

Mr. Pizza

Software Engineering (14:332:452)

Submitted May 5, 2024

<https://github.com/Wingo206/SoftwareEngineering>

Members

1. Joshua Menezes jam1092@scarletmail.rutgers.edu
2. Ethan Sie ems384@scarletmail.rutgers.edu
3. Vineal Sunkara: vss54@scarletmail.rutgers.edu
4. Damon Lin: dl1023@scarletmail.rutgers.edu
5. Aman Patel anp181@scarletmail.rutgers.edu
6. Anna Yeakel ahy21@scarletmail.rutgers.edu
7. Nikash Rajeshbabu nr623@scarletmail.rutgers.edu
8. Arya Shetty aas445@scarletmail.rutgers.edu
9. Keanu Melo Rojas knm95@scarletmail.rutgers.edu
10. Brandon Cheng bhc31@scarletmail.rutgers.edu
11. Thomas P O'Connell tpo21@scarletmail.rutgers.edu
12. Ji Wu jw1489@scarletmail.rutgers.edu

Table of Contents

Table of Contents.....	1
Report 1 Pt 1: Business Concepts, Operations, and Business Rules.....	6
Project Description.....	6
Data Collections and Availability.....	6
Data Operations.....	6
Business Policies.....	8
Sub Teams.....	11
Report 1 Pt 2: System Requirements.....	12
REQ Driver/Map Section.....	13
REQ-ORDER Section: Arya Shetty, Vineal Sunkara, Nikash Rajeshbabu.....	15
REQ-Menu Section.....	17
REQ-Customer Section: Aman Patel, Ethan Sie, Anna Yeakel.....	19
Business Glossary.....	23
Report 1 Pt 3: Use Cases.....	25
Actors and Goals.....	25
Map/Driver - Store locations, Order Tracking and Delivery.....	27
Map/Driver Use Cases:.....	27
Map/Driver Use Case Diagram.....	30
Map/Driver Detailed Use Cases.....	31
Pizza Order Pipeline.....	35
Pizza Pipeline use case diagram:.....	35
Pizza Pipeline Use Cases:.....	35
Pizza Pipeline detailed use case scenarios:.....	39
Menu Administration & Selection.....	45
Menu use case diagram:.....	45
Menu Use Cases.....	45
Menu Detailed Use Cases.....	46
Customer account, feedback, and analytics (Customer experience).....	49
Customer Accounts use case diagram:.....	49
Accounts Use Cases:.....	49
Accounts Detailed Use Cases.....	52
Report 2 Pt 1: Domain Models.....	55
Actor Roles.....	55
Map/Driver Sub-group Domain Model:.....	55
API Operations.....	55
Example Data Generation for Collections.....	57
Pseudocode.....	58
Detailed Central Use-Case: Delivery3.....	60
Pizza Ordering Pipeline Sub-Group Domain Model:.....	63

API Operations.....	63
Example Data Generation for Collections.....	64
Pseudocode.....	64
Detailed Central Use-Case - Payment Menu.....	65
Menu Sub-Group Domain Model:.....	69
API Operations.....	69
Example Data Generation for Collections.....	69
Pseudocode:.....	74
Detailed Central Use-Case:.....	75
Customer Sub-Group Domain Model:.....	77
API Operations.....	77
Example Data Generation for Collections.....	79
Pseudocode.....	79
Detailed Central Use Case - Customer Account.....	80
Report 2 Pt 1: ER Diagram and Schema.....	83
MySQL Schema.....	84
Database Access Permissions.....	86
Report 2 Pt 2: API Specification.....	88
Authentication.....	88
Map/Driver API Specification.....	89
Updated Map/Driver Sequence Diagrams.....	92
Use Case: Map6: Validate Input Address Sequence Diagram.....	92
Use Case: Delivery3: Calculate Optimal Assignment.....	93
Use Case: Chatbot1: Ask the Chatbot a Question.....	94
Pizza Ordering Pipeline API Specification.....	95
Updated Pizza Pipeline Sequence Diagrams.....	99
Menu API Specification.....	101
Updated Menu System Sequence Diagrams.....	103
Customer/Accounts API Specification.....	105
Updated Customer/Accounts Sequence Diagrams.....	108
Third-Party APIs.....	109
REST-based APIs.....	109
Non REST-based APIs / Libraries.....	110
Report 3: Server Side Routing Diagram.....	112
Customer/Accounts Subgroup.....	112
Map/Driver Subgroup.....	113
Map/Driver Subgroup Continued.....	114
Pizza Pipeline Subgroup.....	115
Menu Subgroup.....	116
Report 3: Steps to Run Server.....	117
Clone Repository.....	117

Setup Node Environment.....	117
Setup config.js.....	117
Setup MySQL Database.....	117
Setup SSL Certificate.....	117
Running the Server.....	119
Report 3: Backend Implementation Details.....	120
Routing.....	120
Config File.....	120
SQL Template Substitution.....	120
Input Validation Utilities.....	121
Report 3: API Documentation.....	122
Customer Accounts.....	122
Auth1: Create Customer Account.....	122
Auth2: Customer Login.....	124
Auth3: Employee Login.....	127
Auth4: Admin Login.....	130
Auth5: Get Auth Role.....	133
Auth6: Logout.....	137
Auth7: Get Employee Store Info.....	138
Accounts1: Get Customer Info.....	140
Accounts2: Edit Customer Info.....	142
Accounts3: Delete Account.....	144
Accounts4: Add new Employee.....	144
Accounts5: Assign Existing Employee.....	147
Accounts6: Get Employee Info.....	149
Accounts7: Edit Employee Info as Admin.....	149
Accounts8: Edit Employee Info as Employee.....	152
Support1: Create Support Ticket.....	154
Support2: Respond to Support Ticket.....	155
Support3: View All Tickets.....	156
Support4: View Past Tickets.....	157
Support5: View Unanswered Tickets.....	159
Analytics1: Get Total Revenue By Date.....	160
Analytics2: View Employee Count By Store.....	162
Analytics3: View Total Customers.....	164
Analytics4: View Total Employees.....	166
Analytics5: View Menu Items Sorted By Popularity.....	167
Analytics6: View Revenue Total Per Menu Item.....	170
Analytics7: View Toppings Sorted By Popularity.....	173
Map/Driver Use Cases.....	176
Map1: View Store Locations.....	176

