Milestone 4

SW Engineering CSC 648 - 848 Spring 2023

Food Delivery Application

GatorBites

Team 02

Team Lead: Dylan Burns (dburns6@sfsu.edu)
Github Master/Back End: Christian Jackson
Front End Lead: Tim Polich
Back End Lead: Kevin Aziz
Front End: Wallace Man
Back End: Jeremiah Ruvalcaba

Date Submitted	Date Revised
05/24/2023	

1 Executive Summary

GatorBites

Introducing GatorBites - the ultimate food delivery app designed exclusively for the hungry and busy SFSU students, staff, and faculty. Our goal is simple - to provide a hassle-free experience for students and staff by delivering delicious food right to their classrooms.

We understand that time is of the essence and that's why we've leveraged our specialized knowledge of the SFSU campus, leading to a whopping 25% reduction in delivery times. Say goodbye to cold food and frustration, as our in-app navigation tool provides our drivers with real-time routing directly to your classroom of choice.

We take pride in our team of active students who work part-time to pay for their college education. These students are familiar with the campus, making food delivery an absolute breeze. However, we understand that not all our drivers may be familiar with the campus, which is why we have a stringent sign-on screening process to ensure they know their way around before joining our team.

At GatorBites, we make food delivery easy. With just four simple steps - create an account, add a payment method, enter your delivery location, and order your food - you can have your favorite meals delivered to you in no time.

Whether you're having a busy day and forgot to bring lunch to campus, need a coffee to power through your late-night classes, or just craving a cheesy pizza for a group study session, GatorBites has got you covered. We know that students and faculty are always on the move, constantly working and planning, and often forget to eat at regular intervals. With GatorBites, you can focus on what really matters, while we take care of your food delivery needs.

Join us today and experience the ultimate food delivery service designed exclusively for SFSU students and staff - GatorBites!

Priority 1 Functions

Unregistered Users

1.1 Unregistered users shall be able to access and browse the web page.
1.2 Unregistered users shall be able to add, remove, and modify any selected items that are placed inside their checkout cart.
1.3 Unregistered users shall be able to freely search for a particular item or Restaurant.

SFSU Registered Users

1.1 Registered users shall be able to login.
1.2 Registered users shall be able to read their own account information.
1.3 Registered user shall be able to update their own account information
1.4 Registered users shall be able to delete their own account information.
1.5 Registered users shall be able to access their payment method an account settings at any point while on the web page.
1.6 Registered users shall be able to inherit all related functions of the unregistered user.
1.7 Registered users shall be able to order.

Admin

1.1 Admin shall be able to create their own account.
1.2 Admin shall be able to input account information.
1.3 Admin shall be able to read their own account information.
1.4 Admin shall be able to update their own account information
1.5 Admin shall be able to remove users or restaurants.
1.6 Admin shall be required to approve restaurant registration
1.7 Admin shall be required to approve the restaurant application.

Product URL: http://34.102.89.98/

2 Usability Test Plan

Test Objective

The major function that we will be testing is adding menu items to a restaurant owner's menu page. This function is important because it allows the restaurant owners who are logged in to their account to be able to set up their menu and even add new items later on.

Test Background and Setup

The system setup will be in a browser on a computer. The test can occur on both Safari and Google Chrome. The computer needs to have either of these two browsers and be connected to the Internet.

The starting point of the usability test will be on the Homepage of our web application. Clicking on the URL below will automatically bring you the Homepage to start the test.

The intended users for this function are the restaurant owners who have filled out the restaurant registration form and have been approved by our Admin. These are the users that will be able to post new menu items onto the database and have it appear in their menu page when their restaurant is clicked on by a customer.

Product URL: http://34.102.89.98/

With these usability tests, we are measuring user satisfaction. This means we are looking for feedback on how easy it was for our focus group to adapt and comfortably use our User Interface. We will be measuring this using the Likert tests that we created ourselves.

Usability Task Description

1. First step of this test is to click on our URL to get to our application. Once you click on it, it should bring you to our homepage.
2. Next, because this is intended for restaurant owners, you need to login as a restaurant owner. Go to the login form for restaurant owners.

3. Once you see a login form, click on the link that will take you to the login for restaurant owners. This will bring you to a separate login page for restaurant owners.
4. Enter your email and password and login.
5. Once you have logged in as a restaurant owner, go to the drop-down menu in the top right and click Edit Menu.
6. Once you do that, you can then add items and fill out the information to be posted on the database and it will display properly on your restaurant menu page
7. This concludes the usability test for adding new menu items as a registered restaurant owner.

