Project description 2: Online food delivery platform The aim here is to create an online food delivery platform that connects users with local restaurants. The system allows users to browse menus, place orders, make payments, and have food delivered to their doorstep. The platform shall provide a seamless and user-friendly experience for both customers and restaurant owners.

List of test cases and test purposes:

NORMAL USER CASES

The test input/output files for testing front-end for users are located in **CISC_327_31/client/test/user**

User Registration

Verify that a user can successfully register for an account with a username, email, and password.
Associating test file: **user_registration.json**

Input:
The testing function will take three arguments: username, email, password
Output:
status is true and a success message is displayed if valid information has been provided.
status is false and an error message is displayed if invalid information has been provided.

Test case: Successful Registration with Valid Information
Objective: To verify that users can successfully register for an account with valid information.
Arrange: Open the registration page.
Act: Fill in valid user details and click the "Register" button.
Assert: Check that the user is registered and redirected to the login page with a success message.

Test case: Unsuccessful Registration with Invalid Information
Objective: To verify that users cannot register for an account if invalid information is provided.
Arrange: Open the registration page.
Act: Fill in invalid user details and click the "Register" button.
Assert: Check that the user is not registered and display an error message that invalid information has been provided.

User Login

Verify that a user can successfully log in to their account using their email or username and password.
Associating test file: **user_login.json**

Input:
The testing function will take two arguments: username_or_email and their password
Output:
status is true and a success message is displayed if the username/email and password are a valid combination, and a redirect happens
status is false and an error message is displayed if the username/email and password are an invalid combination

Test case: Successful Login with Username
Objective: To verify that a user can successfully log in to their account with a valid username and password combination.
Arrange: Register an account, then open the log-in page.
Act: Enter a valid username with the corresponding password, and click the “Login” button.
Assert: Check that the user is successfully logged in and redirected to the main page with a success message.

Test case: Successful Login with Email
Objective: To verify that a user can successfully log in to their account with a valid email and password combination.
Arrange: Register an account, then open the log-in page.
Act: Enter a valid email with the corresponding password, and click the “Login” button.
Assert: Check that the user is successfully logged in and redirected to the main page with a success message.

Test case: Unsuccessful Login with Nonexistent Username
Objective: To verify that a user cannot log in to an account with a nonexistent username.
Arrange: Open the log-in page.
Act: Enter a nonexistent username with any password, and click the “Login” button.
Assert: Check that the user is unable to log in and display an error message that username/email and password do not match.

Test case: Unsuccessful Login with Nonexistent Email
Objective: To verify that a user cannot log in to an account with a nonexistent email.
Arrange: Open the log-in page.

Act: Enter a nonexistent email with any password, and click the “Login” button.
Assert: Check that the user is unable to log in and display an error message that username/email and password do not match.

Test case: Unsuccessful Login with Username and Password That do not Match
Objective: To verify that a user cannot log in to an account with an invalid username and password combination.
Arrange: Register an account, then open the log-in page.
Act: Enter with the username used to create the account with an incorrect password, and click the “Login” button.
Assert: Check that the user is unable to log in and display an error message that username/email and password do not match.

Test case: Unsuccessful Login with Email and Password That do not Match
Objective: To verify that a user cannot log in to an account with an invalid email and password combination.
Arrange: Register an account, then open the log-in page.
Act: Enter with the email used to create the account with an incorrect password, and click the “Login” button.
Assert: Check that the user is unable to log in and display an error message that username/email and password do not match.

Retrieve User Location

Get user location to find local restaurants
Associating test file: **user_retrieve_location.json**

Input:
The testing function will take 5 arguments: street, postal_code, city, province_or_state, country
Output:
status is true and a success message is displayed if the location is valid. Otherwise, status is false and an error message is displayed.

Test case: From Device - Success
Objective: To retrieve the customer’s current location from their system.
Arrange: Navigate to the “Restaurant” section of the platform.
Act: Select “From current location” and allow the platform access to the device’s location.
Assert: Check that the location is valid.

Test case: From Device - Fail
Objective: To retrieve the customer’s current location from their system.
Arrange: Navigate to the “Restaurant” section of the platform.
Act: Select “From current location” and block the platform from accessing the device’s location.
Assert: Check that the location is valid.

