

SW Engineering CSC648/848 Spring 2023

Food Delivery App: GatorGrub

Team 3

Shauhin Pourshayegan: spourshayegan@mail.sfsu.edu	Team Lead, Github Master, Document Editor
Hieu Ma:	Front End Lead
Preetham Ramesh:	Back End Lead
Lin Tun:	Front End Developer
Derrick Liang:	Back End Developer

Milestone 4

Product description, Beta Launch, QA and Usability Testing

Date Submitted:	5/xx/2023
-----------------	-----------

1) Product summary

GatorGrub:

Gator grub is a food delivery app which offers SFSU students, faculty and staff convenient and affordable ways to get food delivered to them. People at universities often find themselves stuck when searching for their next meal. Food options on campus are likely few and far between, and have the potential to taste like the ever-dreaded dining hall food. Students and instructors alike find issues managing their food habits when heavy workloads and packed schedules take precedent. As students of San Francisco State University, we find ourselves trekking back and forth to Stonestown Galleria just so we can get access to more options, wasting our time when it's most valuable to us. Other than the inconvenience of food locations and availability, the food available is not always affordable, which is a major issue for students and important for everyone. The inconvenience of our current options and the lack of affordable options are the primary driving forces of our team project.

Being students, we wanted to provide a solution to these issues when we came to discuss what GatorGrub will be. We propose a service which will strengthen the connection between SFSU students, faculty and staff to their local food scene, as well as offer this primary user base with features which make their lives easier and solve the problems related to getting food on campus. GatorGrub aims to provide these users with a selection of local restaurants offering delivery. Customers who don't have the time to leave their classroom or study area can get their food delivered to any room on campus, allowing their workshops and group study sessions to be uninterrupted by rumbling tummies. Customers can also decide whether they want their food delivered to one of several safe pickup spots throughout SFSU's campus. This feature benefits students who want to get food safely, without worrying about who they will encounter on the long walk to Stonestown or further out. GatorGrub also enables its users to get deals on different local restaurants. This will incentivise more sales, making restaurant owners and delivery drivers content and strengthening GatorGrub's infrastructure.

On GatorGrub, anyone who joins can:

- Unregistered users shall be able to search by restaurant category.
- Unregistered users shall be able to search for food and restaurants.
- Unregistered users shall be able to search the restaurant map from their location.
- Unregistered users shall be able to sort restaurants by delivery time.

- Unregistered users shall be able to sort restaurants by price.
- Unregistered users shall be able to select menu items and add menu items to their cart.
- Unregistered users shall be able to choose an order's delivery location.
- Unregistered users shall be able to read restaurant menus.
- Unregistered users shall be able to register an account with SFSU email.
- Unregistered users shall be able to register an account as a restaurant owner
- Unregistered users shall be able to register an account as a driver

SFSU students, faculty and staff can:

- SFSU Customers shall inherit all unregistered user functional requirements.
- SFSU Customers shall be able to log into their account.
- SFSU Customers shall be able to submit an order.

Restaurant owners can:

- Restaurant owners shall be able to register for an account.
- Restaurant owners shall be able to log into their account.
- Restaurant owners shall be able to upload menu item pictures and prices.
- Restaurant owners shall be able to submit restaurant details. (e.g. hours of operation, avg. order time.)
- Restaurant owners shall be able to choose to accept or reject an order.
- Restaurant owners shall confirm when orders are ready. This updates the order progress.
- Restaurant owners shall be able to review and manage driver tasks.

Drivers for GatorGrub can:

- Drivers shall be able to register as a driver.
- Drivers shall be able to log in as a driver.
- Drivers shall be able to accept or reject an order.
- Drivers shall confirm when food is picked up. This updates the order progress.
- Drivers shall confirm when food is delivered. This updates the order progress.
- Drivers shall be able to see a map of the pickup point.
- Drivers shall be able to see a map of the delivery point.

