Test Plan**

Test objectives: Test the integrity of the restaurant search functionality.
HW and SW setup: For hardware, all you need is a device with access to the internet. For software, you need to install a web browser to access the website.
Feature to be tested: Search

Test #1

Title	Restaurant search by cuisine
Description	The user wants to look at restaurants of a certain type of cuisine
Input	User selects the dropdown menu next to the search bar and selects “Italian”
Expected correct output	One italian restaurant card named Pizzarino is returned and is displayed on the screen
Results	PASS

Test #2

Title	Restaurant search by name
Description	The user wants to browse for restaurants with a given name
Input	Enter “Bob’s Burgers” into the search bar
Expected correct output	Data from a restaurant with the name “Bob’s Burgers” is returned and it is displayed on the screen
Results	PASS

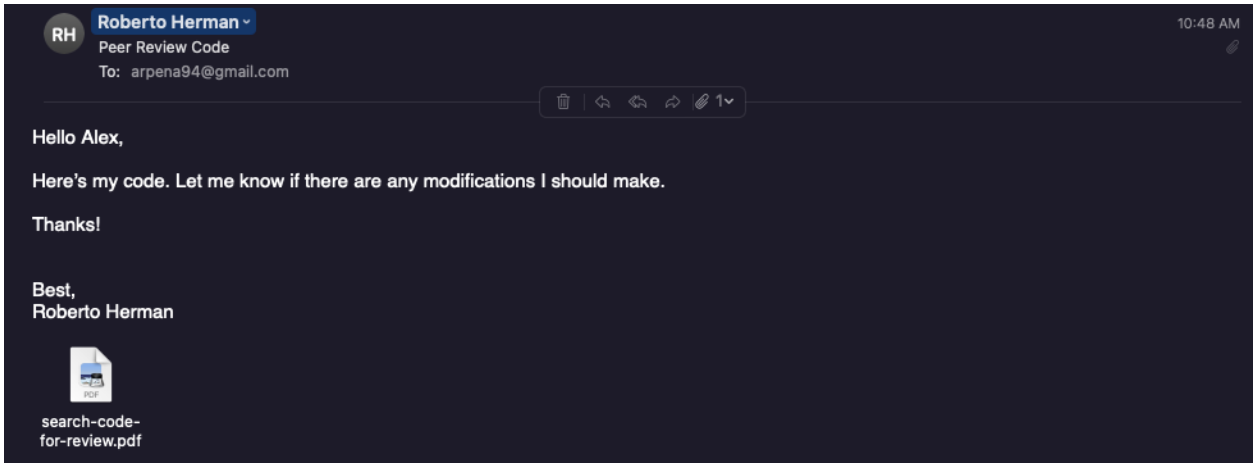
Test #3

Title	Search for a restaurant with an arbitrary keyword

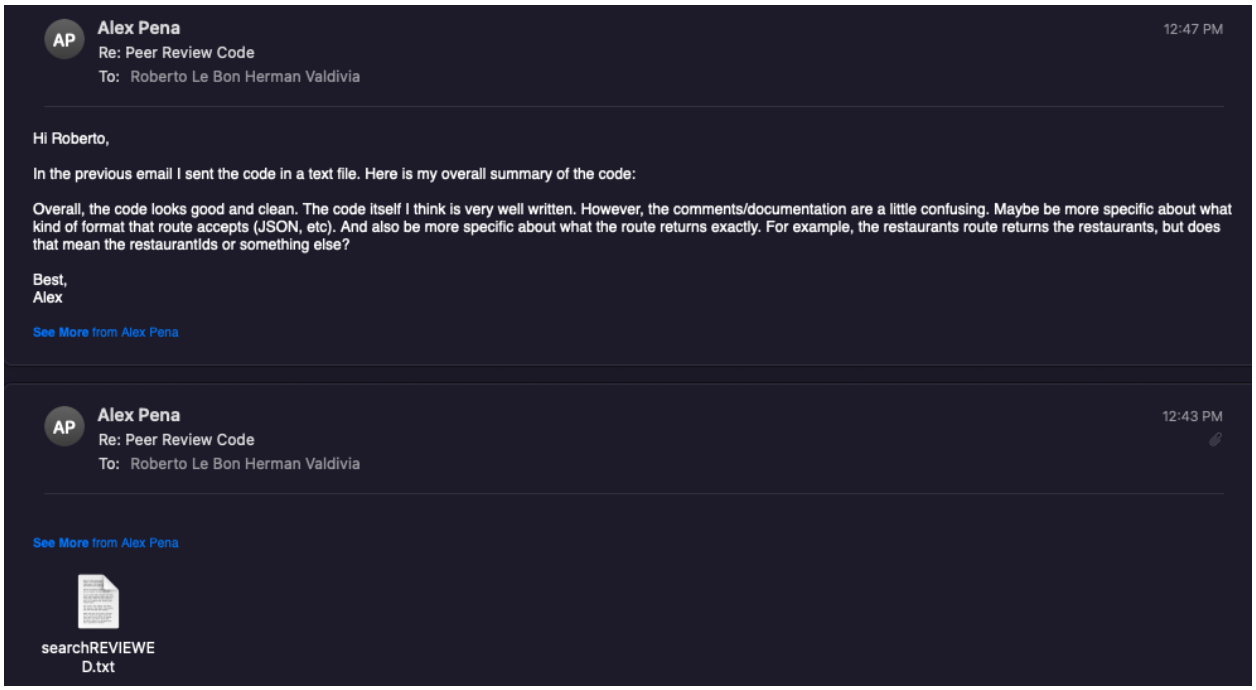
Description	The user wants to search for a restaurant using a keyword other than the name or category of the restaurant.
Input	Enter “delicious” into the search bar
Expected correct output	One restaurant will be returned and displayed since the title or description of the restaurant contains the search term “delicious”
Results	PASS