Map2: View Detailed Store Information.....	177
Map3: Add Store Locations.....	178
Map4: Edit Store Locations.....	180
Map5: Get List of Store Names.....	183
Map6: Validate Input Address.....	184
Delivery1: View Unassigned Orders.....	190
Delivery2: See Available Drivers.....	194
Delivery3: Calculate Optimal Assignment.....	196
Delivery4: Assign Orders To Drivers.....	206
Delivery5: Get Assigned Order Information.....	212
Delivery6: Get Waypoints for Driving.....	215
Delivery7: Update Driver Location.....	222
Delivery8: See Driver's Location.....	226
Delivery9: See My Delivery Destination (Customer).....	228
Delivery10: Get Delivery Time Estimate.....	230
Chatbot1: Ask the Chatbot a Question.....	232
Pizza Pipeline.....	236
O1: Order Payment.....	236
O2: Apply Rewards Points.....	237
O3: Input Valid Address.....	243
O4: Stripe Payment.....	246
O5: Refund Order.....	246
O6: View All Customer Orders.....	253
O7: View All Store Orders.....	265
O8: Cancel Order.....	272
O9: View Order Status.....	277
Menu.....	284
Menu-Load.....	284
Menu-Search.....	287
Check-Availability.....	289
Check-Options-1.....	289
Check-Options-2.....	289
Admin-Edit-1.....	290
Admin-Edit-2.....	296
Admin-Delete-1.....	299
Admin-Delete-2.....	301
Admin-Add-1.....	302
Admin-Add-2.....	305
Employee-Availability-1.....	307
Employee-Availability-2.....	309
Report 3: Third-Party API Documentation.....	311

Map/Driver Subgroup.....	311
Calculate Route Matrix for Delivery1: Unassigned Order Time Estimation.....	311
Calculate Route Matrix for Delivery3: Optimal Assignment Calculation.....	315
Address Validation Request for Map6a: Validate Input Address, Accepted Address.....	326
Address Validation Request for Map6b: Validate Input Address, Invalid Address – Verification Required.....	329
Address Validation Request for Map6c: Validate Input Address, Invalid Address – Fix Required... 332	
Front-end API Usage.....	335
Pizza Pipeline Subgroup.....	336
Interaction with Stripe API for O4.....	336
Stripe Interaction for O4-2.....	341
Report 3 Contributions Tables.....	348
Report 4 Contributions Table.....	350

Report 1 Pt 1: Business Concepts, Operations, and Business Rules

Project Description

The hypothetical popular food chain, Mr. Pizza LLC, provides an online ordering website where customers can order pizza and other items. As the technical branch, we are responsible for creating a website for customers that allows them to place orders online for their favorite Mr. Pizza store. This system is designed to significantly enhance customer engagement through a user-friendly and interactive website. The platform will not only display a detailed menu for each store location but also incorporate real-time order tracking and personalized customer profiles. These features aim to streamline the ordering process and enrich the overall customer experience by providing functionality that is both intuitive and efficient.

The heart of the Mr. Pizza online ordering system lies in its utilization of the Google Maps API, which offers customers an interactive map to easily find store locations and navigate delivery routes. This integration facilitates a seamless connection between the customers and the delivery drivers, ensuring that orders are tracked in real time, which enhances delivery reliability and customer trust. Additionally, the system includes a robust customer portal where users can manage their profiles, view past orders, redeem rewards points, and access support services, thus fostering a more connected and rewarding user experience.

On the administrative side, the platform is equipped with comprehensive backend functionalities that allow employees and managers to handle orders effectively, customize menus per store, and access real-time analytics and business insights. These features are crucial for optimizing operational efficiency and adapting business strategies based on customer behavior and sales trends. Enhanced security measures are also a cornerstone of our project, ensuring that all user data is securely managed and protected against potential threats. By integrating these sophisticated tools and technologies, our goal is to not only improve the operational aspects of Mr. Pizza but also to create a dynamic and engaging platform that attracts new customers and retains existing ones.

Data Collections and Availability

The groups that will be accessing our data collections are Visitors, Customers, Employees, and Admins. Customers have access to anything Visitors have access to.

- **Customer Accounts:** Customers and Admins.
- **Stores:** Visitors and Admins.
- **Employee Accounts:** Employees and Admins.
- **Menu Items / Customizations:** Visitors and Admins.
- **Orders:** Customers and Employees.
- **Deliveries:** Employees
- **Help Tickets:** Customers and Employees.
- **Reviews:** Customers and Employees.
- **Chatbot Data:** Visitors.