2) Usability test plan for selected function

Selected Function for Testing Usability : Searching

I. Test objectives:

The search functionality serves as a fundamental component within our web application, as it equips users with a valuable means to refine their results. The efficacy and user-friendliness of the search feature, as well as its accuracy in addressing user input, will establish the initial and most crucial impression.

Furthermore, input validation is essential and must undergo thorough testing prior to the final release. On one hand, it is important to limit the length and content of user input to prevent any unsuitable code injections. Conversely, when input is deemed valid, we ought to display items that closely align with the user's search query.

Ultimately, the testing objectives outlined above aim to gather user feedback, empowering our development team to enhance the user experience for both current and prospective customers.

II. Test background and setup

(a). System Setup

The system setup process is both swift and straightforward. From the user's standpoint, there is no need for installation. To test the search function, users simply need to launch a web browser on their computer or laptop and ensure that they are connected to the internet. We currently support all the latest versions of web browsers, including Google Chrome, Firefox, Microsoft Edge, and more. By entering the URL into their web browser, users can easily access our food delivery website, "GatorGrub."

(b). Starting Point

The starting point for the search function is the homepage, which can be accessed via the URL provided below. The search bar, which is used to initiate the search function, is located just above the navigation bar. If the user's input is valid in terms of character(s) and length, the corresponding search results will be displayed on the same page. However, if the input is invalid, the search result will be blank, with no restaurants displayed.

(c). Intended User

We understand that users may simply want to search our website without the need to register or log in. As such, for the search function, users of all types, including unregistered, registered, and admin users, are intended users. We aim to provide a seamless and hassle-free experience for anyone who wishes to search for items on GatorGrub, without any limitations or restrictions. Therefore, users can search for items on our website without the need to register or log in.

(d). Task

Task #	Task for Usability Test
1	Find All the restaurants.
2	Find Chinese restaurants.
3	Find a restaurant called Bevande Venue.

(e). URL to be Tested

<http://34.82.124.237:3001/>

III. Questionnaire

Below is a Likert-Scale Questionnaire with questions inquiring user's satisfaction after performing the above tasks. Each "question" is a statement with five choices (from **Strongly Agree** to **Strongly Disagree**) scaling user's response.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Category browsing is intuitive.					
Search function is convenient to use.					
The user interface for finding restaurants is pleasant to use.					
The search function was user-friendly.					
Overall, I am satisfied with the search function and category browsing on Gator Grub.					

Comments:
(Your feedback are especially appreciated)

3) QA test plan and QA testing - max 2 pages (please consult class slides on this topic)

- a) For the same function you chose for the usability test, write a QA test plan (check class slides), with brief and separate sections as follows:
- **HW and SW setup** (including URL):
 - **Feature to be tested:**
 - **QA Test plan: in table format:** This is the plan to be given to QA tester to execute your QA test plan. Contains min of 3 test cases and results of testing them on your system: appr. 1 page. You must provide QA test plan in a separate section in the easy to read tabular form allowing easy reading and analysis by management e.g. like presented in the class slides on SW QA.

I. Test Objectives

Beyond user satisfaction, the QA test for the search function primarily focuses on quality, specifically the accuracy of results and deployment. In order to assess the correctness of the search function, each search result is examined to determine if the item title closely corresponds to the input text. Furthermore, the total number of results is tallied to ensure that all matching records in the database have been successfully retrieved.

II. HW and SW Setup

(a). Server Host

- The server host has been installed on the operating system of AlmaLinux version 9.1.
- Google Compute Engine has been installed on the Windows System, with the free-tier services selected for our web application.
- Google Compute Engine Server has been allocated with 1vCPU, 2 GB RAM.

(b). Software Setup (need to ask and add by back-end)

- MySQL 8.0.32 has been successfully setup, which is required for running the database.
- Express.js 4.18.2 has been successfully setup, which is required for running the web server.
- Node.js 18.14.1 has been successfully setup, which is required for running the server-side language.
- Node.bcrypt.js 5.1.0 library has been successfully setup, which is required for

implement the encryption for all user password.

