Restaurant Manager Assistant

Testing Manual

**November 27, 2022**

**Slippery Rock University**

Bradley Smith **bms1040@sru.edu**

Ethan Pasman **eap1008@sru.edu**

# Overview

This document describes in detail how the Restaurant Manager Assistant is to be tested. The types of testing include:

* Human Factor Testing
* Unit Testing
* White-box Testing
* Boundary Analysis Testing
* Regression Testing

# System Requirements

* Microsoft® Windows® 8, 10, 11.
* Eclipse IDE for Enterprise Java and Web Developers - 2022-06

# 3. Assumptions

* It was assumed that Eclipse has already been installed and the Restaurant Management Assistant project has been imported as described in the Install Document.

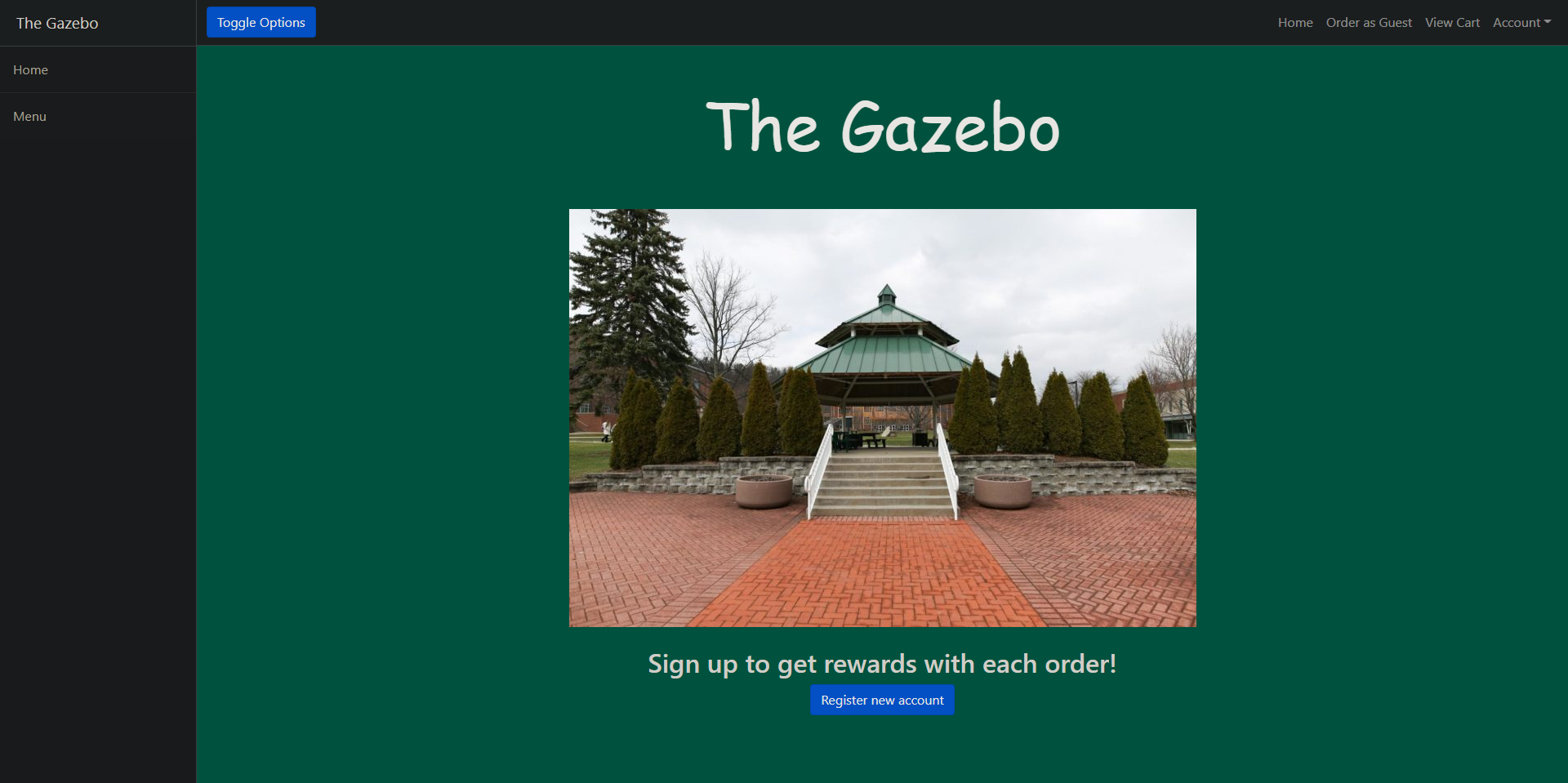
# 4. Navigation

* To assist in navigating this document, Headers have been made for each major topic of the document.
  1. Ex. ‘Unit Testing’ header for appropriate information on test classes and methods written for the program.

# 5. Human Factor Testing

## Human Factor testing is done to gauge how the program will actually be used by humans. During this testing, we ensure that the program does not fail through interaction or lack thereof with any GUI elements.