Test case: Manual Entry - Success
Objective: To retrieve a location from manual entry by the user.
Arrange: Navigate to the “Restaurant” section of the platform.
Act: Select “Manual entry” and enter a valid location.
Assert: Check that the location is valid.

Test case: Manual Entry - Fail
Objective: To retrieve a location from manual entry by the user.
Arrange: Navigate to the “Restaurant” section of the platform.
Act: Select “Manual entry” and enter an invalid location.
Assert: Check that the location is valid.

Selecting a Restaurant

Verify that users can successfully browse and select restaurants to order food.
Associating test file: **user_res_selection.json**

Input:
The testing function will take in one argument: restaurant
Output:
status is true if the list of restaurants is successfully displayed (and a redirect happens) and false otherwise.

Test case: Selecting a Restaurant
Objective: To verify that users can successfully browse and select restaurants to order food.
Arrange: Open the food delivery platform and log in to a user account.
Act: Navigate to the "Restaurant" section of the platform and select a restaurant to browse.
Assert: Check that the user can view a list of restaurants, browse their menus, and select a restaurant to view its details.

Browsing the Menu

Verify that users can successfully browse and explore the food menu of a selected restaurant.
Associating test file: **user_menu_browsing.json**

Input: The ID of the button associated with opening the menu of the restaurant. It is “menu” for now.
Output:
status is true and a new link to the restaurant page happens and a message if the operation is successful

Test case: Browsing the Menu
Objective: To verify that selecting a restaurant directs the user to the selected restaurant’s menu.
Arrange: Open the food delivery platform and log in to a user account. Select a restaurant in the Restaurant section.
Act: Navigate to the menu section of the selected restaurant and start browsing the available food items.
Assert: Check that the user can view a comprehensive menu with food items, descriptions, prices, and options.

Placing an Order

Verify that users can place orders for selected food items from a restaurant’s menu.
Associating test file: **user_order_placement.json**

Input: restaurant name alongside an array of items ordered, delivery address, and payment method.
Output:
status is true alongside a success message, a list of items ordered, and the payment method if the operation is successful.

Test case: Placing an Order
Objective: To confirm that users can place an order for selected food items from a restaurant's menu.
Arrange: Log in to a user account, select a restaurant, and browse the menu.
Act: Add desired food items to the cart, specify order details (e.g., quantity, special requests), and proceed to checkout.
Assert: Check that the user receives a confirmation of the placed order and that the order details are accurate.

Add Items to the Shopping Cart

Verify that users are able to add items to a holding area.
Associating test file: **user_cart_addition.json**

Input: the list containing the id of the items that will be added to the shopping cart

Output:
status is true and a message confirming the order and a list of items containing their names, prices, and quantity is displayed if the operation is successful.

Test case: Add Items to the Shopping Cart

Objective: To verify that users are able to add items to a holding area.
Arrange: Select an item. Assume that the item exists on the menu.
Act: Select an "Add to cart" button
Assert: Check that the quantity of the selected item is greater than 0. Check that the correct quantity of the item appears in the user's cart, with a success message.

Remove Items from the Shopping Cart

Verify that users can successfully remove items from their shopping cart.
Associating test file: **user_cart_removal.json**

Input: ID of the item to be removed
Output:
status is true and the list of the remaining items if the removal of item is successful.

Test case: Remove Items from the Shopping Cart

Objective: To verify that users can successfully remove items from their shopping cart.

Arrange: Add items to the shopping cart to create a test scenario with items in it.

Act: Navigate to the shopping cart section, select one or more items, and click the "Remove" or "Delete" button.

Assert: Check that the selected items are successfully removed from the shopping cart, and the cart's total reflects the removal of the items.

Payment Processing

Verify that payments can be accepted and declined based on entered information.

Associating test file: **user_payment.json**

Input: ID of the user account, ID of their order, their payment method, their card number details, and their billing address
Output:
status is true if the payment is a success.
status is false if the payment is rejected.
Test case: Payment Processing - Successful Payment
Objective: To verify that the payment processing system can successfully process a user's payment.
Arrange: Prepare the necessary payment information, including the user's ID, order ID, payment method, card details, and billing address.
Act: Initiate the payment process with the provided payment information.
Assert: Check that the payment processing system returns a success status and provides a transaction ID along with a confirmation message indicating the successful payment.