- Fuse.js 6.6.2 library has been successfully setup, which is required for search function.

(c). APIs

- Google Analytics Reporting API v4
- Google Maps JavaScript API
- Google Routes API

(d). Web Browser

- In this QA testing, QA tester will use on the web browser of the latest 2 versions, Google Chrome and Microsoft Edge browsers.
- Most of the computers have these browser either installed or automatically upgraded. If not available, the QA tester may have to download them and install.
- The URL for QA test is <http://34.82.124.237:3001/>

III. Features to be tested

Below will be the features of the search function, and the table of QA test plan.

(a). Selection group of categories

- Upon selecting the "All" category, users will be able to view all the restaurants listed on the website. Notably, this category is also the default setting when users first visit our homepage.
- When users switch to other categories, the search results should correspond to the specific category selected. This ensures that users are presented with relevant search results that align with their intended search criteria.

(b). Approximate matching

- For text search, the results do not have to perfectly match the user's input. Instead, approximate matching will be executed.

(c). Responsiveness

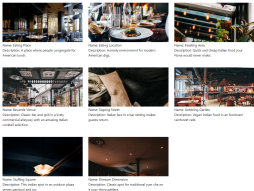

- By expanding or shrinking the browser, the list page of search results is responsive to the change of browser size.


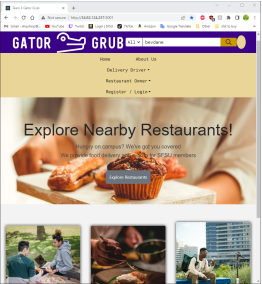
IV. Test Plan

To enhance the readability of the test plan, a tabular format is provided below. This format allows users to easily comprehend the details included in the QA test.

The columns in the QA table have been structured according to both standard and recommended formats, which include the following: test number, test description, test input, expected output, and test results (either PASS or FAIL) on both Google Chrome and Microsoft Edge (the two latest versions for each browser).

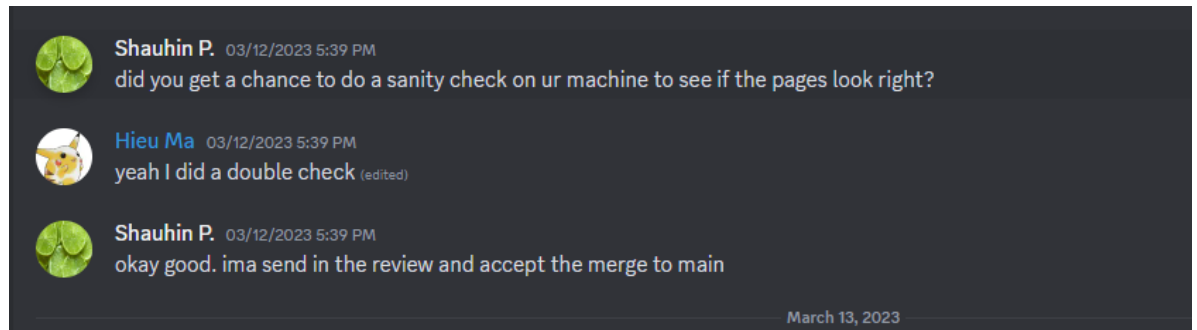
This format ensures that all necessary information is included in the table and that the test results are consistent across different browsers and versions.