Effectiveness Evaluation

In our plan to measure effectiveness, we will first test the completion of certain tasks as well as use cases. These can include logging in as a driver, restaurant, or user. It can also be going through the process of ordering food through our food delivery application or filling out restaurant and driver registration forms. We will measure the number of people who completed the defined tasks within a reasonable time frame and calculate the percentage of task completion. We will also count the amount of errors per task to figure out in what ways our user interface is confusing. This will help us narrow down the aspects of our application and find where our users are getting stuck.

Efficiency Evaluation

To measure efficiency of our application, we will measure and calculate the average time it takes to complete the specified task. We will also be measuring the number of clicks and screens it takes to complete the task. This will be measured automatically using Google Analytics to figure out our website's traffic and where our users lose interest.

User Satisfaction Evaluation

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The process of adding menu items is easy and intuitive					
The process of adding menu items is efficient					
In real application, you would use this function					
I felt that the application was responsive					
The look and feel of the user interface felt approachabl e					

3 QA Test Plan

Test Objectives

The objective of this test is to make sure that restaurant login works, the JavaScript of adding and deleting menu items works properly. Lastly, when the confirm button is clicked, the database is updated with the newly created menu items and rendered properly in the correct restaurant menu page.

Hardware/Software Configuration

For the QA (Quality Assurance) plan, the following hardware and software setup is required to perform testing on the website:

Hardware

Computer system with a minimum of 4GB RAM and a dual-core processor.

Stable internet connection with adequate bandwidth.

Monitor with a screen resolution of at least 1280x800 pixels.

Keyboard and mouse for interaction.

Software

Operating System: Windows 10 or macOS Mojave (or later versions).

Web Browsers

Google Chrome version 95.0.4638.54 or later: This browser will be used for compatibility testing and to ensure optimal performance on the most widely used browser.

Mozilla Firefox version 94.0 or later: This browser will be used to verify cross-browser compatibility and to identify any browser-specific issues.

Feature Tested

The feature being tested allows a registered, logged in restaurant owner to be able to add a new item to their menu. It should take the info from the menu item cards and upload them to the database where they can be rendered.

Product URL: http://34.102.89.98/.

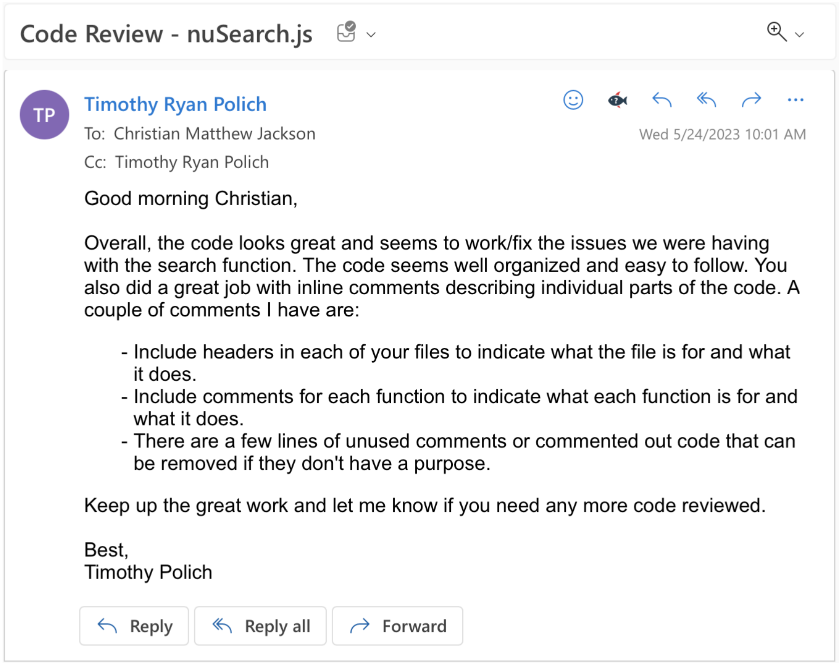
Test #	Title	Description	Input	Expected Correct Output	Test Results
1	Logging in as a restaurant owner	Enter restaurant owner's login credentials	Email: kevinaziz11@gmail.com Password: 10102	Create a user session, bring you to the home page, and change to user nav bar	PASS - Chrome PASS - Firefox
2	Creating a new menu item	Clicking the "Add" button to create a new menu item form where the owner can fill out and once they click "Confirm", the application will POST to the database	Click "Add"	A form asking for information about your new menu item should appear	PASS - Chrome PASS - Firefox
3	Sending new menu item's info to the database	POST to the database and enter the info into the correct	Name: "Pizza" Price: "10"	The application will POST the new menu item	PASS - Chrome PASS - Firefox

		restaurant's database	Description: "Slice of Italy"	on the database. When you search for "Kevin", the new menu item should be rendered	

