Restaurant Manager Assistant

Final Report

**November 30, 2022**

**Slippery Rock University**

Bradley Smith **bms1040@sru.edu**

Ethan Pasman **eap1008@sru.edu**

# References

# Billing

* All Billing code (classes within the edu.sru.group7.restaurantmanager.billing package, as well as the classes PaymentDetails\_Form and Paypal\_Form in the domain package, and the classes PaymentDetailsRepository and PaypalRepository in the repository package), were provided by Cody Kuntz’s WebShopping project.
* Kuntz, C. et al. (24 October, 2022) BillingCode (Version 1) [Source Code]

# Authentication and Security

* All security code (classes within the edu.sru.group7.restaurantmanager.authentication and edu.sru.group7.restaurantmanager.security packages) was written with help from *AmigosCode*’s YouTube tutorial “Spring Security | FULL COURSE”.
* Nelson, M. S. B. (18 December, 2019) Spring Security | FULL COURSE [Tutorial Video] https://youtu.be/her\_7pa0vrg

# CSS Theme

* The CSS Theme used in this project (styles.css and scripts.js within the project) was provided by a free bootstrap website template from *ThemeWagon.*
* ThemeWagon. (August 2022) Elearning - Free Bootstrap 5 CSS3 Education Website Template (Version 1.0.0) [Source Code] https://themewagon.com/themes/elearning-free-bootstrap-5-css3-education-website-template/

# Entity Relationships

* Entity relationships connecting our database within the system, specifically One to Many and Many to Many connections, were written with help from *Code Java*’s YouTube tutorials “Spring Boot One To Many Tutorial with Thymeleaf, Bootstrap and MySQL Database” and “Spring Boot Many to Many Tutorial with Thymeleaf, Bootstrap and MySQL”.
* Minh, N. H. (16 December 2020) Spring Boot One To Many Tutorial with Thymeleaf, Bootstrap and MySQL Database [Tutorial Video] https://www.youtube.com/watch?v=ctwRpskAeIU
* Minh, N. H. (21 December 2020) Spring Boot Many To Many Tutorial with Thymeleaf, Bootstrap and MySQL Database [Tutorial Video] https://www.youtube.com/watch?v=AbGxZmfWulU

# Completion

* We intended to finish warehouse employee pages, further unit testing, and a subsystem for staff scheduling, but did not have time to implement those features. Other than those minor things, the system is complete and fully working to the best of our knowledge.

# Contribution

* Other than the code referenced in the **References** section of this report, all other code and documentation was contributed entirely by Bradley Smith and Ethan Pasman.

# Problem Explanation

* The Restaurant Manager Assistant program is meant to solve the problem of automating various tasks in a restaurant environment.
* Working in a restaurant is a high stress environment for management and staff, and a website adding as much automation as possible would be a good program to help save time and energy when working, as well as provide a simple user interface for both employees and customers.
* We allow customers to order and pay online, serving staff to view and work with all outgoing orders within their location, managers to keep track of the automated inventory and shipments, administration to manage the staff accounts and information, and more.