Data Operations

- **Customer Accounts**

- Visitors: Create an account with a username, email, phone number, and password
- Customers: Change their account information.
- Customers: View their account information.
- Admins: Add, edit, and delete accounts.
- Admins: View total number of customers.
- **Stores**
 - Visitors: View a map of store locations.
 - Visitors: View detailed store information.
 - Visitors: Find the closest store with an order time estimate.
 - Visitors: View list of store names and IDs
 - Admins: Add a new store with store ID and location.
 - Admins: Edit and Delete stores.
- **Employee Accounts**
 - Admins: Add employees with an ID, name, role, and assigned store.
 - Admins: View total amount of currently employed employees and their assigned store.
- **Menu Items / Customizations**
 - Visitors: Select between store locations.
 - Visitors: View menu item information and their prices.
 - Visitors: Cancel the view of an item.
 - Visitors: Use search to filter through menu items.
 - Visitors: Restart the search and view the original menu.
 - Customers: Select between the customization options.
 - Customers: Select a quantity.
 - Customers: Add menu items to their cart.
 - Customers: Delete menu items from their cart.
 - Admins: Add, edit, and delete menu items from a specific store location.
 - Employees: Set the status of Menu Items at a specified store.
- **Orders**
 - Customers: Place an order using the items in their cart and a specified store.
 - Customers: View an order's status
 - Customers: View current and completed orders.
 - Customers: Cancel an order
 - Customers: Refund an order
 - Customers: Apply rewards before checking out to decrease total price
 - Customers: Choose delivery or carryout orders
 - Customers: View delivery driver status of order
 - Customers: View all past and current orders of logged in customer
 - Employees: View all orders for the store they have been assigned to.
 - Employees: Set the status of an order to **Processing, Paid, Started, Ready, In-Transit, Delivered, Canceled, Rejected, Refunded, and Completed.**
 - Employees: Update the order's tracking information and location.
- **Deliveries**
 - Employees: See available drivers.
 - Employees: See assigned delivery batch.
 - Employees: Get optimized routes.

- Employees: Mark delivery as completed.
- Employees: Update delivery batch location.
- Employees: View unassigned orders.
- Employees: Assign delivery batch to drivers.
- Customers: View an order's tracking information on a map.
- Customers: Can mark order as **not delivered** if status shows **completed**
- **Support Tickets**
 - Customers: Create a support ticket to ask a question
 - Customers: view previous tickets that the account has created
 - Employees: view all unanswered support tickets
 - Employees: respond to support tickets
- **Reviews and Statistics**
 - Visitors: View total and average reviews on stores and menu items at a store.
 - Customers: Place reviews of text prompts on menu items.
 - Customers: View their past reviews.
 - Employees and Admins: View all orders or all orders with filters such as selected items, store, status, and date.
 - Employees and Admins: View statistics on orders and menu items, such as total orders, and item selection frequency, for a given store and time frame.
 - Admins: View costs, revenue, and profit of stores or the entire business.
 - Admins: Statistics on Customers such as number of accounts.
 - Admins: Statistics on average delivery time
 - Employees and Admins: View Revenue Total by menu item.
 - Employees and Admins: View most popular menu items.
 - Employees and Admins: View most popular toppings.
- **Mr. Pizza Chatbot**
 - Visitors: Ask a question to the chatbot

Business Policies

Accounts

- The passwords will be hashed with bcrypt to ensure security. When a login request is received, the incoming password will be hashed and the resulting hash will be checked with the stored hash to verify that the inputted password matches, but the actual password will never be stored. This will ensure that in the event of a data breach, customers' passwords are never leaked.
- Sensitive data such as credit card information and the default delivery address are not stored in the Mr. Pizza database since handling this data introduces data vulnerabilities.
- Customers must have an account to order.

Feedback

- After ordering, users can leave a review for items they ordered in the customer order history tab.
- Reviews will be able to be seen on each food item by employees.
- Customer accounts will be tied to the reviews.

Chatbot

- Anyone who visits the Mr. Pizza website may interact with the chatbot with questions or other queries.
- Information about Mr. Pizza's business policies will be made clear to the users through the use of the chatbot.
- The Chatbot will suggest submitting a help ticket if there is a question that it cannot confidently answer.

Employees

- Customers will be able to see who is currently handling their orders throughout the entire process.
- The types of employees are *Driver*, *Default*
- The backend system will only keep track of the employees and orders, so store functionality should be handled locally.

Drivers

- Employees can act as drivers for an order.
- Drivers will be tracked when the customer is checking on delivery progress. They will also mark when the delivery is “in transit,” “completed,” or “failed” on their end.
- Drivers will have special access to the map subsystem of the backend site. This will enable them to see map directions, special instructions, and general order information.
- Drivers will have one of the following statuses:
 - *Idle*: The driver is ready to receive a delivery order.
 - *Assigned*: The driver has been assigned a delivery order but has not yet started delivering.
 - *Delivering*: The driver is on the way to deliver an order.
 - *Returning*: The driver has delivered an order and is returning to the store.
- When an order is marked as delivery, the system will choose a driver in the *Idle* state. A driver is picked based on time since their status has changed to idle, in order of status age. Ex. A driver who was idle for 20 minutes will be picked rather than a driver who was idle for 2 minutes.
- Drivers can only deliver their assigned order, and will not be able to simply grab any order and go. Once the *Idle* status is assigned to the driver, the driver may be notified to take their order.
- Once the driver is ready to go they can set the order status to “in transit,” and once delivered they can set the order status to completed
- The order in which routes are chosen for delivery is optimized according to our business beliefs. We believe that having one really angry customer has a larger negative impact than having two somewhat unsatisfied customers. Thus, we will calculate the routes in a way that will attempt to minimize the longest wait times. For example, if there are 3 orders with the 3rd one having waited significantly longer than the first two, our algorithm might tell the driver to go to the third even if that leads to a greater total driving time. We use a second order function for our waiting time in our route optimizer so that the impact of orders that have waited longer is felt more.

Menu

- The menu page allows the user to choose a local store to deliver or pick up the order, and proceed to checkout after placing items into a cart.
- The user will be able to select their items from the Mr. Pizza menu.