Test case: Payment Processing - Insufficient Funds
Objective: To verify that the payment processing system does not process a user's payment if the user does not have enough funds.
Arrange: Prepare the necessary payment information, including the user's ID, order ID, payment method, card details, and billing address. Add items to the shopping cart such that the price is more than the payment allows.
Act: Initiate the payment process with the provided payment information.
Assert: Check that the transaction fails and the payment processing system returns an error message.

Test case: Payment Processing - Invalid information
Objective: To verify that the payment processing system does not process a user's payment if the user enters invalid information.
Arrange: From a restaurant’s menu, add items to the shopping cart, navigate to the shopping cart page, and select “Check out”.
Act: Initiate the payment process with invalid payment information.
Assert: Check that the transaction fails and the payment processing system returns an error message.

Order Tracking

To verify that users know what stages their orders are in (Sent, Received, Prepared, Delivery)
Associating test file: **user_order_tracking.json**

Input: The ID of the order in each respective stage
Output: the stage of the order matching the expected stage

Test case: Order is sent

Objective: To verify that the order status is changed to “Sent” when the order is sent.

Arrange: Place an order from a restaurant. Open the status page of a specific order.
Act: Before the order is received, select the “Status” button when the specific order is selected.
Assert: Check that the order status has been set to “Sent”.

Test case: Order is Received

Objective: To verify that the order status is changed to “Received” when the order is received.

Arrange: Place an order from a restaurant. Open the status page of a specific order.
Act: After the order is received and before the order is prepared, select the “Status” button when the specific order is selected.
Assert: Check that the order status has been set to “Received”.

Test case: Order is Prepared

Objective: To verify that the order status is changed to “Prepared” when the order is prepared.

Arrange: Place an order from a restaurant. Open the status page of a specific order.
Act: After the order is prepared and before the order is delivered, select the “Status” button when the specific order is selected.
Assert: Check that the order status has been set to “Prepared”.

Test case: Order is Delivered

Objective: To verify that the order status is changed to “Delivered” when the order is delivered.

Arrange: Place an order from a restaurant. Open the status page of a specific order.
Act: After the order is delivered, select the “Status” button when the specific order is selected.
Assert: Check that the order status has been set to “Delivered”.

Viewing Order History

To verify that users can access and view their order history.

Associating test file: **user_order_history.json**

Input: User ID

Output:

status is true and the list of past orders of the user if the operation is successful.

Test case: Viewing Order History

Objective: To verify that users can access and view their order history.

Arrange: Log in to a user account and navigate to the order history section.
Act: Click on the order history tab and review the list of past orders.
Assert: Check that the user can see details of previous orders, including order date, items ordered, order status, and total cost.

Search Function

Verify that users can use the search function to find a specific food or restaurant.
Associating test file: **user_search.json**

Input: The name of the food
Output:
status is true and the restaurant with the food containing the query if the operation is successful.
status is false and a message denoting it if the operation is unsuccessful.

Test case: Successful Search
Objective: To verify that users can use the search function to find a specific food or restaurant.
Arrange: Click on the search bar
Act: Type in stuff onto the keyboard, and click on the “Search” or press “Enter”
Assert: Check that the user can see either a specific restaurant, a specific food, or results in a message saying their search does not exist.

Test case: No Results Found
Objective: To verify that when no results are found, a message displays that no results are found.
Arrange: Click on the search bar
Act: Search an item or restaurant that does not yield and search results, and click on the “Search” or press “Enter”
Assert: Check that a message appears saying that no results are found.

User Feedback

To ensure that users can provide feedback and ratings for their orders and the quality of food for the restaurant owner to see.
Associating test file: **user_feedback.json**

Input: Past order id alongside feedback comment and ratings consisting of two categories: quality and delivery_experience, each rated by a nonnegative integer ranging from 0 to 5

Output:
status is true and a success message is displayed if the operation is successful.