# System Requirements

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

# Diagrams

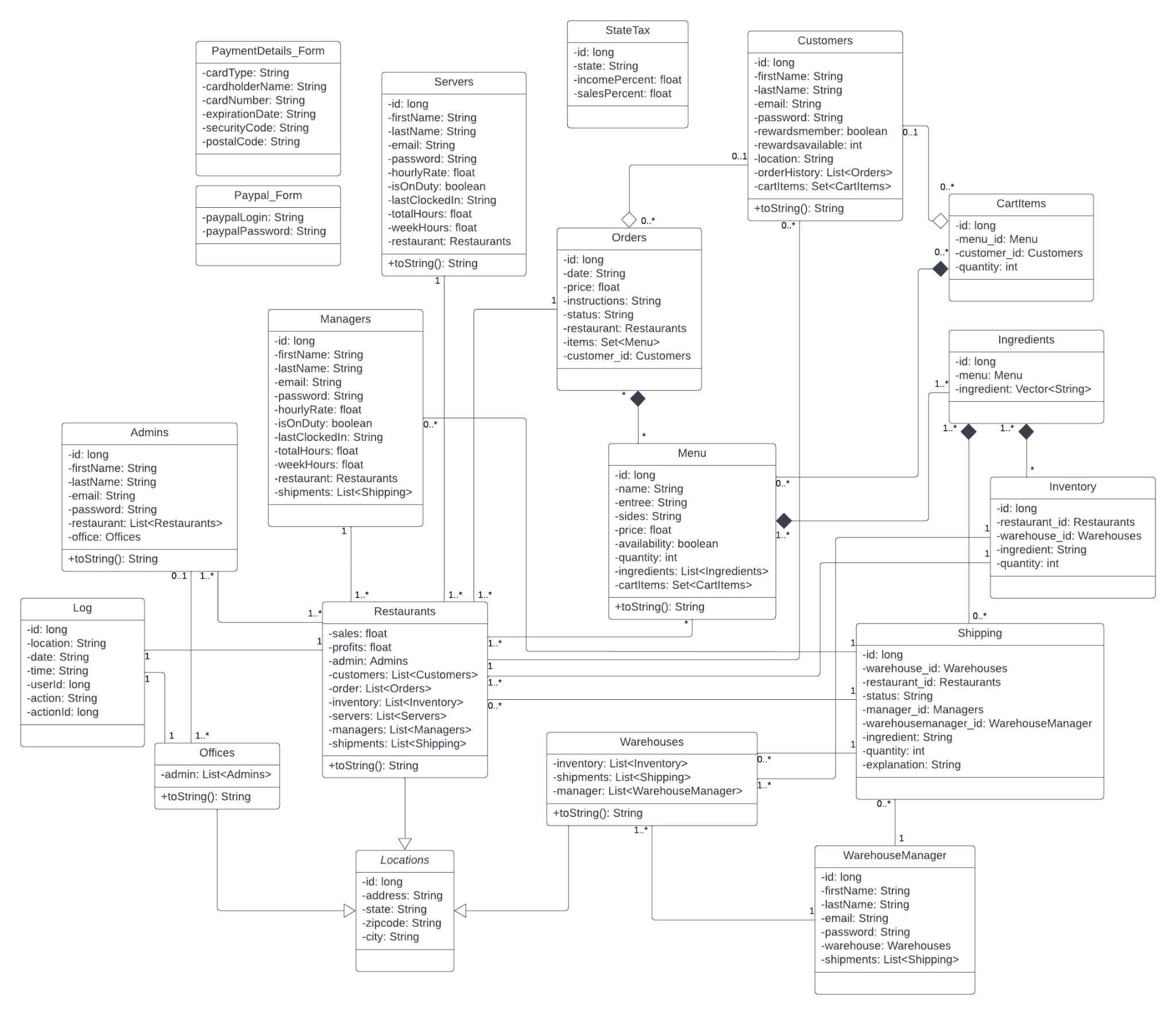


Figure 1 – Domain ER diagram

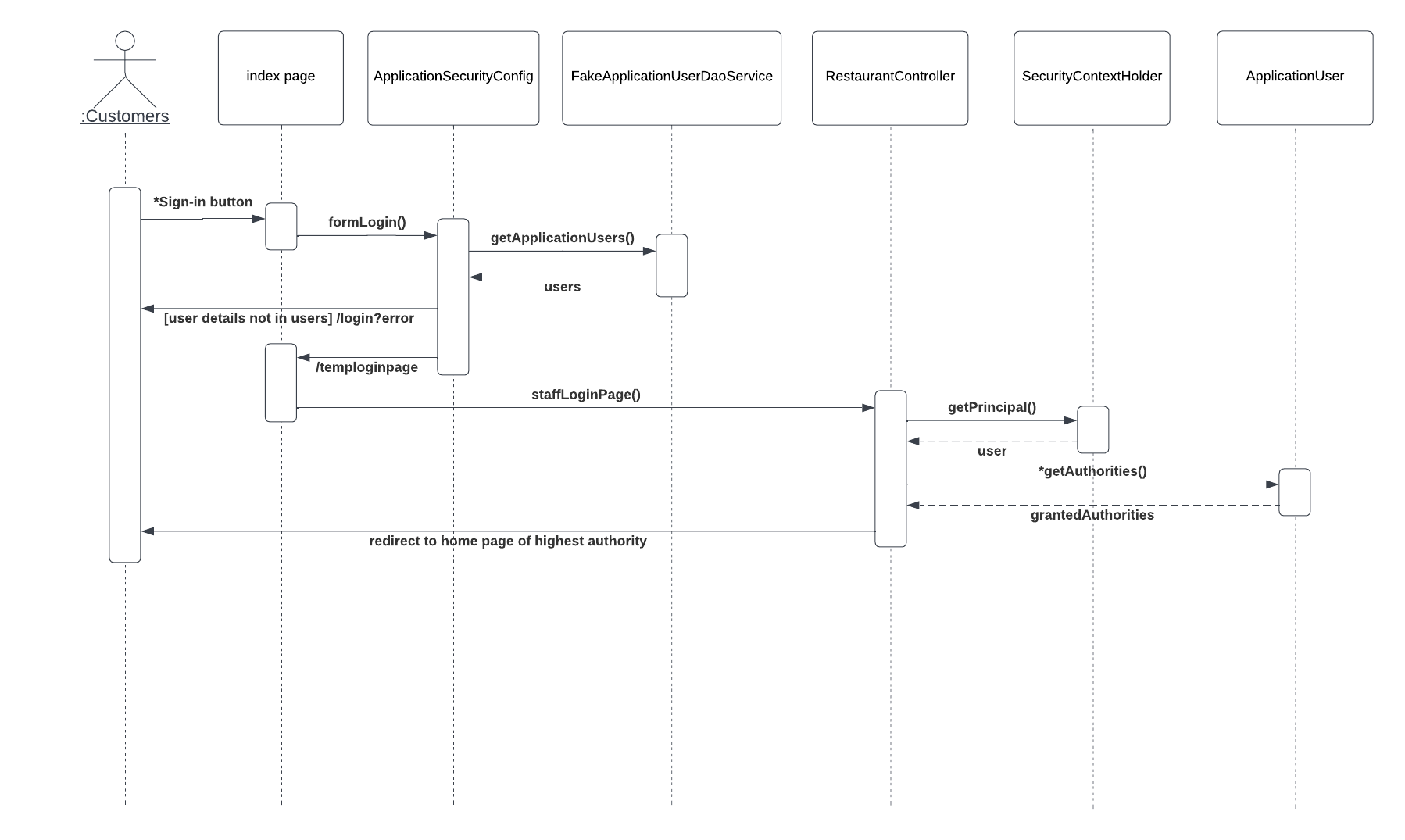


Figure 2 – Customers Sequence Diagram

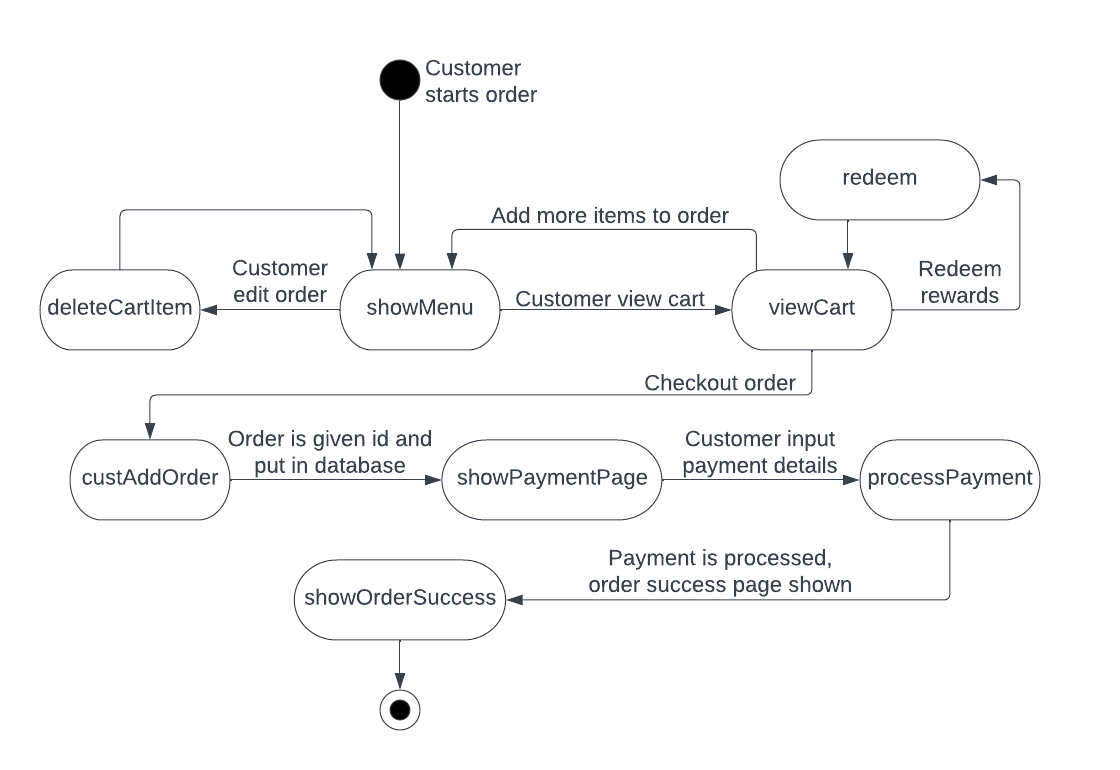


Figure 3 – Order Interaction Activity Diagram

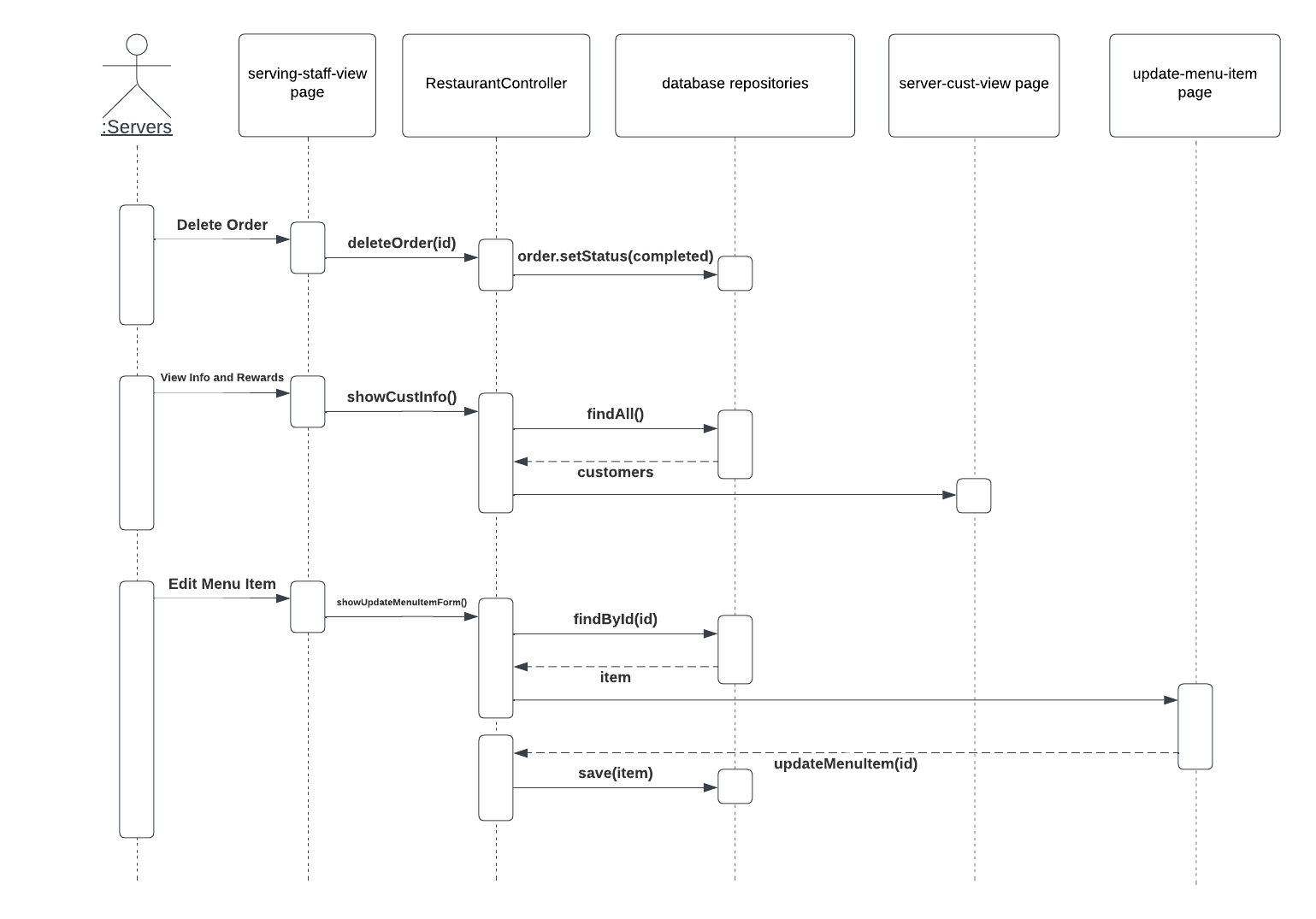


Figure 4 – Servers Sequence Diagram

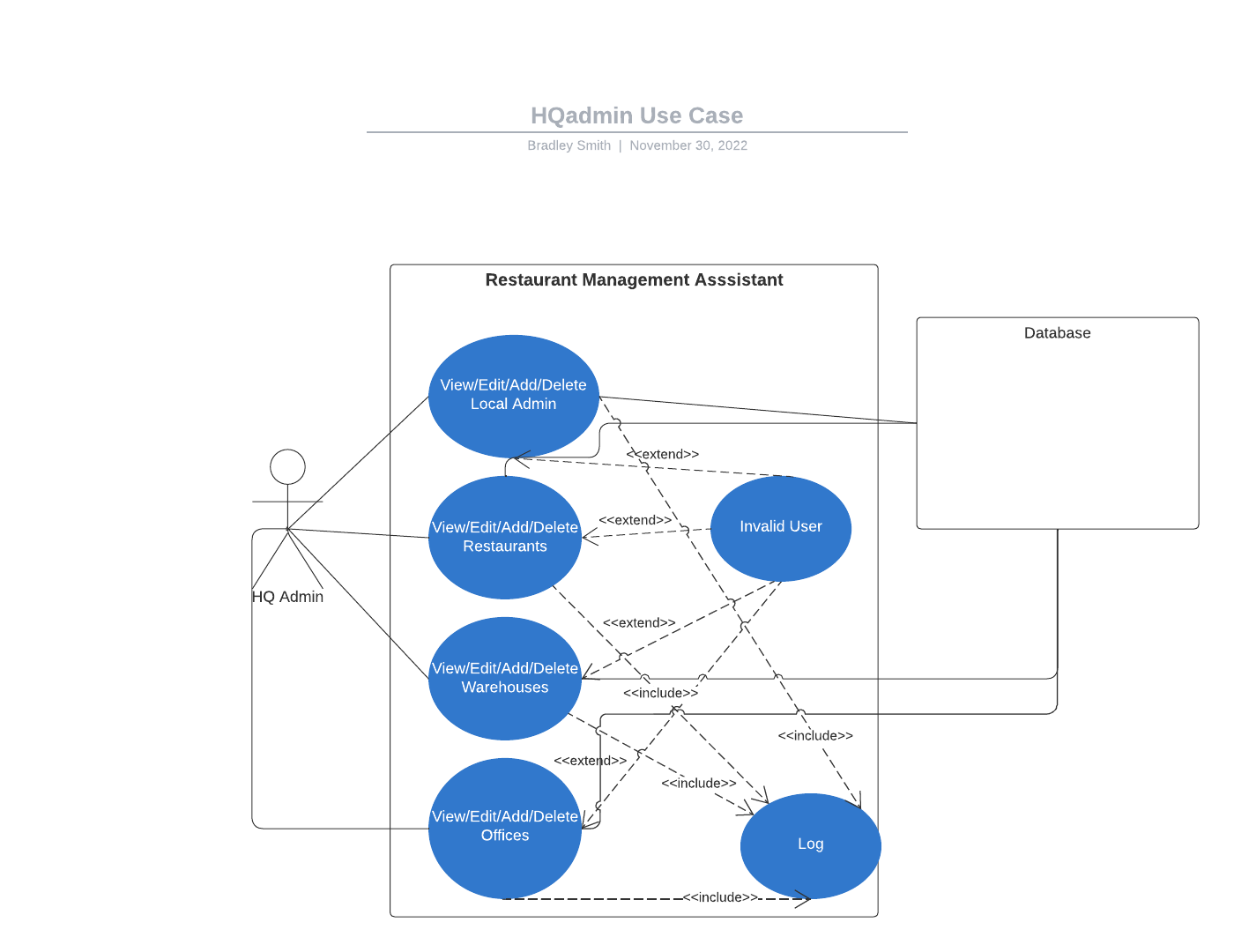


Figure 5 – HQ Admin Use Case Diagram

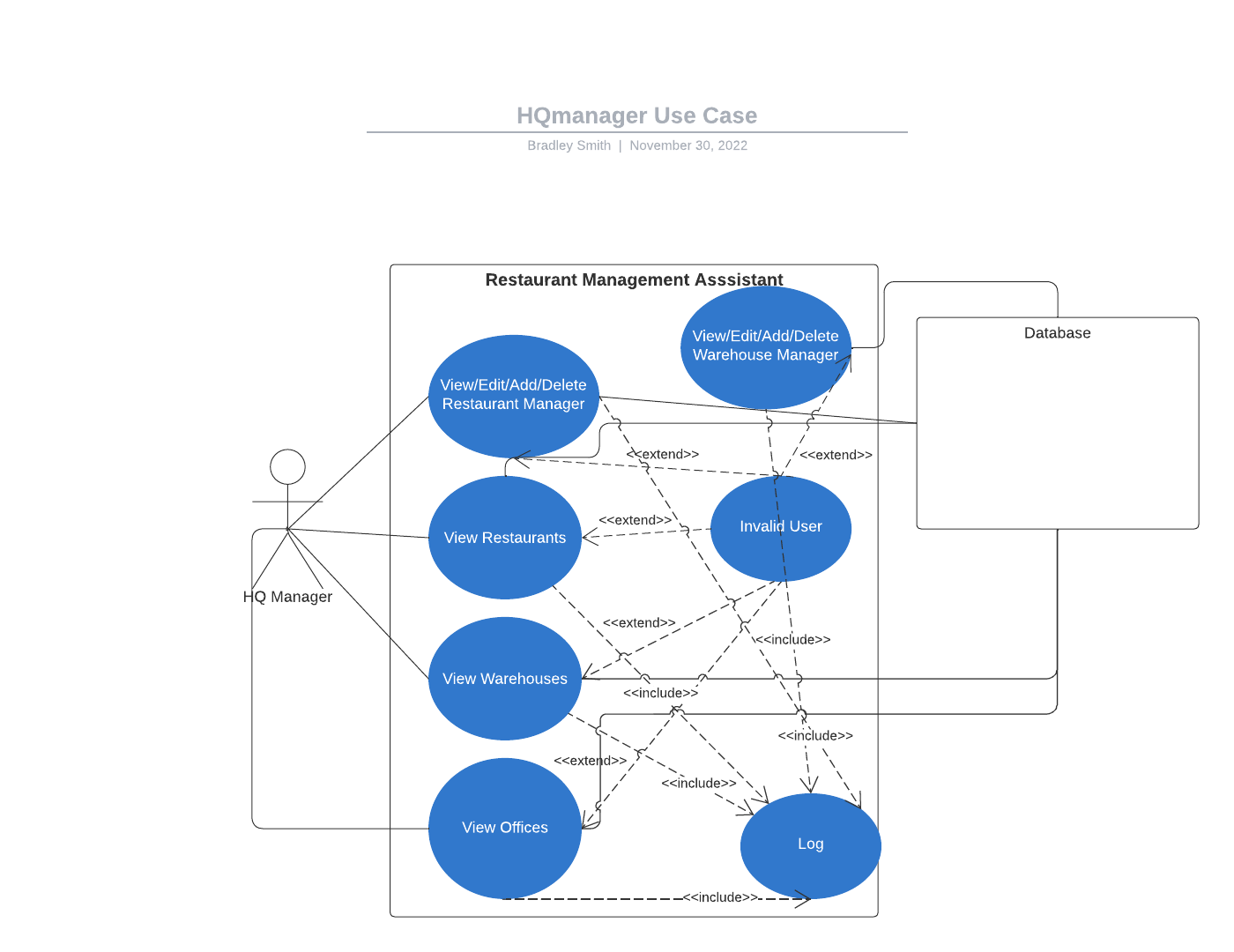


Figure 6 – HQ Manager Use Case Diagram

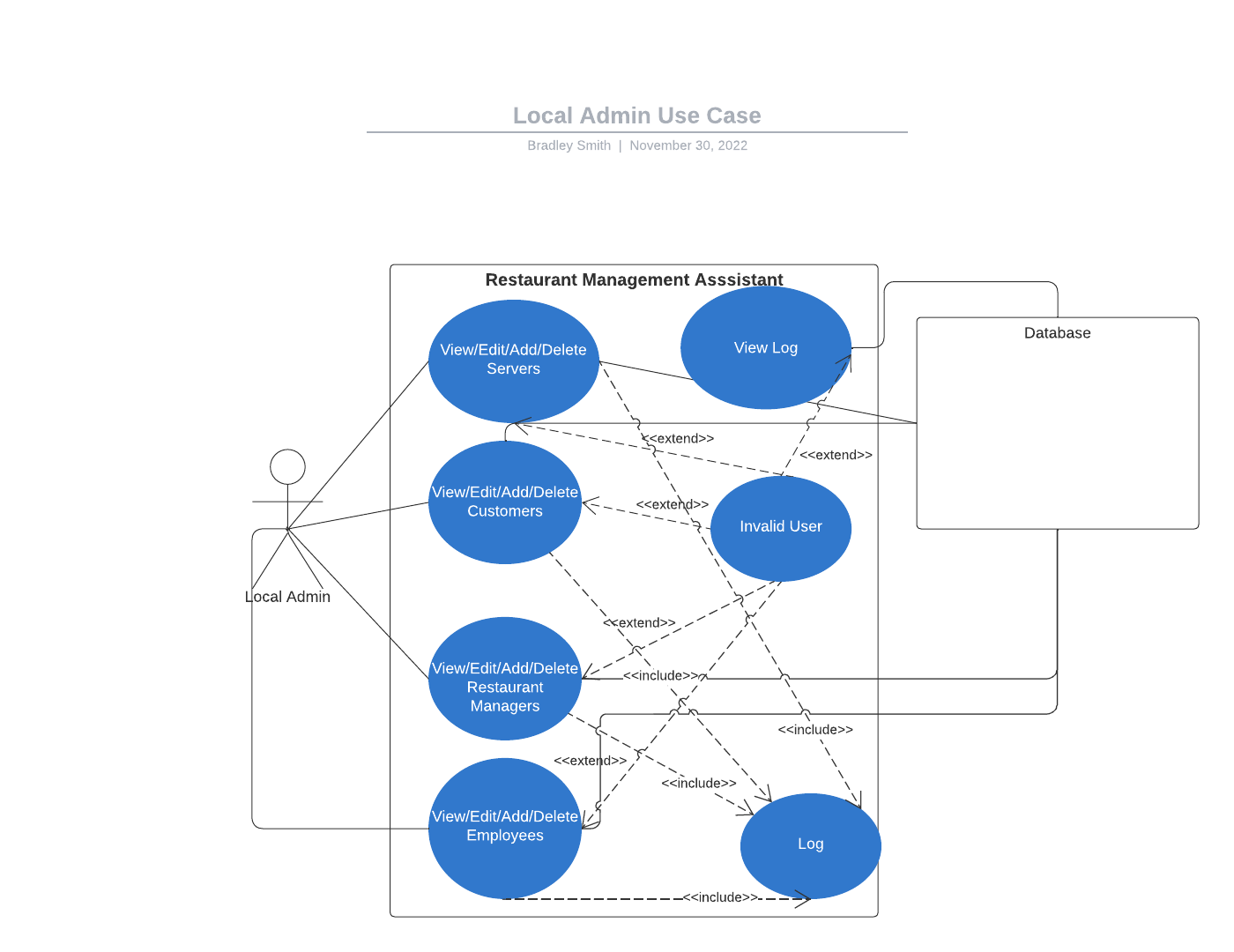


Figure 7 – Local Admin Use Case Diagram

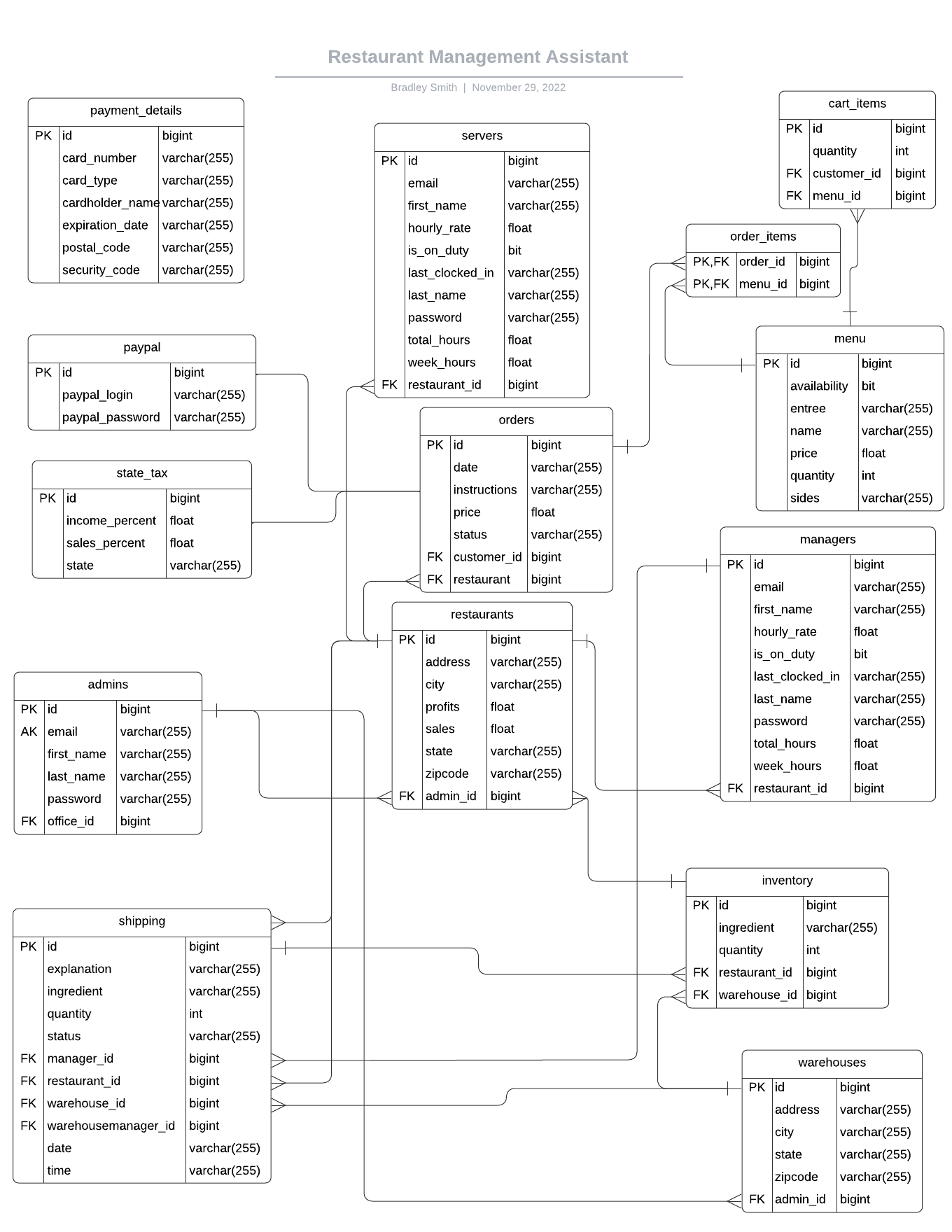


Figure 8 – Restaurant ER Diagram

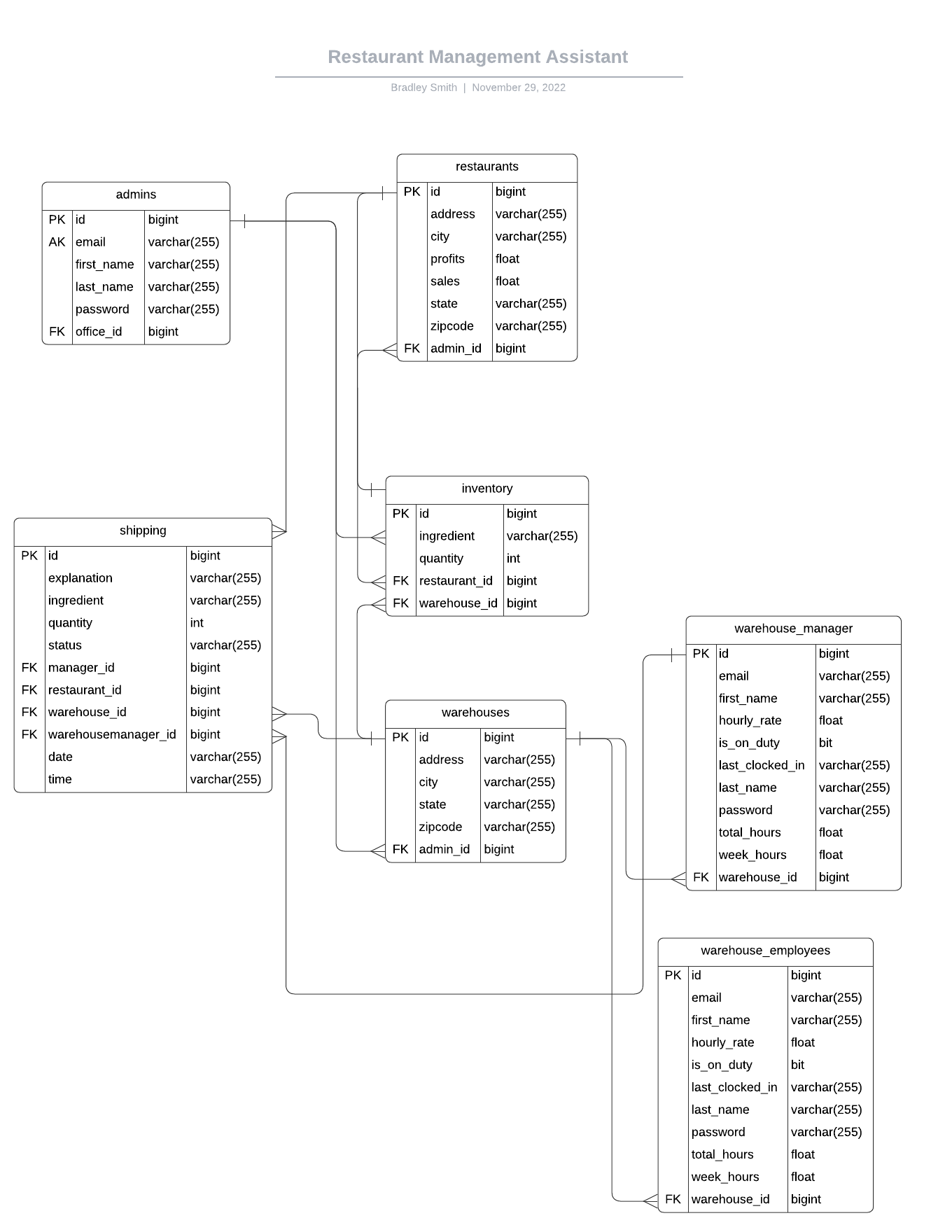


Figure 9 – Warehouse ER Diagram

# Caveats / Minefields

# Low Amount of Unit Testing and Integration Testing

## One of the problems in the system is the relatively low amount of unit testing done for the amount of code in the project.

## Despite the fact that the majority of the system’s code and methods are contained within the RestaurantManagerController class, the fact that this class was the only one tested by a unit test is something that can be improved upon with more thorough testing.

* Some of the methods in the controller test class have been commented out due to an error that is thrown only from the test methods, and due to time constraints we have not been able to figure out how to fix said error. Each method that had been commented out was tested thoroughly through human factor testing, however.

# System Optimization

## Another of the problems we were unable to get to fully is system optimization.

## Currently, there are a lot of methods in our controller class that do very similar things or display very similar pages, but subtle differences, such as the navigation bar needing to be different for logged in users or forms to update database entries being slightly different depending on the user’s role in the program, have left us with needing to create different pages and methods for each specific thing we wanted to achieve.

* Some of these pages, such as the difference between customer and guest pages, could likely have been done more efficiently by using some more of Thymeleaf’s features on a single page instead of splitting them into two.
* On a similar note, some functions loop through every entry in a repository’s “findAll” method, instead of creating a new method with the specific query intended. Adding more methods in our repository classes to use the specific queries needed for these methods could optimize the program by getting rid of unnecessary looping.