- The menu is specific to chosen location, with updates to availability of menu being taken into account for each store.
- Menu items are categorized into drinks, pizzas, desserts, pastas, etc.
- Menu items can be customized with different sizes, toppings, and other options but these options are chosen by an admin to limit customization.
- Each size, topping and customization option has a price that is added or removed from the item's base price.
- The menu shown is not an original Mr. Pizza menu, but instead based on and scraped from the Domino's menu, which was done to incorporate outside sources to the menu.

Orders

- All stores allow orders within a standardized hourly schedule.
- The user must be signed in as a customer or register an account in order to place their order.
- For the users who signed in, the account information will be used to automatically send confirmation, refund, and cancel emails.
- Once purchased, the order will be sent to the local store to be made.
- The customer will be immediately charged after placing the order to deter fake orders, but users are still able to fully refund until the order exits the processing state.
- Order statuses are **Processing, Paid, Started, Ready, In-Transit, Delivered, Canceled, Rejected, Refunded, and Completed.**
- Once payment is confirmed using Stripe API, the order is put into **Paid** status
- Once the store is ready to fulfill an order, they place the order in **Started** status.
- Orders can only be canceled while in the **Processing** and **Paid** status.
- Once the food for an order has been made it is put into **In-Transit** status, and a delivery driver employee will start delivering it.
- If a location is not set, the order is put into **Ready** status, for the customer to pick up.
- Once an order has been delivered or picked up, the status is marked as **Completed**.
- Users can only start their order online. If they call or show up in person, an employee will order for them using the site.
- If the order cannot be completed within a reasonable timeframe, either through the customer not picking up or being unable to confirm the delivery, the order will be marked as **Rejected** and the customer will not be refunded.
- At the discretion of the employee, if the order is unreasonable in nature, they have the right to reject the order, and the customer will be allowed to refund it.
- If the customer cancels or refunds the order, the status will be set to **Refunded** or **Canceled**.
- If there is some error with Stripe API, the order will be left in the **Processing** state. This is the default state of orders when entering the database.
- The hours of operation are 9 AM to 5 PM every day.

Rewards

- The rewards program gives points, which are gained through purchasing orders and can be used to redeem free food items during future orders.
- Every order of at least \$5 grants one *Pizza Point (PP)*.
 - After 5 *PP* is acquired, the customer has the option of redeeming a free reward item, such as a medium cheese pizza.

- These items can only be added if the order already contains an item that will be paid for by the customer. For example, a customer can add a pizza to their order, then redeem the free pizza, and proceed to checkout as normal.
- The free pizza will be added to the order as normal, just without a price. The total price will stay the same, but will decrease when checking out with Stripe.
- The rewards points will be used for the one item that is the most expensive but under \$20 in the order.
- Once a customer has redeemed the pizza and placed the order, their pizza points are deducted by 5 PP.
- A customer cannot make their *PP* smaller, even by disabling the rewards program. Each account will always keep track of the amount of *PP* the account has accrued. The only way *PP* can decrement is if the order was canceled or refunded while the order was in the processing state.

Search

- The search operation and functionality focus primarily on filtering menu items to create a new menu.
- This text-based search will only accept a reasonable number of characters, and after searching, the page will be populated with menu items sorted by categories.
- By using FZF's filter function, our search tool will be able to identify the user's desirable item/s regardless of incomplete words or misspelling.
 - For example, pizza, piazz, and piza will be seen as the same input, being pizza, which will then be used to filter items that do not match this string in their names.
 - This feature helps our users with the ease of use for both viewing the menu and searching items, since the user can quickly make a new menu that will only include items with the key word.
- A reset button will be provided in case the user may want to revert the filter changes done by the search operation/s.

Sub Teams

Subteam 1 - Map/Driver - Store locations, Order Tracking and Delivery: Brandon Cheng, Ji Wu, Damon Lin

Subteam 2 - Pizza - Ordering Pipeline and Checkout: Arya Shetty, Vineal Sunkara, Nikash Rajeshbabu

Subteam 3 - Menu - Items, Store-based availability, Adding items to cart, Item customization: Joshua Menezes, Keanu Melo Rojas, Thomas O'Connell

Subteam 4 - Customer - Accounts, Feedback, Rewards: Ethan Sie, Anna Yeakel, Aman Patel

Report 1 Pt 2: System Requirements

Part 2 Contributions	
Groups	Percentage/Part
Map/Driver	Brandon Cheng (33%), Ji Wu (34%), Damon Lin (33%)
Pizza Pipeline	Arya Shetty (33%), Vineal Sunkara (33%), Nikash Rajeshbabu (33%)
Menu and Items	Keanu Melo Rojas (33%), Joshua Menezes (34%), Thomas O'Connell (33%)
Customer/Accounts	Ethan Sie (33%), Anna Yeakel(34%), Aman Patel(33%)

REQ Driver/Map Section

REQ-Visitor-Map-1- As a visitor, I am able to view different store locations on the map view.

Business Concepts: Restaurant Locations

Data Operations: View a map of store locations.

REQ-Visitor-Map-2- As a visitor, I am able to select a particular location and get a small overview of the location.

Business Concepts: Restaurant Locations

Operations: View detailed store information.

Business Rules: Overview includes the rating of the branch, busyness, and a picture of the branch.

REQ-Visitor-Map-3- As a visitor, I can get a time estimate on the nearest store location.

Business Concepts: Time Estimation

Operations: Find the closest store with an order time estimate.

Business Rules: Time estimate is based on location, busyness and distance.

REQ-Map-2- As a customer, I can track where my driver is along his route through the Google Maps API.

Business Concepts: Communication, Tracking

Operations: View an order's tracking information on a map.