# Guest Users



## Start the Restaurant Manager Application.

* Click the “Toggle Options” button.
* Note the sidebar correctly moving in and out of view upon pressing it.
* Press the “Menu” element in the sidebar.
* Note that a page displaying the menu is shown, and that each item’s properties, Name, Entree, Side, and Price, have loaded correctly.
* Press the “Return Home” button on the bottom of the page, the “Home” element on the sidebar, or the “Home” hyperlink on the top navigation bar.
* Note that the home page is displayed.
* Press the “Account” drop down menu on the top navigation bar.
* Press the “Contact Us” element.
* Note that the contact page is displayed.

# Guest Ordering

* Press the “Order as Guest” hyperlink on the top navigation bar.
* Press the “Add” button directly to the right of any menu item.
* Note that the item’s quantity property has increased by 1.
* Press the “Remove” button to the right of any menu item.
* Note that the item’s quantity property has decreased by 1, unless it was previously 0, in which note that its quantity stays at 0.
* Press the “View Cart” button on the bottom of the page or the “View Cart” hyperlink in the top navigation bar.
* Note that the cart page is displayed with only the items and quantities interacted with on the order page.
* Note that a “Sales Tax” and “Total” are shown under the cart Items, or only a “Total” if there are no items in the cart, and that the total price is correct. Note that the Sales Tax and Total are unable to be removed from cart, but all menu items are able to be.
* Press the “Remove from cart” hyperlink directly to the right of any item in your cart.
* Note that the item is removed from the cart, as well as its quantity getting set to 0 on the order page. If all items are removed, note that the Sales Tax is automatically removed as well.
* Press the “Add more items to order” button below the cart.
* Note that the order page is displayed properly.
* Press the “View Cart” button on the bottom of the page or the “View Cart” hyperlink in the top navigation bar.
* Press the “Checkout” button at the bottom of the page.
* Note that the checkout page is displayed properly. The default payment method selected should be “Pay with card”.
* Note that the Card Number, Security Code, and Billing ZIP Code fields only accept numbers.
* Note that the “Submit Payment” button does not allow the payment to be submitted if any field is left blank.
* Press the “Pay with Paypal” radio button.
* Note that the form switches to Paypal information properly.
* Note that the “Submit Payment” button does not allow the payment to be submitted if any field is left blank.
* Fill out each field and press the “Submit Payment” button.
* Note that the order success page is displayed. Note that the “Order again” button redirects to the order page and the “Home” button redirects to the home page.

# User Registration

* Press the “Register new account” button at the bottom of the home page.
* Note that the “Cancel Registration” button at the bottom of the page redirects to the home page. Note that pressing the “Add Customer” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Press the “Sign up for rewards?” checkbox to allow testing with rewards.
* Fill out each field, ensuring to remember the email and password fields, as they are what is used to login to the system.
* Press the “Add Customer” button.
* Note that the user is automatically redirected to the login page.

**Logging in**

* Note that pressing the “Sign in” button does not allow logging in if either field is blank.
* Enter an incorrect value in the “Username” field, and then press “Sign in”.
* Enter an incorrect value in the “Password” field, and then press “Sign in”.
* Note that it displays “Bad credentials” and prompts the user to login again.
* Enter the correct email and password used in registration to ensure that user registration works.
* Note that pressing “Sign in” will redirect to the home page, but with slight differences, such as no text at the bottom to register, and clicking on the “Account” drop down giving different options.
* Repeat each step listed in the **Guest Users** tests to ensure the user pages work the same while logged in.

# Customer Ordering

* Press the “Quick Order” hyperlink on the top navigation bar.
* Press the “Add” button directly to the right of any menu item.
* Note that the item’s quantity property has increased by 1.
* Press the “Remove” button to the right of any menu item.
* Note that the item’s quantity property has decreased by 1, unless it was previously 0, in which note that its quantity stays at 0.
* Press the “View Cart” button on the bottom of the page or the “View Cart” hyperlink in the top navigation bar.
* Note that the cart page is displayed with only the items and quantities interacted with on the order page.
* Note that a “Redeem rewards” button is present next to the “Add more items to order” button, and that attempting to redeem rewards currently will not do anything.
* Note that a “Sales Tax” and “Total” are shown under the cart Items, or only a “Total” if there are no items in the cart, and that the total price is correct. Note that the Sales Tax and Total are unable to be removed from cart, but all menu items are able to be.
* Press the “Remove from cart” hyperlink directly to the right of any item in your cart.
* Note that the item is removed from the cart, as well as its quantity getting set to 0 on the order page. If all items are removed, note that the Sales Tax is automatically removed as well.
* Press the “Add more items to order” button below the cart.
* Note that the order page is displayed properly.
* Press the “View Cart” button on the bottom of the page or the “View Cart” hyperlink in the top navigation bar.
* Add enough items through the order page that the total price of the order will be greater than or equal to $50.00 to test customer rewards.
* Press the “Checkout” button at the bottom of the page.
* Note that the checkout page is displayed properly. The default payment method selected should be “Pay with card”.
* Note that the Card Number, Security Code, and Billing ZIP Code fields only accept numbers.
* Note that the “Submit Payment” button does not allow the payment to be submitted if any field is left blank.
* Press the “Pay with Paypal” radio button.
* Note that the form switches to Paypal information properly.
* Note that the “Submit Payment” button does not allow the payment to be submitted if any field is left blank.
* Fill out each field and press the “Submit Payment” button.
* Note that the order success page is displayed. Note that the “Order again” button redirects to the order page and the “Home” button redirects to the home page.