# Single Controller Class

## A third problem we wanted to fix but didn’t get done is the fact that all of our controller code is loaded through a single class, RestaurantController, instead of organized and split into several different controller classes.

## This problem makes the system seem disorganized when looking through the code, and can make trying to find a specific method more confusing.

# Lack of Payment Gateway

## Currently, the system keeps track of sales and profits for each restaurant under the hood, but nothing is done with this information.

## The same goes for customer payment details taken from the payment page, and the payout amounts counted for each employee as calculated in RestaurantController.payCalc(), as the project is not connected to any payment gateway API, and cannot actually process the transaction by moving money from their account.

* If the project is to be deployed in a real-world environment, connecting the code to a payment gateway will be necessary to actually process payments.

# Documentation

**8.5 Customers**

showOrderType

@RequestMapping("/Customer-ordertype-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showOrderType(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

Order page for customer or guest

viewCart

@RequestMapping("/Customer-cart-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) viewCart(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

Cart page, cart loaded through either Customer account or HttpSession if guest user

deleteCartItem

@GetMapping("/editcart/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteCartItem(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Cart id

model - Remove cartItem from cart through cart-view page

redeemRewards

@RequestMapping("/redeem") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) redeemRewards()

Customer redeem rewards button Redeems 5 rewards points at a time Discount is 10% of order (not including tax) Creates new discount menu object to be added as a cartItem

updateRewards

public void updateRewards()

Updates already redeemed rewards if the customer changes order after redeeming

custRewardsInfo

@GetMapping("/rewardsinfo") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) custRewardsInfo(org.springframework.ui.Model model)

Rewards info page for customers

showMenu

@RequestMapping("/showmenu") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showMenu()

**Returns:**

Menu page. View menu loaded from database or excel file