Business Rules: The customer may only access his/her particular driver's location after ordering

API Usage: Google Maps, Geolocation

REQ-Map-3- As a customer, I can see the approximate route my driver is taking through the Google Maps API.

Business Concepts: Tracking

Operations: View an order's tracking information on a map.

Business Rules: Customers can check the status of their order whenever they wish.

API Usage: Google Maps

REQ-Map-4- As a customer, I can see approximately how long the driver will take through the Google Maps API.

Business Concepts: Tracking

Operations: View an order's tracking information on a map.

Business Rules: Customers can check the status of their order whenever they wish.

API Usage: Google Maps

REQ-Map-5- As an admin, I can add, edit, or delete restaurant locations through the Google Maps API.

Business Concepts: Restaurant Locations

Operations: Add a new store with store ID and location. Edit and Delete stores.

Business Rules: No one restaurant location can have the same store ID or location.

API Usage: Google Maps

REQ-Map-6- As an admin, I can get a compilation of all store branches.

Business Concepts: Restaurant Locations

Operations: View list of store names.

Business Rules: Visitors can view stores to choose the store they want to look at the menu for.

REQ-Driver-1- As a delivery driver, I can mark an order as delivered after I deliver it.

Business Concepts: Post-Order Protocol

Operations: Set/update the status of an order

Business Rules: Orders marked as delivered will be fully completed.

REQ-Driver-2- As a delivery driver, I can cancel the delivery if some accident occurs.

Business Concepts: Emergency Protocol

Operations: Set/update the status of an order

Business Rules: Drivers need to be able to cancel the delivery if there are unforeseen circumstances while driving.

REQ-Driver-Map-1- Drivers can get a path from the Google Map API to get to their destination.

Business Concepts: Delivery Protocol

Operations: Pathfinding

Business Rules: Drivers are to use the Google Map API available to drivers on the website.

API Usage: Google Maps

REQ-Driver-Map-2- Drivers can get an optimal route that minimizes overall tardiness when they need to send more than one order per trip through data taken from Google Maps.

Business Concepts: Delivery Protocol

Operations: Pathfinding

Business Rules: Drivers are to abide by the priority set by the website (overall tardiness over individual order tardiness)

API Usage: Google Maps

REQ-Driver-Map-3- Drivers can see their assigned orders.

Business Concepts: Delivery Protocol

Operations: View all orders for a given store

Business Rules: Drivers are able to view details regarding their assigned orders.

REQ-Driver-Map-4- As an employee, I can view a list of all of the unassigned orders.

Business Concepts: Delivery Protocol

Operations: View all orders for a given store.

Business Rules: Employees can see orders that are ready to be assigned to drivers.

REQ-Driver-Map5- As an employee, I am able to assign orders to drivers.

Business Concepts: Delivery Protocol

Operations: Assign delivery batch to drivers.

Business Rules: Employees are able to use the Google Maps API to assign orders to drivers.

REQ-Chatbot-1: As a visitor, I can access the Mr. Pizza AI Chatbot and ask general questions about Mr. Pizza.

Business Concepts: Customer Support

Operations: Ask Chatbot a question

Business Rules: The information is limited to that from report and menu. The Chatbot does not handle any real-time information.

REQ-Chatbot-2: As a Visitor, I can ask the Mr. Pizza AI Chatbot questions about the specific menu items.

Business Concepts: Customer Support

Operations: Ask Chatbot a question

Business Rules: The information is limited to that from report and menu. The Chatbot does not handle any real-time information.

REQ-ORDER Section: Arya Shetty, Vineal Sunkara, Nikash Rajeshbabu

REQ-Order-Creation1: As a customer, I can create an order of menu items on the website to have the food items delivered or picked up, which will be stored in a cart.

Business Concepts: Customer experience

Operations: Add Menu Items to their cart, place an order using the items in their cart and a specified store

Business Rules: Orders can be created by guest or customer accounts, and viewed by employees

REQ-Order-Creation2: As an employee, I can view an order to start making orders to progress the process.

Business Concepts: Worker access

Operations: View an order's status and tracking information on a map

Business Rules: Orders can be viewed and marked as creating, ready, and completed

REQ-Order-Creation3: As an employee, I can start the delivery process and get the food to the customer completing the order.

Business Concepts: Worker access

Operations: Set the status of an order to processing, paid, started, in transit, ready, rejected, canceled, refunded, and completed.

Business Rules: Orders can be viewed and marked as creating, ready, and completed

REQ-Order-Creation4: As a customer, I can pay for my order through the Stripe API

Business Concepts: Customer Experience, Payment Handling

Operations: Record customer credentials and process transaction

Business Rules: Orders that have processed payment will continue on the pizza pipeline

REQ-Order-Creation5: As a customer, I will receive a receipt in the form of email after completion of order through the NodeMailer API

Business Concepts: Communication, Notification

Operations: Automatic email messaging

Business Rules: If the order was successful, the customer will be notified of the status

REQ-Order-Creation6: As a customer, I can input my desired address if I choose the order to be a delivery.

Business Concepts: Customer Experience

Operations: Address Validation

Business Rules: Must be a valid address within a 7-mile radius of the Mr. Pizza store, if the address is valid, the customer may proceed to payment.

REQ-Order-Creation7: As a customer, I can redeem my rewards points (once they reach a count of 5) in order to redeem exactly one quantity of the most expensive item in the order.

Business Concepts: Customer Experience

Operations: Rewards Points Usage

Business Rules: Must have at least 5 rewards points before a customer can redeem an item. Reward points are incremented by one for every successful order. If a customer cancels/refunds an order, one reward point will be deducted.

REQ-Order-Management1: As an employee, I can change the status of the order as the order progresses in order to keep the customer informed about the order.

Business Concepts: Employee access

Operations: View current and completed orders.