# Rewards and Customer Account

* Press the “Account” drop down menu on the top navigation bar.
* Press the “View Profile” element.
* Note that the attributes First name, Last name, and Email are correct.
* Note that the rewards attribute will be increased by 5.
* Press the “More info on rewards” button.
* Note that the rewards page displays correctly, with the rewards points available number being the same.
* Press the “Quick Order” hyperlink on the top navigation bar.
* Press the “Add” button directly to the right of any menu item.
* Press the “View Cart” button or “View Cart” hyperlink in the top navigation bar to go to your cart.
* Press the “Redeem rewards points” button.
* Note that a rewards discount was added, and that the price is 10% of the total price, not counting taxes.
* Press the “Add more items to order” button.
* Press the “Add” button directly to the right of any menu item.
* Press the “View Cart” button or “View Cart” hyperlink in the top navigation bar to go to your cart.
* Note that the rewards price is updated with the new change in total.
* Press the “Remove from cart” hyperlink to the right of any menu item.
* Note that the rewards price is updated again with the new change in total.
* Press the “Account” drop down menu on the top navigation bar.
* Press the “View Profile” element.
* Press the “View order history” button.
* Note that the previous orders created are shown.
* Press the “Return to profile” button.
* Press the “Edit profile” hyperlink to the right of the customer information.
* Note that any changes made to any field are updated correctly upon pressing the “Update customer” button.
* Press the “Account” drop down menu on the top navigation bar.
* Press the “Change Password” element.
* Note that any changes made to the password field are updated upon pressing the “Update Password” button.
* Press the “Sign out” button.
* Press the “Log Out” button on the confirmation page.
* Note that the user is redirected to the login page.
* Attempt to login with the new password set to ensure changing password works.

# Servers

* Log into a server account.
* Note that the home page upon logging in is the server page, with Outgoing orders and the current menu.
* Press the “Clock In” button at the top of the page.
* Note that the button changes to a “Clock Out” button.
* Press the “View info and rewards” hyperlink directly to the right of any customer in the outgoing orders table.
* Note that the customer account info and rewards are correct.
* Press the “Serving Staff Home” element in the sidebar.
* Press the “Delete Order” button near the left side of an order.
* Note that said order is removed from the table.
* Press the “Clock Out” button.
* Note that the button changes again to “Clock In”.

# Restaurant Managers

* Log into a restaurant manager account.
* Note that the home page for Restaurant Managers is the restaurant local action log page.
* Press the “Clock In” button.
* Note that the button changes to a “Clock Out” button.
* Press the “View Serving Staff Pages” element in the sidebar.
* Note that the server pages display accurately. The Orders table will display all orders, even ones marked as completed by servers, and the menu will display all menu items, even ones not available to customers.
* Press the “View Customers” element in the sidebar.
* Note that the customers list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any customer.
* Note that any fields changed will be updated after pressing the “Update Customer” button.
* Press the “Delete” hyperlink to the right of any customer.
* Note that the customer was deleted and removed from the table.
* Press the “View Menu” element in the sidebar.
* Note that the menu list displays properly with accurate information.
* Press the “View Servers” element in the sidebar.
* Note that the servers list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any server.
* Note that any fields changed will be updated after pressing the “Update Server” button.
* Press the “Delete” hyperlink to the right of any server.
* Note that the server was deleted and removed from the table.
* Press the “View Inventory” element in the sidebar.
* Note that the inventory list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any inventory item.
* Note that any fields changed will be updated after pressing the “Update Inventory” button.

# Warehouse Managers

* Log into a warehouse manager account.
* Note that the home page for warehouse managers is the warehouse local action log page.
* Press the “View Shipments” element in the sidebar.
* Note that the local shipments list displays properly with accurate information.
* Press the “Accept” hyperlink to the right of any shipment.
* Note that the shipment’s status changed to “In Progress”.
* Press the “Decline” hyperlink to the right of any shipment.
* Note that the shipment’s status changed to “Declined”.
* Press the “View Inventory” element in the sidebar.
* Note that the local inventory list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any inventory.
* Note that any fields changed will be updated after pressing the “Update Inventory” button.
* Press the “View Employees” element in the sidebar.
* Note that the local employee list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any employee.
* Note that any fields changed will be updated after pressing the “Update employee” button.