**8.6 Servers**

showCustInfo

@GetMapping("/serverviewcustinfo/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showCustInfo(@PathVariable("id")

Customers orderCust,

org.springframework.ui.Model model)

**Parameters:**

id - ID of Customer corresponding to order

orderCust - Customer

model -

**Returns:**

server-cust-view. Shows selected customer information based on their email that we Query the Repository for.

**deleteOrder**

@GetMapping("/deleteorder/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteOrder(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Order

model -

**Returns:**

servingstaffview. Updates selected order status to "Completed" from "Paid".

showServerView

@GetMapping("/servingstaffview") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showServerView(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

Serving staff home view

**8.7 Restaurant Managers**

showInventoryView

@RequestMapping("/manager-inventory-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showInventoryView(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

manager-inventory-view. Shows the Inventory page of the current Managers assigned Restaurant

showRestViewShipping

@RequestMapping("/manager-shipment-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showRestViewShipping(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

manager-shipment-view. Shows all shipments for Restaurant Location from logged in Manager.

localManShowUserList

@RequestMapping("/manager-cust-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManShowUserList(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

manager-cust-view. Shows the list of all customers to Local Manager.

localManShowUpdateCustForm

@GetMapping("/localmanagercustedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManShowUpdateCustForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-customer. Shows the selected Customers profile information for Local Manager to

change.

localManShowUpdateInventoryForm

@GetMapping("/managerinventoryedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManShowUpdateInventoryForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-inventory. Shows the selected restaurant

localManUpdateInventory

@PostMapping("/managerinventoryupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManUpdateInventory(@PathVariable("id")

long id,

@Validated

Inventory inventory,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

inventory -

result -

model -

**Returns:**

manager-inventory-view. Shows the Local Managers inventory view. Saves updated Inventory item into repository.