Business Rules: Orders can be reviewed from a store's history.

REQ-Order-Management2: As a customer, I can view the order history of the orders I have placed in the past.

Business Concepts: Customer access

Operations: View current and completed orders.

Business Rules: Orders can be reviewed from a customer's history.

REQ-Order-PostProcess1: As a customer, I can refund or cancel an order if it has not been started to receive my payment back through the Stripe API

Business Concepts: Customer access

Operations: Access transactions history

Business Rules: Orders can be reviewed from a customer account's history and refunded or canceled

REQ-Order-Status-1- As a customer, I can view the order status with ease after I place it.

Business Concepts: Order Review

Operations: View an order's status and tracking information on a map.

Business Rules: An order status may not be viewed if an order was not placed.

REQ-Order-Status-2- As a customer, I can get updates on the order status as it changes via images that refresh depending on current order status.

Business Concepts: Order Review

Operations: View an order's status and tracking information on a map

Business Rules: The status and location are only viewable with the associated account.

Business Rules: Order statuses may only be changed to a different state under reasonable circumstances.

REQ-Menu Section

REQ-Menu-Load-1- As a visitor, I can see one of Mr. Pizza's menus after selecting a store location, so I only get to see what is available at that specific location.

Business Concepts: Selection of menu.

Operations: Query the server for the menu of the given store location.

Business Rules: A menu will be given only when the visitor selects a store location.

REQ-Menu-Search-1- As a visitor, I can search for a set of items, so I can view a menu that only has items of my desire.

Business Concepts: Search for items.

Operations: Filter items based on a text input.

Business Rules: The given items must match the text that the visitor gives in the search tool.

REQ-Menu-Search-2- As a visitor, I can restart any filtering, so I can obtain and view the original menu.

Business Concepts: Search for items.

Operations: Query the server for the menu it originally gave to the visitor.

Business Rules: The given menu must not be filtered and it should be the menu of the store location the visitor selected at the start.

REQ-Menu-Select-1- As a visitor, I can view information and customization options for any item after selecting an item from the menu.

Business Concepts: Selection of items.

Operations: View menu item information, options, and their prices.

Business Rules: Only available items and options are visible to the visitor.

REQ-Menu-Select-2- As a visitor, I can cancel the view of an item to go back to the menu.

Business Concepts: Navigation.

Operations: Revert interactions.

Business Rules: The visitor may see the menu they currently had before selecting an item and not repeat the search and/or selection of the menu.

REQ-Menu-Items-1- As an administrator, I can add and delete items, so that I can specify what items the menu displays.

Business Concepts: Menu management.

Operations: Set the status of Menu Items at a specified store.

Business Rules: Only administrators should be able to add or delete items from the menu.

REQ-Menu-Items-2- As an administrator, I can change the status of an item, so that I can make the menu alert the availability of an item.

Business Concepts: Menu management.

Operations: Set the status of Menu Items at a specified store.

Business Rules: Item status changes should reflect availability in the menu.

REQ-Menu-Options-1- As an administrator, I can add and delete customization options, so that I can specify what options/changes an item can receive.

Business Concepts: Menu customization.

Operations: Add, edit, and delete menu items from the entire Mr. Pizza franchise.

Business Rules: Administrators should manage customization options.

REQ-Menu-Options-2- As an administrator, I can change the prices for customization options, so that I can control the price of items that have special requests from the customer.

Business Concepts: Pricing, menu customization.

Operations: Set the status of Menu Items at a specified store.

Business Rules: Prices for customization should be set by administrators.

REQ-Menu-Customization-1- As a customer, I can change between an item's customization options, so that I can make an item.

Business Concepts: Customization.

Operations: Change the properties of an item.

Business Rules: The visitor will be limited to select certain options due to a property's mutual exclusivity.

REQ-Menu-Customization-2- As a customer, I can choose the quantity of an item I want to order so that I can order multiple of one item without having to select it again.

Business Concepts: Ordering, user interface.

Operations: Customize aspects of menu items.

Business Rules: The visitor will be limited to make this number range between 1 and a given maximum value.

REQ-Menu-Cart-1- As a customer, I can add items to save them in my final order.

Business Concepts: Ordering.

Operations: Add item/s to the order.

Business Rules: The selected and personalized item/s are the only thing/s added to the order.

REQ-Menu-Cart-2- As a customer, I can delete items from my final order.

Business Concepts: Ordering.

Operations: Delete item/s to the order.

Business Rules: The selected and personalized item/s are the only thing/s deleted from the order.

REQ-Customer Section: Aman Patel, Ethan Sie, Anna Yeakel

REQ-Customer-Account1: As a potential customer, I want to create an account in which I can view my order history and accumulate reward points.

Business Concepts: Account Registration, Personalization

Operations: Collect user information, validate information, and create user profiles

Business Rules: Valid email required, proper password complexities.

REQ-Customer-Account2: As a returning customer, I want to log into my account to view my order history and check on my reward point status.

Business Concepts: Account Access, Authentication

Operations: Input credentials, validate credentials, access profile

Business Rules: Account must exist, (possible authentication feature? Though that might be too complicated)

REQ-Customer-Account3: As a returning customer, I want to view my previous orders, so that I can easily repurchase items or track my spending.

Business Concepts: Order History, Transaction Tracking

Operations: Retrieve order history, display orders

Business Rules: Only orders made by the logged-in user are accessible

REQ-Customer-Account4: As a returning customer, I want to view my points balance and see how many points I have and understand what rewards I am eligible for.

Business Concepts: Loyalty Program, Reward Points

Operations: Display reward points balance, outline redemption options

Business Rules: Points are earned with each purchase, and points expire after a certain period

REQ-Customer-Account5: As a customer, I can log into my account and edit my account information

Business Concepts: Account Personalization

Operations: Edit customer account information

Business Rules: Must be logged into a valid customer account.