Test Number	Test Title and Description	Test Input	Expected Output	Test Results (Chrome)	Test Results (Edge)
1	Selection group of categories - Test for the Search bar: Upon selecting the "All" category, users will be able to view all the restaurants listed on the website.	<ul style="list-style-type: none"> On the homepage of GatorGrub, navigate to the search bar at the top of the screen. "All" should already be selected. Click on the search icon to the left. Observe the results on the search results page. 	 <p>An empty string means all restaurants. Under "All" category, there will be 8 restaurants.</p>	Pass	Pass
2	Selection group of categories - Search specific category: Users expect to be able to search for food by cuisine. The test confirms that the search function should allow users to do this.	<ul style="list-style-type: none"> On the homepage of GatorGrub, navigate to the search bar at the top of the screen. "All" should already be selected. Click on the cuisine. Change the selected cuisine to "Italian". Click on the search icon to the left. Observe the results on the search results page. 	 <p>Results: Name: Sipping Street Description: Italian fare in a bar setting makes guests return.</p> <p>Name: Feasting Area Description: Quick and cheap Italian food your Nona would never make.</p> <p>Name: Bevande Venue Description: Classic bar and grill in a lively commercial alleyway with an amazing Italian</p>	Pass	Pass

			cocktail selection.		
3	<p>Approximate matching search:</p> <p>There must be some allowance for typos in search queries. The test must ensure the search can function regardless of simple typos.</p>	<ul style="list-style-type: none"> On the homepage of GatorGrub, navigate to the search bar at the top of the screen. "All" should already be selected. Click on the search bar input field. Enter or type "bevdane avenue" in this field. Click on the search icon to the left. Observe the results on the search results page. 	<p>"bevdane avenue" is a misspelling of "Bevande" from the restaurant "Bevande Venue".</p> <p>Results:</p>  <p>Name: Bevande Venue Description: Classic bar and grill in a lively commercial alleyway with an amazing Italian cocktail selection.</p> <p>Name: Bevande Venue Description: Classic bar and grill in a lively commercial alleyway with an amazing Italian cocktail selection.</p>	Pass	Pass
4	<p>Responsiveness:</p> <p>By expanding or shrinking the browser, the list page of search results should be responsive to the change of browser size.</p>	<ul style="list-style-type: none"> Have the browser window maximized. Be on the homepage of GatorGrub. Drag the window to the side of the screen to collapse it horizontally. Observe the results. 		Pass	Pass

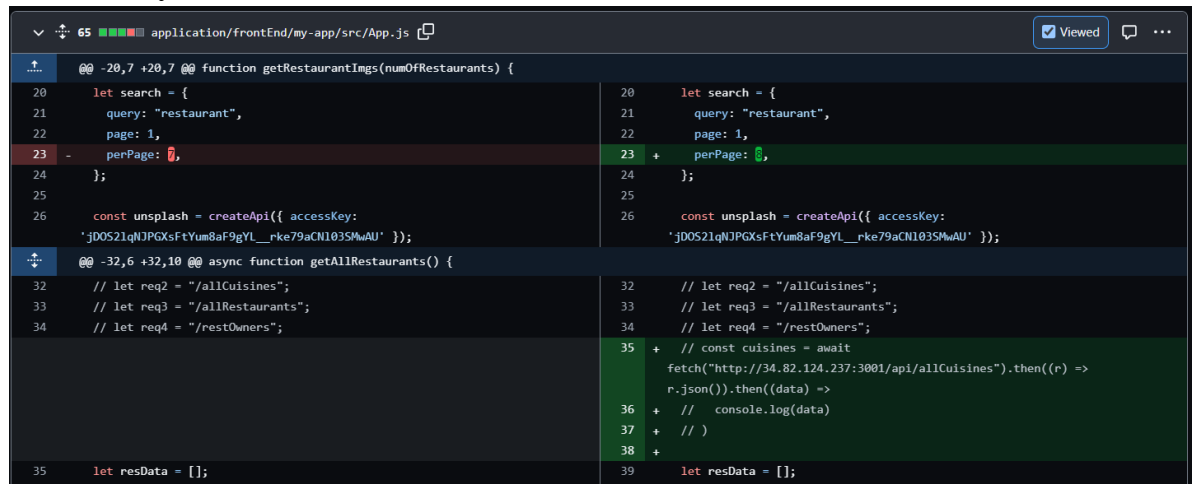
4) Peer Code Review:

Person who code is being reviewed sends e-mail to reviewer with pointer to the code and asks for review:




Reviewer reviews the code in whatever way is most practical for you (e.g. commenting on code in repository, or using github review options):


For this code review, the reviewer was on call with the reviewee. They skimmed through code together before the reviewer had to review independently with the reviewee still listening to hear feedback immediately.