localManUpdateCust

@PostMapping("/localmanagercustupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManUpdateCust(@PathVariable("id")

long id,

@Validated

Customers customer,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

customer -

result -

model -

**Returns:**

update-customer. Updates entered customer information from Local Manager. Validates email does not already exist.

localManDeleteCust

@GetMapping("/localmanagercustdelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManDeleteCust(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

redirect to manager page list of customers

localManShowMenu

@RequestMapping("/manager-menu-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManShowMenu(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

manager-menu-view. Menu page with manager authorities and ability to edit menu manually

localManShowServers

@RequestMapping("/manager-server-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManShowServers(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

manager-server-view. View list of servers

showLocalManUpdateServerForm

@GetMapping("/localmanagerserveredit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showLocalManUpdateServerForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

update-server. Form to update server information

localManUpdateServer

@PostMapping("/localmanagerserverupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManUpdateServer(@PathVariable("id")

long id,

@Validated

Servers server,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id - Server id

server -

result -

model - Updates server information in database

localManDeleteServer

@GetMapping("/localmanagerserverdelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManDeleteServer(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

model - Deletes server information from database

localManAddShipmentForm

@RequestMapping("/manager-add-shipment/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) localManAddShipmentForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

add-shipment. Displays the add shipment form to the current Restaurant Manager. Lists the warehouses inventory items/quantity.

saveShipping

@PostMapping("/add-shipment") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) saveShipping(@ModelAttribute

Shipment shipment,

org.springframework.ui.Model model)

**Parameters:**

shipment - Shipment

model -

**Returns:**

manager-shipment-view. Saves quantities entered in add-shipment and supplmenets additional unentered data.