REQ-Customer-Account6: As a customer, I can delete my account from the Mr. Pizza database.

Business Concepts: Account Personalization

Operations: Delete a customer account

Business Rules: Must be logged into a valid customer account.

REQ-Customer-Support1: As a customer, I can submit a support ticket to ask a question about a specific function of the website.

Business Concepts: Customer support

Operations: Access profile, input credentials, access help ticket section, input question

Business Rules: Must be signed in

REQ-Customer-Support2: As a customer, I can view my previous support tickets that I have submitted with the employee's answer.

Business Concepts: Customer support

Operations: Access profile, input credentials, access help ticket section

Business Rules: Must be signed in

REQ-Employee-Support1: As an employee, I can view unanswered support tickets, the customer id who created each ticket, and can respond to unanswered tickets.

Business Concepts: Customer support

Operations: Input credentials, access help ticket answer section, input response

Business Rules: Must be signed in as an employee

REQ-Admin-Account1: As an administrator, I want to have the ability to create customer accounts to assist with manual registrations or special cases.

Business Concepts: Account Management, Administrative Support

Operations: Collect user information, validate information, manually input data, create user profiles.

Business Rules: Must have administrative privileges, all user information must meet existing business rules for validation.

REQ-Admin-Account2: As an administrator, I need to update customer account information upon request or to correct errors, ensuring all information is current and accurate.

Business Concepts: Account Maintenance, Data Accuracy

Operations: Search for customer profiles, edit user information, validate updated information, save changes

Business Rules: Must verify administrative access, changes must adhere to the same validation rules as new accounts (e.g., valid email, proper password complexities).

REQ-Admin-Account3: As an administrator, I want the capability to delete customer accounts, either at the customer's request or due to firing.

Business Concepts: Account Lifecycle Management, Compliance

Operations: Identify account for deletion, verify deletion criteria, execute account deletion.

Business Rules: Administrative access required, must follow a clear process for deletion that includes customer notification (where applicable) and data retention policies.

REQ-Admin-Account4: As an administrator, I need to access and manage the feedback provided by customers to address their concerns and improve the service quality.

Business Concepts: Feedback Management, Quality Control

Operations: Access feedback database, categorize feedback, initiate response or action, archive feedback upon resolution

Business Rules: Administrative access required, feedback must be managed within a specific timeframe, privacy and confidentiality of customer information must be maintained.

REQ-Admin-Account5: As an administrator, I must have the ability to manually adjust reward points in a customer's account, either to correct errors or as part of promotional activities.

Business Concepts: Loyalty Program Management, Customer Service

Operations: Search for customer accounts, adjust reward points balance, log reason for adjustment, notify customer of changes

Business Rules: Adjustments must be authorized and documented, customers should be notified of any changes to their points balance, adjustments should comply with loyalty program rules.

REQ-Admin-Account6 : As an admin I have the ability to login to my account and see my account information.

Business Concepts: Information Accessibility

Operations: retrieve account information

Business Rules: Must have a registered and valid admin account.

REQ-Admin-Account7 : As an admin I have the ability to add new employees, assign existing employees to different stores, and edit employee information.

Business Concepts: Employee Management

Operations: Add new employees, assign existing employees to new store

- Business Rules: Must have be signed into a valid admin account.
- REQ-Employee-Account1** : As an employee I have the ability to login to my account and see my account information.
- Business Concepts: Information Accessibility
 Operations: retrieve account information
 Business Rules: Must have a registered and valid employee account.
- REQ-Employee-Account2**: As an employee I have access to edit my own email and password after being added to the employee database.
- Business Concepts: Account Maintenance
 Operations: Edit employee information
 Business Rules: Must have a valid registered employee account.
- REQ-Analytics1**: As an Admin, I want to view the total company revenue for a specific time period to assess financial performance.
- Business Concepts: Financial Reporting, Revenue Tracking
 Operations: Select date range, calculate total revenue, display results
 Business Rules: Must have a valid registered Admin account.
- REQ-Analytics2**: As an Admin, I want to get the total company revenue by month to prepare detailed monthly financial reports.
- Business Concepts: Financial Analysis, Monthly Reporting
 Operations: Input year and month, retrieve revenue data, summarize findings
 Business Rules: Must have a valid registered Admin account.
- REQ-Analytics3**: As an Admin, I want to see the employee count by store to manage staffing requirements effectively.
- Business Concepts: Staff Management, Resource Allocation
 Operations: Retrieve employee data by store, count employees, display totals
 Business Rules: Must have a valid registered Admin account.
- REQ-Analytics4**: As an Admin, I want to view the total number of customers to measure the reach of our marketing campaigns.
- Business Concepts: Customer Base Analysis, Marketing Effectiveness
 Operations: Count total registered customers, display total
 Business Rules: Must have a valid registered Admin account.
- REQ-Analytics5**: As an HR manager, I want to view the total number of employees to ensure compliance with labor regulations.
- Business Concepts: Compliance, Human Resources Management
 Operations: Summarize total employees across all departments and locations
 Business Rules: Must have a valid registered Admin account.
- REQ-Analytics6**: As an Admin, I want to view menu items sorted by popularity to adjust menu offerings based on customer preferences.
- Business Concepts: Menu Management, Customer Preferences
 Operations: Retrieve order data, calculate frequency of menu items ordered, rank and display by popularity
 Business Rules: Must have a valid registered Admin account.
- REQ-Analytics7**: As an Admin, I want to view the revenue generated by each menu item to identify high-performing products.
- Business Concepts: Product Performance, Revenue Attribution
 Operations: Calculate revenue per menu item, sort by revenue amount, display results

Business Rules: Must have a valid registered Admin account.