Test case: User Feedback
Objective: To ensure that users can provide feedback and ratings for their orders and the quality of food for the restaurant owner to see.
Arrange: Place an order and receive the food delivery.
Act: Navigate to the order history or feedback section, select the specific order, and provide feedback and ratings based on the order's quality, delivery experience, or any other relevant criteria.
Assert: Check that the feedback and ratings are successfully submitted and recorded for the specific order, and that they can be viewed by the user and the food delivery platform.

RESTAURANT OWNER CASES

The test input/output files for testing front-end for users are located in **CISC_327_31/client/test/owner**

Register a Restaurant

To verify that restaurant owners can register their business on the platform. To do so, the registrant must tick the box stating they are a restaurant owner to be able to register as one.
Associating test file: **owner_registration.json**

Input:
The testing function will take in 7 arguments: restaurant_name, restaurant_location, restaurant_type, restaurant_description, owner_name, owner_email, proof
Output:
status is true and a success message is displayed if the registration is successful.
status is false and an error message is displayed if the entry already exists or one or more of the entries are invalid.

Test case: Regular case
Objective: To verify that restaurant owners can register their business on the platform when entries are valid.
Arrange: Go to the Register page
Act: Type in the information into the corresponding text boxes
Assert: Check that the restaurant is registered and the user is redirected to the login page.

Test case: Entry Already Exists

Objective: To verify that restaurant owners cannot register their business on the platform when the entry already exists.
Arrange: Go to the Register page
Act: Type in information of a restaurant that already exists in the system into the corresponding text boxes.
Assert: Check that the restaurant is not registered, and an error message is displayed.

Test case: One or More Arguments are Invalid
Objective: To verify that restaurant owners cannot register their business on the platform when one or more entries are invalid.
Arrange: Go to the Register page
Act: Type in invalid information into the corresponding text boxes
Assert: Check that the restaurant is not registered, and an error message is displayed.

Add to menu

Verify that restaurant owners can add menu entries.
Associating test file: **owner_menu_add.json**

Input:
The testing function will take in 4 arguments: item, price, description, image.
Output:
status is true and a success message is displayed if the entry is added.
status is false and an error message is displayed if the entry already exists.

Test case: Item Does Not Already Exist
Objective: Verify that restaurant owners can add menu entries.
Arrange: On a restaurant owner account, go to the Menu page, click on the Add button.
Act: Fill in required information to add an item.
Assert: Verify that the menu item has been added and a success message is displayed.

Test case: Item Already Exists
Objective: Verify that restaurant owners cannot add menu entries that already exist.
Arrange: On a restaurant owner account, go to the Menu page, click on the Add button. Fill in the required information to add an item.
Act: Select the “Add” button again, and fill in the same information.

Assert: Check that the menu entry is not added successfully, the existing item is still on the menu, and an error message is displayed.

Remove From Menu

Verify that restaurant owners can remove menu entries
Associating test file: **owner_menu_remove.json**

Input:
The testing function will take in 1 argument: item
Output:
status is true and a success message is displayed if the selected item has successfully been removed. Otherwise, status is false and an error message is displayed.

Test case: Item exists
Objective: Verify that restaurant owners can remove menu entries if the item exists
Arrange: On a restaurant owner account, go to the Menu page, then click on the Remove button.
Act: Type in the name of the food the owner wish to take down
Assert: Selected item is removed from the menu, and a success message is displayed.

Test case: Item does not exist
Objective: Verify that items cannot be removed if they do not exist.
Arrange: On a restaurant owner account, go to the Menu page, then click on the Remove button.
Act: With some form of desynchronisation with the platform, type in the name of the food the owner wish to take down
Assert: An error message is displayed.

Create Category

Allows restaurant owners to place items in categories for easier searching.
Associating test file: **owner_category_add.json**

Input:
The testing function will take in one argument: category
Output:
status is true and a success message is displayed if the operation is successful. status is false and an error message is displayed if a category already exists, or no category name is provided.