**8.8 Warehouse Managers**

showWarehouseShipments

@RequestMapping("warehouseman-shipment-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWarehouseShipments(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

warehouseman-shipment-view. Returns selected shipments from the warehouse belonging to currently logged in warehouse manager.

warehouseManAcceptShipment

@GetMapping("/warehouseman-shipment-accept/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) warehouseManAcceptShipment(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Shipping id

model -

**Returns:**

warehouseman-shipment-view. Changes status of currently selected shipment from 'Pending' to 'In Progress'.

warehouseManDenyShipment

@GetMapping("/warehouseman-shipment-deny/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) warehouseManDenyShipment(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Shipping id

model -

**Returns:**

warehouseman-shipment-view. Changes status of currently selected shipment from 'Pending' to 'Declined'.

showWarehouseLog

@RequestMapping("warehouseman-log-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWarehouseLog(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

warehouseman-log-view. Shows the log belonging to that warehouse manager

showWarehouseInventory

@RequestMapping("warehouseman-inventory-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWarehouseInventory(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

warehouseman-inventory-view. Shows the current inventory for the warehouse of the warehousemanager that is logged in.

warehouseManShowUpdateInventory

@GetMapping("/warehouseman-inventory-edit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) warehouseManShowUpdateInventory(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-inventory.Shows the update inventory page for the selected inventory item to the warehouse manager.

warehouseManUpdateInventory

@PostMapping("/warehouseman-inventory-update/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) warehouseManUpdateInventory(@PathVariable("id")

long id,

@Validated

Inventory inventory,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id - Inventory id

inventory -

result -

model -

**Returns:**

warehouseman-inventory-view. Saves data entered by warehouse manager in the update-inventory page

showWarehouseEmployees

@RequestMapping("warehouseman-employees-view")

public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWarehouseEmployees(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

warehouseman-employees-view. Shows all employees belonging to warehouse that the logged in warehouse manager is assigned.

warehouseManShowUpdateEmployee

@GetMapping("/warehouseman-employee-edit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) warehouseManShowUpdateEmployee(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Employee id

model -

**Returns:**

update-employee. Shows update form for selected employee to the warehouse manager.

warehouseManUpdateEmployee

@PostMapping("/warehouseman-employee-update/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) warehouseManUpdateEmployee(@PathVariable("id")

long id,

@Validated

WarehouseEmployees employee,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id - Employee id

employee -

result -

model -

**Returns:**

warehouseman-employees-view. Saves data entered by warehouse manager in update-employee page

**8.9 Local Administrators**

showCustSignUpForm

@RequestMapping("/custsignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showCustSignUpForm(Customers customer)

**Parameters:**

customer -

**Returns:**

add-customer. Shows add customer page to Local Admin

showServerSignUpForm

@RequestMapping("/serversignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showServerSignUpForm(Servers server)

**Parameters:**

server -

**Returns:**

add-server. Shows add server page to Local Admin

showEmployeeSignUpForm

@RequestMapping("/localadmin-employee-signup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showEmployeeSignUpForm(WarehouseEmployees employees)

**Parameters:**

employees - Warehouse Employees

**Returns:**

add-warehouse-employee page. Shows the employee creation form to a local admin

showAdminEmployeesView

@GetMapping("/admin-employee-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showAdminEmployeesView(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

admin-warehouse-employee-view. Shows all employees from every warehouse an Admin is assigned.

showAdminShipmentView

@GetMapping("/admin-shipment-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showAdminShipmentView(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

admin-warehouse-employee-view. Shows all shipments from all warehouses that the localadmin is assigned

addEmployee

@RequestMapping("/addemployee") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addEmployee(@Validated

WarehouseEmployees employee,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

employee - WarehouseEmployee

result -

model -

**Returns:**

admin-employee-view. Adds employee created from local admin

showUpdateEmployeeForm

@GetMapping("/localadminemployeeedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateEmployeeForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - WarehouseEmployee id

model -

**Returns:**

update-warehouse-employee. Shows the update employee page based on employee selected from admin-employee-view

updateEmployee

@PostMapping("/localadminemployeeupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateEmployee(@PathVariable("id")

long id,

@Validated

WarehouseEmployees employee,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id - Warehouse Employee

employee - Warehouse Employee

result -

model -

**Returns:**

admin-employee-view. Saves updated information from update-warehouse-employee.

deleteEmployee

@GetMapping("/localadminemployeedelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteEmployee(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Warehouse Employee

model -

**Returns:**

admin-employee-view. Deletes the warehouse employee selected.

showManagerSignUpForm

@RequestMapping("/mansignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showManagerSignUpForm(Managers manager)

**Parameters:**

manager -

**Returns:**

add-LFmanager. Shows add Local Manager page to Local Admin

addCust

@RequestMapping("/addcustomer") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addCust(@Validated

Customers customers,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

customers -

result -

model -

**Returns:**

admin-cust-view. Saves entered customer information from Local Admin. Verifies that email does not already exist in Customers table.

addServer

@RequestMapping("/addserver") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addServer(@Validated

Servers server,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

server -

result -

model -

**Returns:**

admin-server-view. Saves entered server information from Local Admin. Verifies email does not already exist in Servers table.

addManager

@RequestMapping("/addmanager") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addManager(@Validated

Managers manager,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

manager -

result -

model -

**Returns:**

add-LFmanager. Saves entered manager information from Local Admin. Verifies email does not already exist in Managers table.

showUpdateCustForm

@GetMapping("/localadmincustedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateCustForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-customer. Shows update customer page to Local Admin.

showUpdateServerForm

@GetMapping("/localadminserveredit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateServerForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-server. Shows update server page to Local Admin.

showUpdateManagerForm

@GetMapping("/localadminmanedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateManagerForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-LFmanager. Shows update Local Manager page to Local Admin.

updateCust

@PostMapping("/localadmincustupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateCust(@PathVariable("id")

long id,

@Validated

Customers customer,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

customer -

result -

model -

**Returns:**

admin-cust-view. Saves updated customer information entered by Local Admin. Verifies that email information entered does not exist in Customers table.

updateServer

@PostMapping("/localadminserverupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateServer(@PathVariable("id")

long id,

@Validated

Servers server,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

server -

result -

model -

**Returns:**

admin-server-view. Saves updated server information entered by Local Admin. Verifies that email information entered does not exist in Servers table.

updateManager

@PostMapping("/localadminmanupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateManager(@PathVariable("id")

long id,

@Validated

Managers manager,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

manager -

result -

model -

**Returns:**

admin-manager-view. Saves updated Local Manager information entered by Local Admin. Verifies that email information entered does not exist in Managers table.

addAdminRestaurant

public void addAdminRestaurant(Admins admin)

**Parameters:**

admin - . Grabs all restaurants the admin has been assigned and updates their admin FK.

removeAdminRestaurant

public void removeAdminRestaurant(Admins admin)

**Parameters:**

admin - . Removes admin assigned to restaurant so the restaurant does not get deleted when the admin does.

deleteCust

@GetMapping("/localadmincustdelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteCust(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Customer

model -

**Returns:**

admin-cust-view. Deletes customer from Customers table.

deleteServer

@GetMapping("/localadminserverdelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteServer(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Server

model -

**Returns:**

admin-server-view. Deletes server from Servers table.

deleteManager

@GetMapping("/localadminmandelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteManager(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Manager

model -

**Returns:**

admin-man-view. Deletes manager from Managers table.