<pre> 36 const res = await fetch("http://34.82.124.237:3001/api/allRestaurants").then((r) => r.json()).then((data) => 37 resData = data @@ -48,10 +52,19 @@ async function getSearchRestaurants(search) { 48 return resData; 49 } 50 </pre>	<pre> 40 const res = await fetch("http://34.82.124.237:3001/api/allRestaurants").then((r) => r.json()).then((data) => 41 resData = data </pre>
<pre> 51 function App() { 52 53 const [search, setSearch] = useState(''); 54 const [searchResult, setSearchResult] = useState(''); </pre>	<pre> 52 return resData; 53 } 54 55 + async function getSearchRestaurantsWithCategory(search, category) { 56 + let resData = []; 57 + const res = await 58 fetch("http://34.82.124.237:3001/api/search/\${category}/\${search}").then((59 r) => r.json()).then((data) => 60 resData = data 61) 62 + return resData; 63 + } </pre>
<pre> 55 const [serverData, setServerData] = useState([{}]); 56 57 @@ -67,26 +80,43 @@ function App() { 67 }) 68 69 getRestaurantImgs(restaurants.length).then((r) => { </pre>	<pre> 63 function App() { 64 65 const [search, setSearch] = useState(''); 66 const [searchResult, setSearchResult] = useState(''); 67 + const [searchResultCategory, setSearchResultCategory] = useState(''); 68 69 const [serverData, setServerData] = useState([{}]); 70 80 }) 81 82 getRestaurantImgs(restaurants.length).then((r) => { </pre>
<pre> 70 - setRestaurantImages(r.response.results); 71 - }) 72 - }, []); 73 - 74 - // Search Use Effect 75 - useEffect(() => { 76 - let newRestaurants = []; 77 - 78 - console.log("GOT RESPONSE", searchResult); 79 - 80 - getSearchRestaurants(searchResult).then((r) => { 81 - console.log(r) 82 - for (let i = 0; i < r.length; i++) { 83 - r[i]["item"]["ImgUrl"] = restaurantImages[i].urls.regular; 84 - } 85 - console.log("search restaurants use effect", r) 86 - setSearchRestaurants(r); 87 - }); </pre>	<pre> 83 + setRestaurantImages(r.response.results); 84 + }) 85 + }, []); 86 + 87 + // Search Use Effect 88 + useEffect(() => { 89 + let newRestaurants = []; 90 + 91 + if (searchResultCategory !== 'all') { 92 + getSearchRestaurantsWithCategory(searchResult, 93 searchResultCategory).then((r) => { 94 + for (let i = 0; i < r.length; i++) { 95 + r[i]["item"]["ImgUrl"] = restaurantImages[i].urls.regular; 96 + } 97 + console.log("search restaurants use effect", r) 98 + setSearchRestaurants(r); 99 + }); 100 + } else if (searchResultCategory === 'all' && searchResult === '') { 101 + getAllRestaurants().then((r) => { 102 + let newRestaurants = []; 103 + for (let i = 0; i < r.length; i++) { 104 + newRestaurants.push({item: r[i]}); 105 + } 106 + for (let i = 0; i < newRestaurants.length; i++) { 107 + newRestaurants[i]["item"]["ImgUrl"] = 108 restaurantImages[i].urls.regular; 109 + } 110 + setSearchRestaurants(newRestaurants); 111 + }); 111 + } else { 112 + getSearchRestaurants(searchResult).then((r) => { </pre>