# Restaurant Admins

* Log into a restaurant admin account.
* Note that the home page for Restaurant Admins is the restaurant local action log page.
* Press the “View Customers” element in the sidebar.
* Note that the local customers list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any customer.
* Note that any fields changed will be updated after pressing the “Update Customer” button.
* Press the “Delete” hyperlink to the right of any customer.
* Note that the customer was deleted and removed from the table.
* Press the “Add a new customer” button.
* Note that pressing the “Add Customer” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Fill out each field and press the “Add Customer” button.
* Note that the new customer is added to the customers table.
* Press the “View Managers” element in the sidebar.
* Note that the local managers list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any manager.
* Note that any fields changed will be updated after pressing the “Update Manager” button.
* Press the “Delete” hyperlink to the right of any manager.
* Note that the manager was deleted and removed from the table.
* Press the “Add a new manager” button.
* Note that pressing the “Add Manager” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Change the restaurant location field to the same location as the logged in admin account.
* Fill out each field and press the “Add Manager” button.
* Note that the new manager is added to the managers table.
* Press the “View Servers” element in the sidebar.
* Note that the local servers list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any server.
* Note that any fields changed will be updated after pressing the “Update Server” button.
* Press the “Delete” hyperlink to the right of any server.
* Note that the server was deleted and removed from the table.
* Press the “Add a new server” button.
* Note that pressing the “Add Server” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Change the restaurant location field to the same location as the logged in admin account.
* Fill out each field and press the “Add Server” button.
* Note that the new server is added to the servers table.
* Press the “View Employees” element in the sidebar.
* Note that the warehouse employees list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any employee.
* Note that any fields changed will be updated after pressing the “Update Employee” button.
* Press the “Delete” hyperlink to the right of any employee.
* Note that the employee was deleted and removed from the table.
* Press the “Add a new employee” button.
* Note that pressing the “Add Employee” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Change the warehouse location field to the same warehouse as the logged in admin account.
* Fill out each field and press the “Add Employee” button.
* Note that the new employee is added to the employees table.

# HQ Managers

* Log into an HQ manager account.
* Note that the home page for HQ Managers is the full action log page.
* Press the “View Restaurant Managers” element in the sidebar.
* Note that the managers list displays properly with accurate information. The list of managers is for any restaurant location.
* Press the “Edit” hyperlink to the right of any manager.
* Note that any fields changed will be updated after pressing the “Update Manager” button.
* Press the “Delete” hyperlink to the right of any manager.
* Note that the manager was deleted and removed from the table.
* Press the “Add a new manager” button.
* Note that pressing the “Add Manager” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Fill out each field and press the “Add Manager” button.
* Note that the new manager is added to the managers table.
* Press the “View Warehouse Managers” element in the sidebar.
* Note that the managers list displays properly with accurate information. The list of managers is for any warehouse location.
* Press the “Edit” hyperlink to the right of any manager.
* Note that any fields changed will be updated after pressing the “Update Manager” button.
* Press the “Delete” hyperlink to the right of any manager.
* Note that the manager was deleted and removed from the table.
* Press the “Add a new manager” button.
* Note that pressing the “Add Manager” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Fill out each field and press the “Add Manager” button.
* Note that the new manager is added to the managers table.
* Press the “View Offices” element in the sidebar.
* Note that the office list displays properly with accurate information.
* Press the “View Restaurants” element in the sidebar.
* Note that the restaurant list displays properly with accurate information.
* Press the “View Warehouses” element in the sidebar.
* Note that the warehouse list displays properly with accurate information.

# HQ Admins

* Log into an HQ admin account.
* Note that the home page for HQ Admins is the full action log page.
* Press the “View Administrators” element in the sidebar.
* Note that the admins list displays properly with accurate information. The list of admins is for any restaurant or warehouse location.
* Press the “Edit” hyperlink to the right of any admin.
* Note that any fields changed will be updated after pressing the “Update Admin” button.
* Press the “Delete” hyperlink to the right of any admin.
* Note that the admin was deleted and removed from the table.
* Press the “Add a new Admin” button.
* Note that pressing the “Add Admin” button does not allow registration if all fields are not filled out and that the “Email” field requires an address with text before and after an @ symbol.
* Fill out each field and press the “Add Admin” button.
* Note that the new admin is added to the admin table.
* Press the “View Restaurants” element in the sidebar.
* Note that the restaurant list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any restaurant.
* Note that any fields changed will be updated after pressing the “Update Restaurant” button.
* Press the “Delete” hyperlink to the right of any restaurant.
* Note that the restaurant was deleted and removed from the table.
* Press the “Add a new Restaurant” button.
* Note that pressing the “Add Restaurant” button does not allow registration if all fields are not filled out.
* Fill out each field and press the “Add Restaurant” button.
* Note that the new restaurant is added to the admin table.
* Press the “View Warehouses” element in the sidebar.
* Note that the warehouse list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any warehouse.
* Note that any fields changed will be updated after pressing the “Update Warehouse” button.
* Press the “Delete” hyperlink to the right of any warehouse.
* Note that the warehouse was deleted and removed from the table.
* Press the “Add a new Warehouse” button.
* Note that pressing the “Add Warehouse” button does not allow registration if all fields are not filled out.
* Fill out each field and press the “Add Warehouse” button.
* Note that the new warehouse is added to the warehouse table.
* Press the “View Offices” element in the sidebar.
* Note that the office list displays properly with accurate information.
* Press the “Edit” hyperlink to the right of any office.
* Note that any fields changed will be updated after pressing the “Update Office” button.
* Press the “Delete” hyperlink to the right of any office.
* Note that the office was deleted and removed from the table.
* Press the “Add a new Office” button.
* Note that pressing the “Add Office” button does not allow registration if all fields are not filled out.
* Fill out each field and press the “Add Office” button.
* Note that the new office is added to the offices table.