4 Peer Code Review

Reviewee Christian Jackson
Reviewer Timothy Polich
File nuSearch.js

- Christian reached out during a voice chat meeting about reviewing his code.


Code Review - nuSearch.js

Timothy Ryan Polich
To: Christian Matthew Jackson
Cc: Timothy Ryan Polich
Wed 5/24/2023 10:01 AM

Good morning Christian,

Overall, the code looks great and seems to work/fix the issues we were having with the search function. The code seems well organized and easy to follow. You also did a great job with inline comments describing individual parts of the code. A couple of comments I have are:

- Include headers in each of your files to indicate what the file is for and what it does.
- Include comments for each function to indicate what each function is for and what it does.
- There are a few lines of unused comments or commented out code that can be removed if they don't have a purpose.

Keep up the great work and let me know if you need any more code reviewed.

Best,
Timothy Polich

```
// Need header describing what this file is for/does.
function displayRestaurants(restaurants) {
    // Clear any previous search results
    const popularRestaurantsSection =
document.querySelector('section.popular-restaurants');
    popularRestaurantsSection.innerHTML = '';

    // Update the number of search results
    const numberOfResults =
document.querySelector('h3.number-of-results');
    numberOfResults.textContent = restaurants.length + ' Results';

    // Create a new card for each restaurant
    restaurants.forEach(function (restaurant) {

        // Create a new div with the class 'restaurant-card'
        const restaurantCard = document.createElement('div');
        restaurantCard.classList.add('restaurant-card');

        const handleImageClick = () => {
            window.location.href =
`/restaurantMenuPage/${restaurant.restaurantID}`;
        };

        const restaurantImage = document.createElement('img');
        restaurantImage.src = restaurant.image_url;
        restaurantImage.classList.add('clickable');
        restaurantCard.appendChild(restaurantImage);

        restaurantImage.addEventListener('click', handleImageClick);

        // Add the restaurant name
        const restaurantName = document.createElement('h2');
        restaurantName.textContent = restaurant.restaurant_Name;
        restaurantCard.appendChild(restaurantName);

        // Add the delivery information
        const deliveryInfo = document.createElement('span');
        deliveryInfo.textContent = ` 3.5 mi `;
        deliveryInfo.classList.add('delivery-info');
        restaurantCard.appendChild(deliveryInfo);

        // Add the delivery fee
        const deliveryFee = document.createElement('span');
        deliveryFee.textContent = `${restaurant.delivery_time} `;
        deliveryFee.classList.add('delivery-fee');
        deliveryInfo.appendChild(deliveryFee);
        // Add the restaurant rating'

        const restaurantRating = document.createElement('span');
        deliveryFee.classList.add('star');
        restaurantRating.innerHTML = `4.5 <i class="fas fa-star">
</i>`;
        restaurantCard.appendChild(restaurantRating);

        const heartIcon = document.createElement('span');
        heartIcon.classList.add('heart-icon', 'far', 'fa-heart');
        restaurantCard.insertBefore(heartIcon, restaurantImage);

        heartIcon.addEventListener('click', function () {
            // Send a POST request to toggle the
favorite status
            fetch('/favorites', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({
                        restaurant_id: restaurant.restaurantID
                    }),
                })
                .then(function (response) {
                    return response.json();
                })
                .then(function (response) {
                    if (response.success) {
                        // Update the heart icon based on the
server's response
                        if (response.action === 'added') {
                            heartIcon.classList.remove('far',
'heart-outline');
                            heartIcon.classList.add('fas',
'heart-red');
                        } else {
                            heartIcon.classList.remove('fas',
'heart-red');
                            heartIcon.classList.add('far',
'heart-outline');
                        }
                    } else {
                        console.log('Error updating favorites:',
response.message);
                    }
                });
        });
        popularRestaurantsSection.appendChild(restaurantCard);
    });
}
Need function headers describing what each function does.
async function handleFormSubmit(event) {
    event.preventDefault();

    const searchTerm = document.getElementById('search-input').value;
    const cuisineType =
document.getElementById('category-select').value;
    let url;

    if (searchTerm === '' && cuisineType != '') {
        url = '/getCuisineType?searchTerm=' + cuisineType
    } else {
        url = '/getRestaurants?searchTerm=' + searchTerm
    }
We can remove these if not being used.
    //console.log('searchTerm', searchTerm);
    //console.log('cuisineType', cuisineType);
    //console.log("attempting to fetch from url: ", url);

    try {
        const response = await fetch(url).then(function (response) {
            return response.json();
        });
        let specificRestaurants = response;

        fetch('/getAllRestaurants')
            .then(function (response) {
                return response.json();
            })
            .then(function (allRestaurants) {
           We can remove this if not being used.
                // Call the displayRestaurants() function with all
the

                let newRestaurants = [];

                for (let i = 0; i < specificRestaurants.length; i++)
{
                    for (let j = 0; j < allRestaurants.length; j++) {
                        if (specificRestaurants[i].restaurantID ===
allRestaurants[j].restaurantID) {
                            newRestaurants.push(allRestaurants[j]);
                        }
                    }
                    if (window.location.pathname === '/') {
                        displayRestaurants(newRestaurants);
                    }
                }
            });

    } catch (error) {
        console.error(`Error fetching search results:
${error.message}`);
    }
}

async function fetchRestaurants() {
    const response = await fetch('/getAllRestaurants');

    if (response.ok) {
        const data = await response.json();
        return data;
    } else {
        throw new Error(`Error fetching restaurants:
${response.statusText}`);
    }
}

/**
 * Add an event listener to the search form to call the
handleFormSubmit function
 * when the form is submitted
 */
document.addEventListener("DOMContentLoaded", () => {
    const searchForm = document.querySelector("form.search-form");
    searchForm.addEventListener("submit", handleFormSubmit);
});
```