**8.10 HQ Manager**

hqManShowManagers

@RequestMapping("/HQmanager-managers-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManShowManagers(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

hqmanager-managers-view. View list of managers

showHQManUpdateManagerForm

@GetMapping("/HQmanagermanedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showHQManUpdateManagerForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Manager id

model -

**Returns:**

update-lfmanager. Form for updating local managers

hqManUpdateManager

@PostMapping("/hqmanagermanupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManUpdateManager(@PathVariable("id")

long id,

@Validated

Managers manager,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id - Manager id

manager -

result -

model - Updates manager information in database

hqManDeleteManager

@GetMapping("/HQmanagermandelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManDeleteManager(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Manager id

model - Deletes manager information from database

showLFManagerAddForm

@RequestMapping("/HQmanageraddmanager") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showLFManagerAddForm(Managers manager)

**Parameters:**

manager -

**Returns:**

add-lfmanager. Form for adding new local managers

showHQManUpdateWHManagerForm

@GetMapping("/hqmanagerwhmanedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showHQManUpdateWHManagerForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - WarehouseManager

model -

**Returns:**

update-manager. Shows the WH manager update form to HQ manager.

hqManUpdateWHManager

@PostMapping("/hqmanagerwhmanupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManUpdateWHManager(@PathVariable("id")

long id,

@Validated

WarehouseManager manager,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id - WarehouseManager

manager -

result -

model -

**Returns:**

HQmanager-WHmanagers-view. Saves data entered by HQmanager in update-WHmanager form.

hqManDeleteWHManager

@GetMapping("/HQmanagerwhmandelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManDeleteWHManager(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - WarehouseManager

model -

**Returns:**

HQ-managers-view. Deletes the currently selected warehouse manager.

showWHManagerAddForm

@RequestMapping("/HQmanageraddWHmanager") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWHManagerAddForm(WarehouseManager manager,

org.springframework.ui.Model model)

**Parameters:**

manager -

model -

**Returns:**

add-WHmanager. Shows the add WH manager form to the HQ manager.

hqManShowWHManagers

@RequestMapping("/HQmanager-WHmanagers-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManShowWHManagers(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

HQManager-WHmanagers-view. View list of Warehouse managers

addLFManager

@RequestMapping("/addlfmanager") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addLFManager(@Validated

Managers manager,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

manager -

result -

model - Adds local manager to database

addWHManager

@RequestMapping("/addwhmanager") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addWHManager(@Validated

WarehouseManager warehouseManager,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

warehouseManager -

result -

model -

**Returns:**

HQmanager-WHmanagers-view. Saves the data entered by HQmanager to WarehouseManager

hqManShowRestaurants

@RequestMapping("/HQmanager-restaurants-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManShowRestaurants(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

hqmanager-restaurants-view. View list of restaurants

hqManShowOffices

@RequestMapping("/HQmanager-offices-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManShowOffices(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

hqmanager-offices-view. View list of offices

hqManShowWarehouses

@RequestMapping("/HQmanager-warehouses-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) hqManShowWarehouses(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

manager-warehouses-view. View list of warehouses

**8.11 HQ Administrator**

showAdminList

@RequestMapping("/HQadmin-admin-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showAdminList(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

HQadmin-admin-view. Shows all current Local Admins to HQ Admin

showOfficesList

@RequestMapping("/HQadmin-offices-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showOfficesList(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

HQadmin-offices-view. Shows all offices to HQ Admin

showRestaurantList

@RequestMapping("/HQadmin-restaurants-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showRestaurantList(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

HQadmin-restaurants-view. Shows all restaurants to HQ Admin

showWarehouseList

@RequestMapping("/HQadmin-warehouses-view") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWarehouseList(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

HQadmin-warehouses-view. Shows all warehouses to HQ Admin

showAdminSignUpForm

@RequestMapping("/adminsignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showAdminSignUpForm(Admins admin,

org.springframework.ui.Model model)

**Parameters:**

admin -

**Returns:**

add-LFadmin. Shows add Local Admin page to HQ Admin

showOfficeSignUpForm

@RequestMapping("/officesignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showOfficeSignUpForm(Offices office)

**Parameters:**

office -

**Returns:**

add-office. Shows add office page to HQ Admin

showRestaurantSignUpForm

@RequestMapping("/restaurantsignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showRestaurantSignUpForm(Restaurants restaurant)

**Parameters:**

restaurant -

**Returns:**

add-restaurant. HQ administrators add restaurant page

showWarehouseSignUpForm

@RequestMapping("/warehousesignup") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showWarehouseSignUpForm(Warehouses warehouse)

**Parameters:**

warehouse -

**Returns:**

add-warehouse. HQ administrators add warehouse page

addAdmin

@RequestMapping("/addadmin") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addAdmin(@Validated

Admins admin,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

admin -

result -

model -

**Returns:**

HQAdmin-admin-view. Saves entered Local Admin information from HQ Admin. Verifies email does not already exist in Admins table. Updates restaurant to saved Admin.

addOffice

@RequestMapping("/addoffice") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addOffice(@Validated

Offices office,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

office -

result -

model -

**Returns:**

HQAdmin-offices-view. Saves entered office information from HQ Admin.

addRestaurant

@RequestMapping("/addrestaurant") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addRestaurant(@Validated

Restaurants restaurant,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

restaurant -

result -

model -

**Returns:**

HQadmin-restaurants-view. Saves entered restaurant information from HQ Admin. Loads a default inventory for New Restaurants location.

addWarehouse

@RequestMapping("/addwarehouse") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) addWarehouse(@Validated

Warehouses warehouse,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

warehouse -

result -

model -

**Returns:**

HQadmin-warehouses-view. Saves entered warehouse information from HQ Admin. Loads a default inventory for New Warehouses location.

showUpdateAdminForm

@GetMapping("/HQadminadminedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateAdminForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-LFadmin. Shows update admin page to HQ admin.

showUpdateOfficeForm

@GetMapping("/HQadminofficeedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateOfficeForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-office. Shows update office page to HQ admin.

showUpdateRestaurantForm

@GetMapping("/HQadminrestaurantedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateRestaurantForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-restaurant. Shows update restaurant page to HQ admin.

showUpdateWarehouseForm

@GetMapping("/HQadminwarehouseedit/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showUpdateWarehouseForm(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id -

model -

**Returns:**

update-warehouse. Shows update warehouse page to HQ admin.

updateAdmin

@PostMapping("/HQadminadminupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateAdmin(@PathVariable("id")

long id,

@Validated

Admins admin,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

admin -

result -

model -

**Returns:**

HQadmin-admin-view. Saves updated Local admin information entered by HQ Admin. Verifies that email information entered does not exist in Admins table.

updateOffice

@PostMapping("/HQadminofficeupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateOffice(@PathVariable("id")

long id,

@Validated

Offices office,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

office -

result -

model -

**Returns:**

HQadmin-offices-view. Saves updated office information entered by HQ Admin.

updateWarehouse

@PostMapping("/HQadminwarehouseupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateWarehouse(@PathVariable("id")

long id,

@Validated

Warehouses warehouse,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

warehouse -

result -

model -

**Returns:**

HQadmin-warehouses-view. Saves updated warehouse information entered by HQ Admin.

updateRestaurant

@PostMapping("/HQadminrestaurantupdate/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) updateRestaurant(@PathVariable("id")

long id,

@Validated

Restaurants restaurant,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

id -

restaurant -

result -

model -

**Returns:**

HQadmin-restaurants-view. Saves updated restaurant information entered by HQ Admin.

deleteOffice

@GetMapping("/HQadminofficedelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteOffice(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Office

model -

**Returns:**

HQadmin-offices-view. Deletes office from Offices table.

deleteRestaurant

@GetMapping("/HQadminrestaurantdelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteRestaurant(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Restaurant

model -

**Returns:**

HQadmin-restaurants-view. Deletes restaurant from Restaurants table.

deleteWarehouse

@GetMapping("/HQadminwarehousedelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteWarehouse(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Warehouse

model -

**Returns:**

HQadmin-warehouses-view. Deletes warehouse from Warehouses table.

deleteAdmin

@GetMapping("/HQadminadmindelete/{id}") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) deleteAdmin(@PathVariable("id")

long id,

org.springframework.ui.Model model)

**Parameters:**

id - Admin

model -

**Returns:**

HQadmin-admin-view. Deletes Local Admin from Admins table. Removes assigned restaurant before deleting admin to avoid deleting restaurant as well.

showHQAdminLog

@GetMapping("/hqlogadminview") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showHQAdminLog(org.springframework.ui.Model model)

**Parameters:**

model -

**Returns:**

hq-admin-log-view. Shows the entire log to the HQ Admin

**8.12 Order Placement**

JavaDoc

AddTaxes

public void AddTaxes()

Create new sales tax menu object based on cartItems total price Append to price of existing tax object if order exists

createNewOrder

public CartItems createNewOrder(long id,

CartItems cartItem)