Test Case: Regular case
Objective: To verify that categories can be created.
Arrange: Go to the Menu part, and click on the categorize button.
Act: Enter the new category name in the text box and press Create.
Assert: The new category is created and a success message is displayed

Test Case: Category already exists
Objective: To verify that category names are unique.
Arrange: Go to the Menu part, and click on the categorize button. Create a new category name in the text box and press Create. Click the categorize button again.
Act: Enter the same category name and press Create.
Assert: There is only one category of that name, and an error message is displayed.

Test Case: No Category Name Provided
Objective: To verify that a category cannot be created if no name is provided.
Arrange: Go to the Menu part, and click on the categorize button
Act: Press Create
Assert: The category is not created, and an error message is displayed.

Remove Category

Allows restaurants to remove categories
Associating test file: **owner_category_remove.json**

Input:
This testing function takes in an argument: the category name
Output:
status is true and a success message is displayed if the selected category is removed.
status is false and an error message is displayed if the selected category could not be removed.

Test case: Category Exists
Objective: To verify that categories can be removed.
Arrange: As a restaurant owner, go to the “Menu” page, and select the “Categories” button.
Act: Select the ‘X’ icon beside a category.
Assert: Check that the category has been removed, with a success message, and that all food with that category gets that category removed.

Test case: Category Does Not Exist
Objective: To verify that restaurant owners can only remove categories that exist.
Arrange: As a restaurant owner, go to the “Menu” page, and select the “Categories” button.
Act: With some form of desynchronisation with the platform, select the ‘X’ icon beside a category.
Assert: Check that an error message is displayed, saying that the category to be removed does not exist.

Place Item in Category

Allows restaurant owners to place existing items into existing categories.
Associating test file: **owner_categorize_item.json**

Input:
The function takes in two arguments: item and category
Output:
status is true and a success message is displayed if the selected item has been successfully placed in the selected category.
status is false and an error message is displayed if the item or category does not exist.

Test case: Item and Category Exist
Objective: Verify that existing items can be placed into existing categories.
Arrange: As a restaurant owner, go to the “Menu” page, click on the “Categorize” button
Act: Enter the category name and the food item name into their respective boxes and press Enter
Assert: Check that the selected item is in the selected category.

Test case: Item Does Not Exist
Objective: Verify that nonexistent items cannot be placed into existing categories.
Arrange: As a restaurant owner, go to the “Menu” page, click on the “Categorize” button
Act: Enter the category name and a non-existent food item name into their respective boxes and press Enter
Assert: Check that an error message is displayed, saying that the item does not exist.

Test case: Category Does Not Exist
Objective: Verify that existing items cannot be placed into nonexistent categories.
Arrange: As a restaurant owner, go to the “Menu” page, click on the “Categorize” button

Act: Enter a non-existent category name and a food item name into their respective boxes and press Enter
Assert: Check that an error message is displayed, saying that the category does not exist.

Order Management

Allows restaurant owners to view and accept or reject orders to users (if reject, give back a reason). Accept would also change the status of users' order to Received, and restaurant owners can also change the status of users' order so as users can know.
Associating test file: **owner_order_management.json**

Input:
The testing function will take in 2 arguments: order_id, note
Output:
status is true and a success message is displayed alongside the content of the order if the operation is successful.
status is false and an error message is displayed otherwise.

Test case: Accept Incoming Order
Objective: To verify that incoming orders can be accepted by the restaurant.
Arrange: Go to the list of orders either in the Incoming section in the Order page
Act: Click on an order, accept it (The order is assumed to exist)
Assert: The order is moved to the Accepted Orders section and the owner will be paid, and a success message is displayed.

Test case: Reject Incoming Order
Objective: To verify that incoming orders can be rejected by the restaurant.
Arrange: Go to the list of orders either in the Incoming section in the Order page
Act: Click on an order, reject it + give a message (The order is assumed to exist)
Assert: The order is moved to the Rejected Orders section, and a success message is displayed.

Test case: Reject Accepted Order
Objective: To verify that orders can be rejected after being accepted by the restaurant.
Arrange: Go to the list of orders either in the Accepted Section in the Order page
Act: Click on an order, reject it + give a message (The order is assumed to exist)
Assert: When the rejected order is already accepted, the customer is refunded, the order is moved from Accepted Orders to Rejected Orders, and a success message is displayed.