5 Self check on Best Practices for Security

Asset to be protected	**Vulnerabilities/Types of Attacks**	**Mitigation/Protection Strategy**
User Personally Identifiable Information (PII)	Unauthorized access, data breaches	Implement strong access controls, encryption of sensitive data, regular security audits, and monitoring systems for suspicious activity.

Database containing user credentials	SQL injection, unauthorized access	Implement input validation, parameterized queries, and prepared statements to prevent SQL injection attacks. Encrypt passwords using strong hashing algorithms and salt.
Web application infrastructure	DDoS attacks, Cross-site scripting (XSS), Cross-site request forgery (CSRF)	Implement DDoS mitigation techniques, use secure coding practices to prevent XSS and CSRF vulnerabilities, regularly update and patch software and frameworks.

- Yes, we encrypt passwords in the database using strong hashing algorithms and salt to protect user credentials.
- Input data validation:
    - For the search bar input, we validate that it contains up to 40 alphanumeric characters, preventing any potential injection or malicious input.
    - During SFSU customer registration, we ensure that the email address provided includes "sfsu.edu" at the end to verify the user's affiliation with the university

6 Adherence to Original Non-Functional Specs

Original Non-Functional Requirements

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0. **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. All or selected application functions shall render well on mobile devices. **DONE**
4. Data shall be stored in the database on the team's deployment server. **DONE**
5. No more than 50 concurrent users shall be accessing the application at any time. **DONE**
6. Privacy of users shall be protected. **DONE**
7. The language used shall be English (no localization needed) **DONE**
8. Application shall be very easy to use and intuitive. **DONE**
9. Application shall follow established architecture patterns. **DONE**
10. Application code and its repository shall be easy to inspect and maintain. **DONE**
11. Google analytics shall be used. **ISSUE**
12. No email clients shall be allowed. Interested users can only message to sellers via in-site messaging. One round of messaging (from user to seller) is enough for this application. **ISSUE**
13. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. **DONE**
14. Site security: basic best practices shall be applied (as covered in the class) for main data items. **DONE**
15. Media formats shall be standard as used in the market today. **DONE**
16. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development. **DONE**
17. The application UI (WWW and mobile) 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 nav bar. (Important so as to not confuse this with a real application). **DONE**