# 6. Unit Testing

## Unit testing is ensuring the functionality of the code within the project to see any faults or logic errors that may be missed in human factor testing. Our unit testing is done for several classes, including everything within the Billing, Controller, Domain, and Repository packages, as that is where the majority of our code and methods are located. Each method was tested as a test method in JUnit5. Mockito was used to create mocks of the Model and BindingResult used in parameters for some of the methods for the RestaurantControllerTests.

**PaymentDetailsTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in PaymentDetails.
* transferFieldsTest()
  + This method tests the transferFields method in PaymentDetails, which allows copying the fields from one PaymentDetails object to another.
* buildFromFormTest()
  + This method tests the buildFromForm method in PaymentDetails, which allows copying the fields from a PaymentDetails\_Form object onto a PaymentDetails object.

**PaypalTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Paypal class.
* transferFieldsTest()
  + This method tests the transferFields method in Paypal class, which allows copying the fields from one Paypal object to another.
* buildFromFormTest()
  + This method tests the buildFromForm method in Paypal class, which allows copying the fields from a Paypal\_Form object onto a Paypal object.

# RestaurantControllerTests

* load()
  + This method loads the controller’s loaddata method before each test, as sample data to be tested with.
* restaurantControllerConstructorTest()
  + This test method ensures an instance of restaurantController created for use in the test class is instantiated correctly, through the defined constructor.
* getCurrentSessionTest()
  + This test method ensures that the current HttpSession is created.
* getIsLoggedInTest()
  + This test method ensures that the isLoggedIn variable returns the proper value.
* setIsLoggedInTest()
  + This test method ensures that the isLoggedIn variable is set to the proper value.
* checkStringIntFloatTypeTest()
  + This test method ensures that any spreadsheet files read in through the system are parsed properly.
* loadMenuTest()
  + This test method ensures that the menu loads from a spreadsheet into the menu repository.
* loadRestaurantIngredientsTest()
  + This test method ensures that the restaurant inventory loads from a spreadsheet into the inventory repository.
* loadWarehouseIngredientsTest()
  + This test method ensures that the warehouse inventory loads from a spreadsheet into the inventory repository.
* loadDefaultIngredientsTest()
  + This test method ensures that the ingredients for each menu item load from a spreadsheet into the ingredients repository.
* loadTaxesTest()
  + This test method ensures that the tax information for each state loads from a spreadsheet into the stateTax repository.
* loadDataTest()
  + This test method ensures that the sample data created is loaded into each corresponding repository to be used for testing purposes.
* homePageTest()
  + This test method ensures that the home page displays properly.
* addAttributesTest()
  + This test method ensures that the specified attributes are added into a mock model to be displayed from the html pages.
* loginPageTest()
  + This test method ensures that the login page displays properly, and upon authentication will set isLoggedIn to true and redirect to proper home pages.
* loggedInTest()
  + This test method ensures that the customer home page displays properly.
* logoutTest()
  + This test method ensures that logging out page displays properly, and upon confirmation will set isLoggedIn to false and redirect to login.
* getLoggedInUserTest()
  + This test method ensures that the customer account currently authenticated returns the proper value.
* showMenuTest()
  + This test method ensures that the menu page displays properly.
* authorityCheckForLoginRedirectsTest()
  + This test method ensures that the home pages for each role are displayed properly with the corresponding authentications.
* showUpdatePassFormTest()
  + This test method ensures that the update password form displays properly.
* updatePassTest()
  + This test method ensures that the password updated will correctly save to the repository and work in authentication and that it was logged to the action log.
* showCustRegisterFormTest()
  + This test method ensures that the customer registration form displays properly.
* addNewCustTest()
  + This test method ensures that the new customer account is saved to the repository and logged to the action log.
* contactPageTest()
  + This test method ensures that the contact page displays properly.
* infoPageTest()
  + This test method ensures that the customer account information page displays properly.
* orderHistoryPageTest()
  + This test method ensures that the customer’s order history page displays properly.
* userShowUpdateCustFormTest()
  + This test method ensures that the update customer form displays properly.
* showManListTest()
  + This test method ensures that the managers list page displays properly.
* showServerListTest()
  + This test method ensures that the servers list page displays properly.
* showUserListTest()
  + This test method ensures that the customer list page displays properly.
* showAdminListTest()
  + This test method ensures that the admin list page displays properly.
* showOfficesListTest()
  + This test method ensures that the office list page displays properly.
* showRestaurantListTest()
  + This test method ensures that the restaurant list page displays properly.
* showWarehouseListTest()
  + This test method ensures that the warehouse list page displays properly.
* showCustSignUpFormTest()
  + This test method ensures that the add customer form displays properly.
* showServerSignUpFormTest()
  + This test method ensures that the add server form displays properly.
* showEmployeeSignUpFormTest()