<pre>88 89 - }, [searchResult]); 90 91 // useEffect(() => { 92 // let newRestaurants = restaurants; @@ -104,12 +134,11 @@ function App() { 104 useEffect(() => { 105 let newRestaurants = restaurants; 106 107 - if (restaurantImages) { 108 for (let i = 0; i < restaurants.length; i++) { 109 - // console.log(restaurantImages[i]?.urls?.regular); 110 111 newRestaurants[i]["ImgUrl"] = restaurantImages[i]?.urls?.regular 112 ?? ''; 111 } 112 - } 113 114 setRestaurants(newRestaurants); 115 }, [restaurantImages, restaurants]); @@ -118,7 +147,7 @@ function App() { 118 <></pre>	<pre>112 + for (let i = 0; i < r.length; i++) { 113 + r[i]["item"]["ImgUrl"] = restaurantImages[i]?.urls?.regular; 114 + } 115 + setSearchRestaurants(r); 116 + }); 117 + } 118 119 + }, [searchResult, searchResultCategory]); 120 121 // useEffect(() => { 122 // let newRestaurants = restaurants; 134 useEffect(() => { 135 let newRestaurants = restaurants; 136 137 + // if (restaurantImages) { 138 for (let i = 0; i < restaurants.length; i++) { 139 + newRestaurants[i]["ImgUrl"] = restaurantImages[i]?.urls?.regular 140 ?? 'https://images.unsplash.com/photo-1517248135467-4c7edcad34c4?ixlib=rb- 141 4.0.3&ixid=MnwzMjA3fDB8MjBwaG90by1wYld1fHx8fGVufD88fHx8&auto=format&fit=cr 142 op&w=1170&q=80'; 140 } 141 + // } 142 143 setRestaurants(newRestaurants); 144 }, [restaurantImages, restaurants]); 147 <></pre>
<pre>119 <div className="App"> 120 <header className="App-header"> 121 - <Navbar search={search} setSearch={setSearch} setSearchResult= 122 {setSearchResult} /> 122 </header> 123 </div> 124 <Routes></pre>	<pre>148 <div className="App"> 149 <header className="App-header"> 150 + <Navbar search={search} setSearch={setSearch} setSearchResult= 151 {setSearchResult} setSearchResultCategory={setSearchResultCategory} /> 151 </header> 152 </div> 153 <Routes></pre>
<pre>12 13 - async function submitOrder(orderPrice, location, restaurantID, userName, 14 setModalShow, selectedItems) { 14 console.log("submit order login username", userName) 15 if (userName) { 16 // console.log("submitted order"); @@ -23,8 +27,8 @@ async function submitOrder(orderPrice, location, restaurantID, userName, setModa 23 'Content-Type': 'application/json' 24 }, 25 body: JSON.stringify({ 26 - CustomerID: 0, 27 - DriverID: 0,</pre>	<pre>16 17 + async function submitOrder(orderPrice, location, restaurantID, userName, 18 setModalShow, selectedItems, userID) { 18 console.log("submit order login username", userName) 19 if (userName) { 20 // console.log("submitted order"); 27 'Content-Type': 'application/json' 28 }, 29 body: JSON.stringify({ 30 + CustomerID: userID, 31 + DriverID: 0,</pre>
<pre>28 RestaurantID: restaurantID,</pre>	<pre>32 RestaurantID: restaurantID,</pre>