4. **Code Review**

Code review request email:



Code reviewer’s reply with summary of comments:



The code with the in-line reviewer comments:

```
/* CODE REVIEW COMMENTS:
 * 1. Needs a file header explaining what the file is for
 * 2. '/restaurant' - unclear what data type this returns.
 * 3. '/restaurant/restaurant' endpoint naming seems redundant.
 *
 * Overall: Code looks good and clean. The code itself I think is very well written.
               However, the comments/documentation are a little confusing
               Maybe be more specific about what kind of
               format that route accepts (JSON, etc). And also be more specific about
               what the route returns exactly. For example, the restaraunts route returns
               the restaurants, but does that mean the restaurantIds or something else?
*/

const express = require('express');
const router = express.Router();
const restaurantModel = require('../models/Restaurant');
const utilModel = require('../models/Util');
const addressModel = require('../models/Address');

// utility function to insert address into restaurant object
async function insertRestaurantAddress(restaurants) {
    for (let restaurant of restaurants) {
        // console.log('passing in id:', restaurant.restaurantId);
        let addressRes = await addressModel.getAddressById(restaurant.restaurantId);
        restaurant.address = addressRes;
        delete restaurant.addressId;
    }
}

// returns all restaurants. If query has a name, then match name. Works for:
// /api/search/restaurant?name=<restaurantName>
// /api/search/restaurant?cuisine=<cuisine>
// /api/search/restaurant?
// *** CODE REVIEW: How does this function get its input and what is its output? ***
// *** CODE REVIEW: How do the conditions work? ***
router.get('/restaurant', async (req, res, next) => {
    if (req.query.name) {
        let name = req.query.name;
        const restaurantRes = await restaurantModel.getByName(name);
        await insertRestaurantAddress(restaurantRes);
        res.status(200).json({ status: 'ok', restaurants: restaurantRes });
    } else if (req.query.cuisine) {
        let cuisine = req.query.cuisine;
        const restaurantRes = await restaurantModel.getByCuisine(cuisine);
        await insertRestaurantAddress(restaurantRes);
        res.status(200).json({ status: 'ok', restaurants: restaurantRes });
    } else {
        let restaurantRes = await restaurantModel.getAll();
        await insertRestaurantAddress(restaurantRes);
        res.status(200).json({ status: 'ok', restaurants: restaurantRes });
    }
});

// gets a list of all UNIQUE cuisines available in our app (DB). Useful for dropdown.
// ** CODE REVIEW: Similar here with not knowing exactly what the input and outputs
//of this function are***
router.get('/restaurant/cuisines', async (req, res, next) => {
    let cuisines = await utilModel.getAllCuisines();
    res.status(200).json({ status: 'ok', cuisines });
});

// ** CODE REVIEW: is this a good name for a route? 'restarants/rastaurants' **
router.get('/restaurant/restaurants', async (req, res, next) => {
    let restaurants = await utilModel.getAllRestaurants();
    res.status(200).json({ status: 'ok', restaurants });
});

module.exports = router;
```

5. **Self-check On Best Practices For Security**

Asset to be protected	**Types of possible/expected attacks**	**Your strategy to mitigate/protect the asset**
Username	Unauthorized user gains access, Cross site scripting, SQL Injection	Check input and only allow valid ones, limiting the length of input
Password	Unauthorized user gains access, Cross site scripting, SQL Injection	Encrypting the password using bcrypt, limiting the length of input
Email	Unauthorized user gains access, Cross site scripting, SQL Injection	Validating email by requiring students and faculty to have it ending with *sfsu.edu* or *mail.sfsu.edu* and requiring drivers and restaurant owners to have a email ending in *.com* or *.net*
Database	SQL Injection	Validate input
Images	Image replacements	Restrict access to images directory, Restaurant owner validation

- I confirm the PW in DB is being encrypted
- I confirm input data is being validated on all forms, including search bar.

6. **Self-check: Adherence to original non-functional specs**

Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0. Application delivery shall be from chosen cloud server	**DONE**
Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers	**ON TRACK**
All or selected application functions must render well on mobile devices (specifics to be developed in consultation with users e.g. Petkovic)	**ON TRACK**

Ordering and delivery of food shall be allowed only for SFSU students, staff and faculty	**DONE**
Data shall be stored in the database on the team's deployment cloud server.	**DONE**
No more than 50 concurrent users shall be accessing the application at any time	**ON TRACK**
Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.	**DONE**
The language used shall be English (no localization needed)	**DONE**
Application shall be very easy to use and intuitive	**ON TRACK**
Application should follow established architecture patterns	**DONE**
Application code and its repository shall be easy to inspect and maintain	**ON TRACK**
Google analytics shall be used	**ON TRACK**
No e-mail clients shall be allowed.	**DONE**
Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.	**DONE**
Site security: basic best practices shall be applied (as covered in the class) for main data items	**ON TRACK**
Application shall be media rich (images, maps etc.). Media formats shall be standard as used in the market today	**ON TRACK**
Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development	**ON TRACK**
The application UI (WWW and mobile) shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Spring 2021	**DONE**

For Demonstration Only at the top of the WWW page. (Important so as to not confuse this with a real application).	