  + This test method ensures that the add warehouse employee form displays properly.
* showAdminEmployeesViewTest()
  + This test method ensures that the warehouse employee list page displays properly.
* showAdminShipmentViewTest()
  + This test method ensures that the shipment list page displays properly.
* addEmployeeTest()
  + This test method ensures that the employee added is properly saved to the warehouseEmployee repository and logged to the action log.
* showUpdateEmployeeFormTest()
  + This test method ensures that the update warehouse employee form displays properly.
* updateEmployeeTest()
  + This test method ensures that the warehouse employee is updated within the repository and is logged to the action log.
* showManagerSignUpFormTest()
  + This test method ensures that the add manager form displays properly.
* showAdminSignUpFormTest()
  + This test method ensures that the add admin form displays properly.
* showOfficeSignUpFormTest()
  + This test method ensures that the add office form displays properly.
* showRestaurantSignUpFormTest()
  + This test method ensures that the add restaurant form displays properly.
* showWarehouseSignUpFormTest()
  + This test method ensures that the add warehouse form displays properly.
* addCustTest()
  + This test method ensures that the customer added is properly saved to the customer repository and logged to the action log.
* addServerTest()
  + This test method ensures that the server added is properly saved to the server repository and logged to the action log.
* addManagerTest()
  + This test method ensures that the manager added is properly saved to the manager repository and logged to the action log.
* addAdminTest()
  + This test method ensures that the admin added is properly saved to the admin repository and logged to the action log.
* addOfficeTest()
  + This test method ensures that the office added is properly saved to the office repository and logged to the action log.
* addRestaurantTest()
  + This test method ensures that the restaurant added is properly saved to the restaurant repository and logged to the action log.
* showUpdateAdminFormTest()
  + This test method ensures that the update admin form displays properly.
* showUpdateOfficeFormTest()
  + This test method ensures that the update office form displays properly.
* showUpdateWarehouseFormTest()
  + This test method ensures that the update warehouse form displays properly.
* showUpdateRestaurantFormTest()
  + This test method ensures that the update restaurant form displays properly.
* showUpdateCustFormTest()
  + This test method ensures that the update customer form displays properly.
* showUpdateServerFormTest()
  + This test method ensures that the update server form displays properly.
* showUpdateManagerFormTest()
  + This test method ensures that the update manager form displays properly.
* updateCustTest()
  + This test method ensures that the customer is updated within the repository and is logged to the action log.
* updateServerTest()
  + This test method ensures that the server is updated within the repository and is logged to the action log.
* updateManagerTest()
  + This test method ensures that the manager is updated within the repository and is logged to the action log.
* updateAdminTest()
  + This test method ensures that the admin is updated within the repository and is logged to the action log.
* updateOfficeTest()
  + This test method ensures that the office is updated within the repository and is logged to the action log.
* updateWarehouseTest()
  + This test method ensures that the warehouse is updated within the repository and is logged to the action log.
* updateRestaurantTest()
  + This test method ensures that the restaurant is updated within the repository and is logged to the action log.
* deleteCustTest()
  + This test method ensures that the customer is deleted from the repository and is logged to the action log.
* deleteServerTest()
  + This test method ensures that the server is deleted from the repository and is logged to the action log.
* deleteManagerTest()
  + This test method ensures that the manager is deleted from the repository and is logged to the action log.
* deleteRestaurantTest()
  + This test method ensures that the restaurant is deleted from the repository and is logged to the action log.
* deleteWarehouseTest()
  + This test method ensures that the warehouse is deleted from the repository and is logged to the action log.
* deleteOrderTest()
  + This test method ensures that the order status is set to completed and is logged to the action log.
* showServerViewTest()
  + This test method ensures that the server home page displays properly.
* clockInTest()
  + This test method ensures that the clock in button functions as expected.
* clockOutTest()
  + This test method ensures that the clock out button functions as expected.
* resetWeeklyHoursTest()
  + This test method ensures that the method to reset each employee’s weekly hours to 0 works properly.
* payCalcTest()
  + This test method ensures that the proper amount of money is removed from the restaurant’s profits after paying each employee.
* showLocalManServerViewTest()
  + This test method ensures that the manager server home page displays properly.
* showCustInfoTest()
  + This test method ensures that the customer information page displays properly.
* blankCustInfoTest()
  + This test method ensures that the customer information page does not give errors for invalid customer accounts.
* showLogTest()
  + This test method ensures that the manager home page displays properly.
* showAdminLogTest()
  + This test method ensures that the admin home page displays properly.
* showHQLogTest()
  + This test method ensures that the HQ manager home page displays properly.
* showHQAdminLogTest()
  + This test method ensures that the HQ admin home page displays properly.
* showWarehouseShipmentsTest()
  + This test method ensures that the shipments list page displays properly.