LimeSauc3 marked this conversation as resolved.  Hide resolved

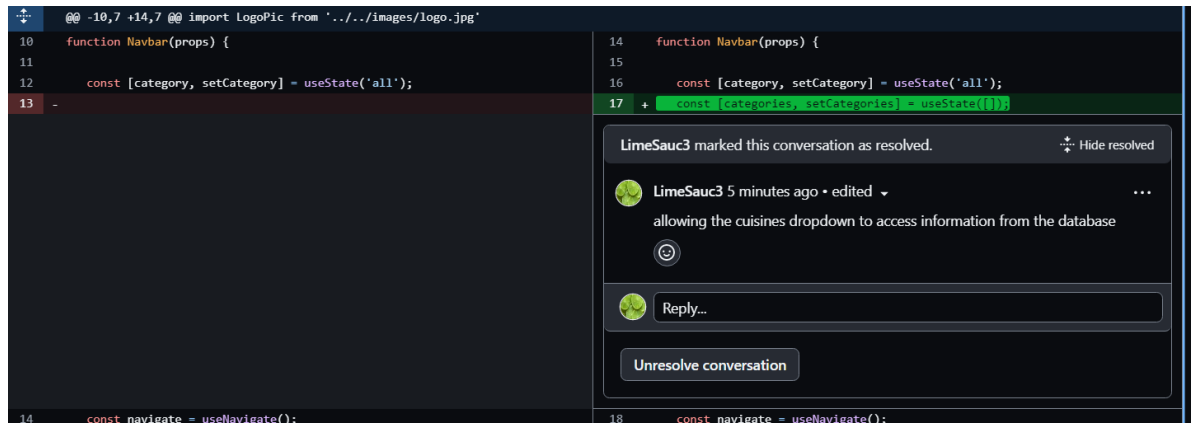
 LimeSauc3 now ...

This fixes a hardcode issue where the orders funnel into one person's account.