**Parameters:**

id - Menu id

cartItem - Populate new cartItem or create cartItems attribute

**Returns:**

cartItem

addToSales

public void addToSales(Orders order)

**Parameters:**

order - Append to sales and profits for restaurant corresponding to order

showPaymentPage

@RequestMapping("/pay") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) showPaymentPage(org.springframework.ui.Model model)

**Parameters:**

model - Display payment input page

processPayment

@RequestMapping("/processpayment") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) processPayment(@Validated

PaymentDetails\_Form form,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

form - mirror of PaymentDetails for form submission

result -

model - Payment processing simulation method Does not do anything with PaymentDetails generated from form

processPaypal

@RequestMapping("/processpaypal") public [String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) processPaypal(@Validated

Paypal\_Form form,

org.springframework.validation.BindingResult result,

org.springframework.ui.Model model)

**Parameters:**

form - mirror of Paypal details for form submission

result -

model - Paypal login credentials processing simulation method Does not actually connect to Paypal, only simulates through credentials

**8.13 Inventory Management**

JavaDoc

removeFromInventory

public void removeFromInventory(Orders order)

**Parameters:**

order - Remove menu ingredients from restaurant inventory based on items in order

# File Path Names

* There are three file paths used for loading data into the system. Each of these file paths is stored as a private final String variable in the RestaurantController class. The files used are the following:
* src/main/resources/Menu.xlsx
* src/main/resources/Ingredients.xlsx
* src/main/resources/US-Tax-Info-By-State.xlsx

# Code Reusability

* There were some methods within the RestaurantController class that were created with the purpose of reusability, such as methods to return the authenticated user, return the guest customer, add, edit, or remove taxes from an order, and more.
* Aside from these methods called from several other methods within the controller, most of our classes are reusable, with the classes such as Customers, Admins, and Locations being very common entities that are tracked in other programs as well, with very similar or the same attributes given.

# Testing

# 11.1 Overview

This section 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

# 11.2 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

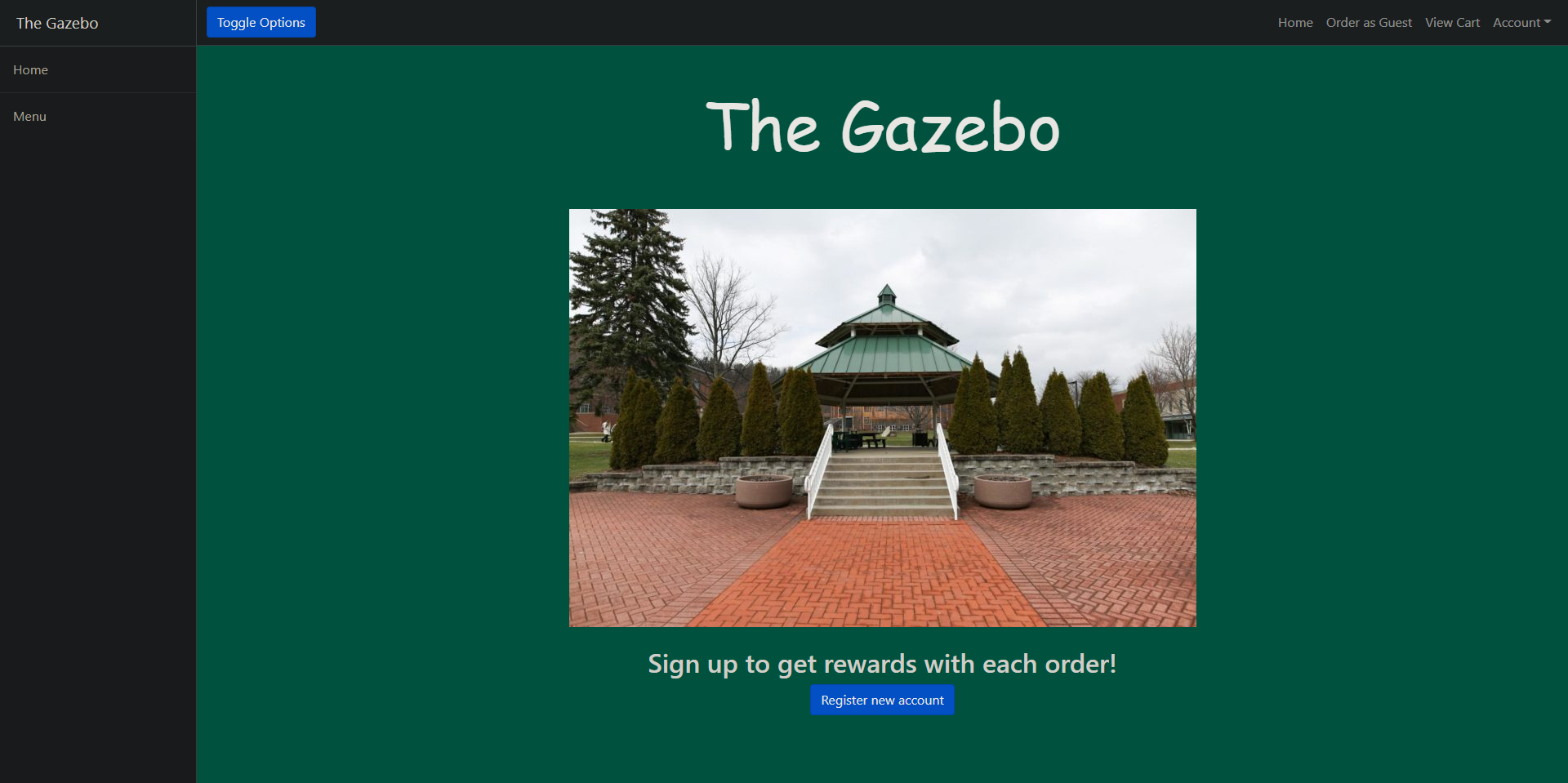


Figure 5

## 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.

# 11.3 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 the RestaurantController class, as that is where the majority of our code and methods are located. Each method from RestaurantController 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.

# 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.

# 11.4 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.

# 11.5 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.

# 11.6 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.

# Logging

* There is a viewable action log within the Restaurant Manager Assistant system. New entries to this log are created any time a major action is done within the system, such as adding new entities within the database, editing or removing existing entities, etc.
* The action log tracks the location, date, time, customer ID of the account making the change, a short description of what was done, and the ID of the entity affected by the change.
* The action log is fully viewable by users with the HQ Admin or HQ Manager role, and users with the Admin, Local Franchise Manager, or Warehouse Manager roles are able to view a subset of the action log, including only logged events at the specific location they work at.
* No user is able to edit or remove an entry from the action log, and log entries are created automatically, with no option to manually add an entry.

# Deployment / Maintenance

* If this system is to be deployed in a real-world environment, there are a few things that would need to be implemented, as they are only simulated within the current code.
* Payment details and paypal login credentials are not processed or connected to any payment gateway for services to be done. We currently only simulate payment, including customers paying for orders, employees getting paid, and restaurants’ profits.
* The basic warehouse employees are set up in the system but have no home pages, unlike all the other staff including warehouse managers.
* HQ logins are hardcoded, and you can not add any new HQ Administrators within the user interface under any authentication.
* Other future work or ideas we had that could be implemented are described within the evaluation manual found at src\main\resources\Documents\Evaluation Manual.
* If these things are fixed, the program will be ready for and able to be deployed and maintained.

# Post Mortem Analysis

* Looking back at this project after completion, there are a number of things we would have done differently.
* Firstly, had we learned JUnit5 near the beginning of the semester, it would have been very helpful to create test classes and methods before implementing the methods themselves instead of needing to create test methods for each method within the system near the end of the project, when we had already done the majority of the work with human factor testing.
* We also should have done our work in a more organized environment, as the majority of our code is within a single Controller class. Organizing and separating out tasks better would have been a more efficient use of our time, as disorganization led to the system becoming annoying to work with at some points.

# Figures

* Figure 1 - Domain UML class diagram
* Figure 2 - Login UML sequence diagram
* Figure 3 - Ordering UML state machine diagram
* Figure 4 - Serving staff UML sequence diagram
* Figure 5 - HQ Admin UML use case diagram
* Figure 6 - HQ Manager UML use case diagram
* Figure 7 - Local Admin UML use case diagram
* Figure 8 - Restaurant ER diagram
* Figure 9 - Warehouse ER diagram
* Figure 10 - Home page for guest users