* warehouseManAcceptShipmentTest()
  + This test method ensures that warehouse managers are able to change the status of shipments to in progress.
* warehouseManDenyShipmentTest()
  + This test method ensures that warehouse managers are able to change the status of shipments to declined.
* showWarehouseLogTest()
  + This test method ensures that the warehouse manager home page displays properly.
* showWarehouseInventoryTest()
  + This test method ensures that the inventory list page displays properly.
* showWarehouseEmployeesTest()
  + This test method ensures that the warehouse employee list page displays properly.
* warehouseManShowUpdateEmployeeTest()
  + This test method ensures that the warehouse employee update form displays properly.
* showInventoryViewTest()
  + This test method ensures that the restaurant inventory list page displays properly.
* showRestViewShippingTest()
  + This test method ensures that the shipping page displays properly.
* localManShowUserListTest()
  + This test method ensures that the customer list page displays properly.
* localManShowUpdateCustFormTest()
  + This test method ensures that the update customer form displays properly.
* localManShowUpdateInventoryFormTest()
  + This test method ensures that the update inventory form displays properly.
* localManUpdateInventoryTest()
  + This test method ensures that the restaurant inventory is updated within the repository and is logged to the action log.
* localManUpdateCustTest()
  + This test method ensures that the customer is updated within the repository and is logged to the action log.
* localManDeleteCustTest()
  + This test method ensures that the customer is deleted from the repository and is logged to the action log.
* localManShowMenuTest()
  + This test method ensures that the menu list page displays properly.
* localManShowServersTest()
  + This test method ensures that the server list page displays properly.
* showLocalManUpdateServerFormTest()
  + This test method ensures that the update server form displays properly.
* localManUpdateServerTest()
  + This test method ensures that the server is updated within the repository and is logged to the action log.
* localManDeleteServerTest()
  + This test method ensures that the server is deleted from the repository and is logged to the action log.
* hqManShowManagersTest()
  + This test method ensures that the manager list page displays properly.
* showHQManUpdateManagerFormTest()
  + This test method ensures that the update manager form displays properly.
* hqManUpdateManagerTest()
  + This test method ensures that the manager is updated within the repository and is logged to the action log.
* hqManDeleteManagerTest()
  + This test method ensures that the manager is deleted from the repository and is logged to the action log.
* showLFManagerAddFormTest()
  + This test method ensures that the add manager form displays properly.
* showHQManUpdateWHManagerFormTest()
  + This test method ensures that the update warehouse manager form displays properly.
* hqManUpdateWHManagerTest()
  + This test method ensures that the warehouse manager is updated within the repository and is logged to the action log.
* showWHManagerAddFormTest()
  + This test method ensures that the add warehouse manager form displays properly.
* hqManShowWHManagersTest()
  + This test method ensures that the warehouse manager list page displays properly.
* addLFManagerTest()
  + This test method ensures that the manager added is properly saved to the manager repository and logged to the action log.
* addWHManagerTest()
  + This test method ensures that the warehouse manager added is properly saved to the manager repository and logged to the action log.
* hqManShowRestaurantsTest()
  + This test method ensures that the restaurant list page displays properly.
* hqManShowOfficesTest()
  + This test method ensures that the office list page displays properly.
* hqManShowWarehousesTest()
  + This test method ensures that the warehouse list page displays properly.
* showOrderTypeTest()
  + This test method ensures that the order page displays properly.
* showOrderSuccessTest()
  + This test method ensures that the order success page displays properly.
* showPaymentPageTest()
  + This test method ensures that the payment page displays properly.
* viewCartTest()
  + This test method ensures that the cart page displays properly.
* addToSalesTest()
  + This test method ensures that the method adding an order’s price to its corresponding restaurant’s sales and profits works as intended.
* deleteCartItemTest()
  + This test method ensures that deleting a cart item from the cart page works properly.
* editCartFromMenuTest()
  + This test method ensures that deleting a cart item from the order page works properly.
* deleteCartItemsTest()
  + This test method ensures that all cart items are removed from a customer’s cart after the order is placed.
* custAddToOrderTest()
  + This test method ensures that adding a cart item from the order page works properly.
* createNewOrderTest()
  + This test method ensures that a new order is created properly when starting a new cart.
* custAddOrderTest()
  + This test method ensures that the order should be correctly populated with the customer’s cart items.
* removeFromInventoryTest()
  + This test method ensures that the ingredients for each menu item in an order are removed from the corresponding restaurant’s inventory after an order is completed.
* custRewardsInfoTest()
  + This test method ensures that the rewards info page displays properly.
* getGuestCustTest()
  + This test method ensures that the guest customer for unauthenticated users is returned correctly.
* getUserLocationTest()
  + This test method ensures that the authenticated customer’s location attribute is returned correctly.
* getUserUIDTest()
  + This test method ensures that the authenticated customer’s user ID is returned correctly.

**AdminsTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Admins.

**CartItemsTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in CartItems.

**CustomersTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Customers.

**IngredientsTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Ingredients.

**InventoryTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Inventory.

**LogTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Log.

**ManagersTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Managers.

**MenuTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Menu.

**OfficesTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Offices.

**OrdersTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Orders.
* RemoveMenuItemTest()
  + This method tests the removeMenuItem method in Orders, which allows removing a menu object from the order's list of menu items.

**PaymentDetails\_FormTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in PaymentDetails\_Form.

**Paypal\_FormTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Paypal\_Form.

**RestaurantsTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Restaurants.

**ServersTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Servers.

**ShipmentTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Shipment.
* addShippingTest()
  + This method tests the addShipping method in Shipment, which allows adding a shipping object to the shipment.

**ShippingTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Shipping.

**StateTaxTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in StateTax.

**WarehouseEmployeesTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in WarehouseEmployees.

**WarehouseManagerTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in WarehouseManager.

**WarehousesTest**

* gettersAndSettersTests()
  + This method tests the getter and setter methods present in Warehouses.

**AdminRepositoryTest**

* findAdminByEmailTest()
  + This method tests the findAdminByEmail method in AdminRepository.

**CartItemsRepositoryTest**

* setup()
  + This method sets up the repositories with sample data necessary for testing.
* findByCustomerTest()
  + This method tests the findByCustomer method in CartItemsRepository.
* findByCustMenuIdTest()
  + This method tests the findByCustMenuId method in CartItemsRepository, which allows finding cartItems specific to both a customer and menu object.

**CustomerRepositoryTest**

* findByEmailTest()
  + This method tests the findByEmail method in CustomerRepository.

**IngredientsRepositoryTest**

* findByMenuItemTest()
  + This method tests the findByMenuItem method in IngredientsRepository.

**InventoryRepositoryTest**

* setup()
  + This method sets up the repositories with sample data necessary for testing.
* findInventoryRestaurantTest()
  + This method tests the findInventoryRestaurant method in InventoryRepository.
* findInventoryWarehouseTest()
  + This method tests the findInventoryWarehouse method in InventoryRepository.

**LogRepositoryTest**

* findByIdTest()
  + This method tests the findById method in LogRepository, which searches by Customer ID.

**ManagerRepositoryTest**

* findByRestaurantTest()
  + This method tests the findByRestaurant method in ManagerRepository.

**OrderRepositoryTest**

* setup()
  + This method sets up the repositories with sample data necessary for testing.
* findOrdersByLocationTest()
  + This method tests the findOrdersByLocation method in OrderRepository.
* findByCustomerIdUnpaidTest()
  + This method tests the findByCustomerIdUnpaid method in OrderRepository, which searches for unpaid orders by customer ID.

**RestaurantRepositoryTest**

* setup()
  + This method sets up the repositories with sample data necessary for testing.
* findByAdminTest()
  + This method tests the findByAdmin method in RestaurantRepository.
* findMissingAdminTest()
  + This method tests the findMissingAdmin method in RestaurantRepository, which searches for restaurants without any admins assigned.

**ServerRepositoryTest**

* findServerLocationTest()
  + This method tests the findServerLocation method in ServerRepository.

**ShippingRepositoryTest**

* setup()
  + This method sets up the repositories with sample data necessary for testing.
* findRestaurantShipmentTest()
  + This method tests the findRestaurantShipment method in ShippingRepository.
* findWarehouseManShipmentTest()
  + This method tests the findWarehouseManShipment method in ShippingRepository, which searches by warehouse ID in descending order
* findAdminShipmentTest()
  + This method tests the findAdminShipment method in ShippingRepository, which searches by warehouse ID

**StateTaxRepositoryTest**

* findByStateTest()
  + This method tests the findByState method in StateTaxRepository.

**WarehouseEmployeeRepositoryTest**

* setup()
  + This method sets up the repositories with sample data necessary for testing.
* findByWarehouseListTest()
  + This method tests the findByWarehouse method in WarehouseEmployeeRepository, with a list of warehouses as the parameter.
* findByWarehouseTest()
  + This method tests the findByWarehouse method in WarehouseEmployeeRepository, with a warehouse ID as the parameter.

**WarehouseManagerRepositoryTest**

* findByWarehouseTest()
  + This method tests the findByWarehouse method in WarehouseManagerRepository.

**WarehouseRepositoryTest**

* findByAdminTest()
  + This method tests the findByAdmin method in WarehouseRepository.

# 7. White-box Testing

* White-box testing is used for ensuring the integrity of the system from a code perspective, ensuring inputs and outputs are as expected and cannot lead to faults or errors.

# User Interface Testing

* The user interface is tested extensively by pushing buttons randomly to see if the system can withstand any combination of input on any page without crashing.

# Input File Testing

* RestaurantControllerTests.checkStringIntFloatTypeTest()
  + This test method is used with the file TestDataSpreadsheet.xlsx to properly test the parsing of input files given throughout the program. This tests that the menu, ingredients, and tax information all load into the program correctly and will not give errors with bad input files.

# 8. Boundary Analysis Testing

* Boundary Analysis is important to make sure our tests are not underfitting for real world data. Testing for different boundaries throughout the system is done to simulate an environment which the system may be used in.

# Boundary Analysis

* For User Interface interactions that may display or work with multiple entries, such as tables or information pages, testing is done for zero, one, or multiple of each entry to be displayed.

# 9. Regression Testing

* Regression testing is a necessary part of testing for when code is changed after tests have been done. Proper regression testing ensures that all parts of the system work as expected even after new parts are introduced or old parts are removed.

# Changes to RestaurantController

* For all changes to restaurantController, the restaurantControllerTest class will be updated with new test methods for any new methods used, and all unit tests will be run again.