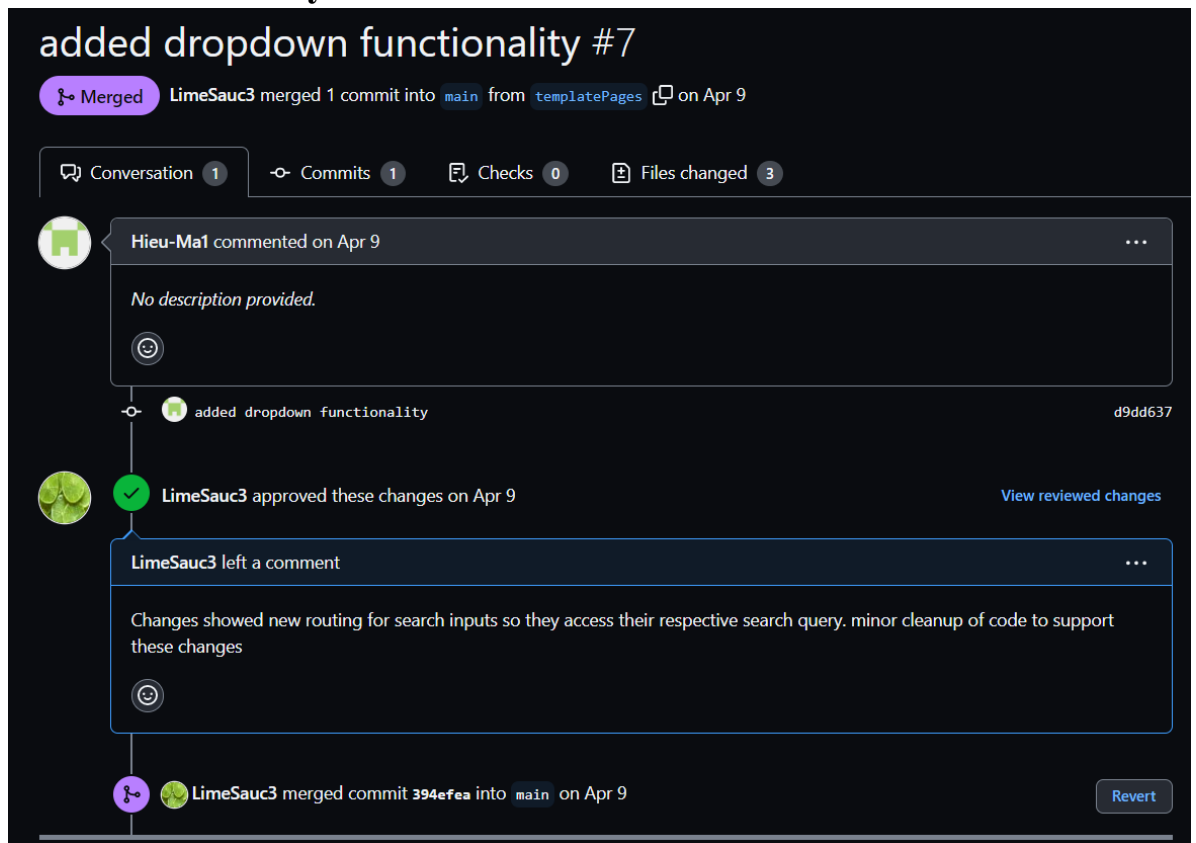


 Reply...

Unresolve conversation



Reviewer sends summary of review in email back to coder:



added headers with authors for each file, removed old code from creat... #14

Open

Hieu-Ma1 wants to merge 1 commit into main from connectApi

Conversation 5

Commits 1

Checks 0

Files changed 55

+246 -16

Hieu-Ma1 commented 13 minutes ago

...ing order route, and made categories dropdown pull from database

added headers with authors for each file, removed old code from creat...

7c7ea85

LimeSauc3 reviewed 11 minutes ago

View reviewed changes

application/backEnd/routes/index.js

165 + // for (let i = 0; i < formData.Items.length; i++){

166 + // results = await db.enterOrderItems(formData[i].menuItemID, formData[i].menuItemID.count,

167 + // }

LimeSauc3 11 minutes ago

Commenting out old code that was messing with creating orders.

Reply...

Resolve conversation

LimeSauc3 reviewed 9 minutes ago

View reviewed changes

Reviewers

LimeSauc3

Still in progress? Convert to draft

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

Notifications

Unsubscribe

You're receiving notifications because you commented.

The screenshot displays a GitHub pull request interface. At the top, there are three review entries from the user 'LimeSauc3':

- Review 1: 'LimeSauc3 reviewed 7 minutes ago' with a diff view of `application/frontEnd/my-app/src/components/navbar/Navbar.js` and a 'Show resolved' button.
- Review 2: 'LimeSauc3 reviewed 5 minutes ago' with a diff view of `application/frontEnd/my-app/src/pages/restaurant/Restaurant.js` and a 'Show resolved' button.
- Review 3: 'LimeSauc3 approved these changes 1 minute ago' with a 'View reviewed changes' link.

Below the reviews, a comment box shows a comment from 'LimeSauc3':

Overall good work. Nice job adding headers on all files and making small QA related tweaks. I see there was work done on orders and cuisine categories. Good to see that and we need a few more changes related to how order items are saved and displayed. Maybe we need to have a group chat with backend to figure out any missing routes.

At the bottom, a green box indicates the pull request status:

- Changes approved**: 1 approving review by reviewers with write access. [Learn more.](#)
- 1 approval**
- This branch has no conflicts with the base branch**: Merging can be performed automatically.
- Merge pull request** button: You can also open this in GitHub Desktop or view command line instructions.

The bottom of the interface shows a 'Write' and 'Preview' section for adding a new comment, with a rich text editor toolbar.

5) Self-check on best practices for security – ½ page

Asset	Types of possible attacks	Mitigation strategy
User Information	Password Breach	We make sure to have

	Unauthorized user gains access to confidential data	encrypted passwords. Require users to authenticate themselves. Track system usage.
Database Systems	SQL Injection Unauthorized user makes system unavailable	We have parameterized queries so users can't input their own sql statements

6) Self-check of the adherence to original Non-functional specs – performed by team leads

Current Development Status of 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, but no mobile native app is to be developed.	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.	ON TRACK
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.	DONE
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 “ <i>SFSU Software Engineering Project CSC 648-848, Spring 2023. For Demonstration Only.</i> ” at the top of the WWW page nav bar. (Important so as to not confuse this with a real application).	DONE