REQ-Analytics8: As an Admin, I want to view the most popular toppings ordered to ensure adequate stock levels.

Business Concepts: Inventory Management, Customer Demand

Operations: Analyze order data for toppings, rank by frequency of orders, display results

Business Rules: Must have a valid registered Admin account.

REQ-Accounts1: As a logged user to either a customer, employee, or admin account, I can logout of my account when leaving the Mr. Pizza website.

Business Concepts: Account Security

Operations: logout of logged in user's account

Business Rules: Must be logged into a valid account with Mr. Pizza.

Business Glossary

- **Address Validation:** A system that allows users, either admins when editing store locations, or customers when going through the checkout process, to input a human-readable address, which will be interpreted, validated, and converted into latitude and longitude coordinates.
- **Admin:** Users who can add, modify, and delete account information as well as update the availability of items.
- **Customer:** A person who uses the Mr. Pizza website to browse food items and order either takeout or delivery.
- **Cart:** A list of items selected by the customer that can be edited with food items selected from the menu.
- **Chatbot:** A helpful assistant on the Mr. Pizza home page that can help answer some questions visitors may have.
- **Customer Account:** A user that has signed up to become a member of Mr. Pizza to receive the perks of being able to save their credentials, view their order history, their rewards program status, and any help requests tied to that account.
- **Delivery Protocol:** Delivery order is prioritized to optimize for minimum squared customer waiting time, which is the sum of total customer waiting time squared. This will ensure that customers who have been waiting for a long time are prioritized when calculating the optimal delivery route.
- **Driver:** Employees who carry out delivery for the store; they are allowed to see where to deliver to and general order information on the website. The most efficient route to multiple deliveries will be provided to the driver via the map within the interface.
- **Fulfillment:** Policy that requires customers to be able to retrieve Pizza within a certain time window whether it be delivery or pick up. If an order is not fulfilled by the customer, the customer will be charged. If the order is not fulfilled on Mr. Pizza's end, the customer will be issued a refund or given store credit for their next order.
- **Locations:** Our locations are selected so that our customers get the quickest delivery times and high customer satisfaction. Each location may also have unique menu items.
- **Menu:** The Mr. Pizza official menu, which customers can order from, is displayed on the website.
- **Menu Items:** A product offered by Mr. Pizza.
- **Mr. Pizza Chat Bot:** A chatbot trained to provide real-time support by answering general questions about Mr. Pizza's operating policies and Mr. Pizza's Menu contents.
- **Order:** Placed by customers on the website, used to keep track of food purchased to make and deliver.
- **Order Batch:** A set of orders that are delivered in one trip by a driver.
- **Operating Hours:** Operating hours for all Mr. Pizza Locations. The hours of operation are 9 AM to 5 PM every day.
- **Payment:** Customer has the option to provide card and pre-pay online whether it's delivery (tip for the driver can also be included on the card) or pickup. The customer can also pay in cash for pickup or delivery.
- **Rewards Points:** Synonymous with Pizza Points. Customers gain 1 reward point after successful purchase, and can store up to a max of 5 points.
- **Rewards System:** A rewards program for customer accounts that allows users to redeem pizza points for free pizza on their next orders.

- Pizza Points: Synonymous with Rewards Points.
Shopping Cart: Cart for customers to add, edit, and remove menu items before continuing with their transaction.
- Tracking: An action that either an employee or a customer can perform that displays the status of the order and, if possible, the location of the order.
- Visitor: Any person who visits the Mr. Pizza website.
- Worker: Employees who work in person at the Mr. Pizza restaurants, they take care of cooking, cleaning, managing the store, and handling customers.

Report 1 Pt 3: Use Cases

Part 3 Contributions	
Groups	Percentage/Part
Map/Driver	Brandon Cheng (34%), Ji Wu (33%), Damon Lin (33%)
Pizza Pipeline	Arya Shetty (33%), Vineal Sunkara (33%), Nikash Rajeshbabu (33%)
Menu and Items	Keanu Melo Rojas (34%), Joshua Menezes (33%), Thomas O'Connell (33%)
Customer/Accounts	Ethan Sie (33%), Anna Yeakel (34%), Aman Patel (33%)

Actors and Goals

Users:

- Visitors (Initiating)
 - They wish to get an overview of store locations on the map view
 - Visitors intend to view detailed information and order time estimates for a specific store location
- Customer (Initiating)
 - Visits the site with the intent of:
 - Viewing menu items
 - Ordering through the webpage
 - Making an account
 - Paying for an order
 - Tracking status and delivery
 - Viewing personal order history
 - Leaving a review for items ordered
- Admin (Initiating)
 - Manages store accounts
 - Add, delete, and edit menu items
 - Add, delete, and edit store locations
- Employee (Initiating)
 - Keep track of customer orders
 - Change the status of orders
- Driver (Initiating)
 - View their assigned order information
 - Get directions to drive to the drop-off location
 - Change the status of their delivery order

Third-Party APIs:

- Google Maps (Participating)
 - Display map view, get routes for drivers
- Geolocation (Participating)
 - Get driver location

- Stripe (Participating)
 - Process payment for orders
 - Process refund for orders
- NodeMailer (Participating)
 - Facilitates the email messaging system

Map/Driver - Store locations, Order Tracking and Delivery

Actors: Google Maps, Customer, Driver

Developers: Brandon Cheng, Ji Wu, Damon Lin

Description:

Customers can perform multiple functions such as looking at available store locations and tracking the location of their order once placed. The Google Maps API provides the necessary data for these functionalities to execute. Moreover, employees, who act as drivers, may interact with such data to find the most optimal route to deliver the orders.

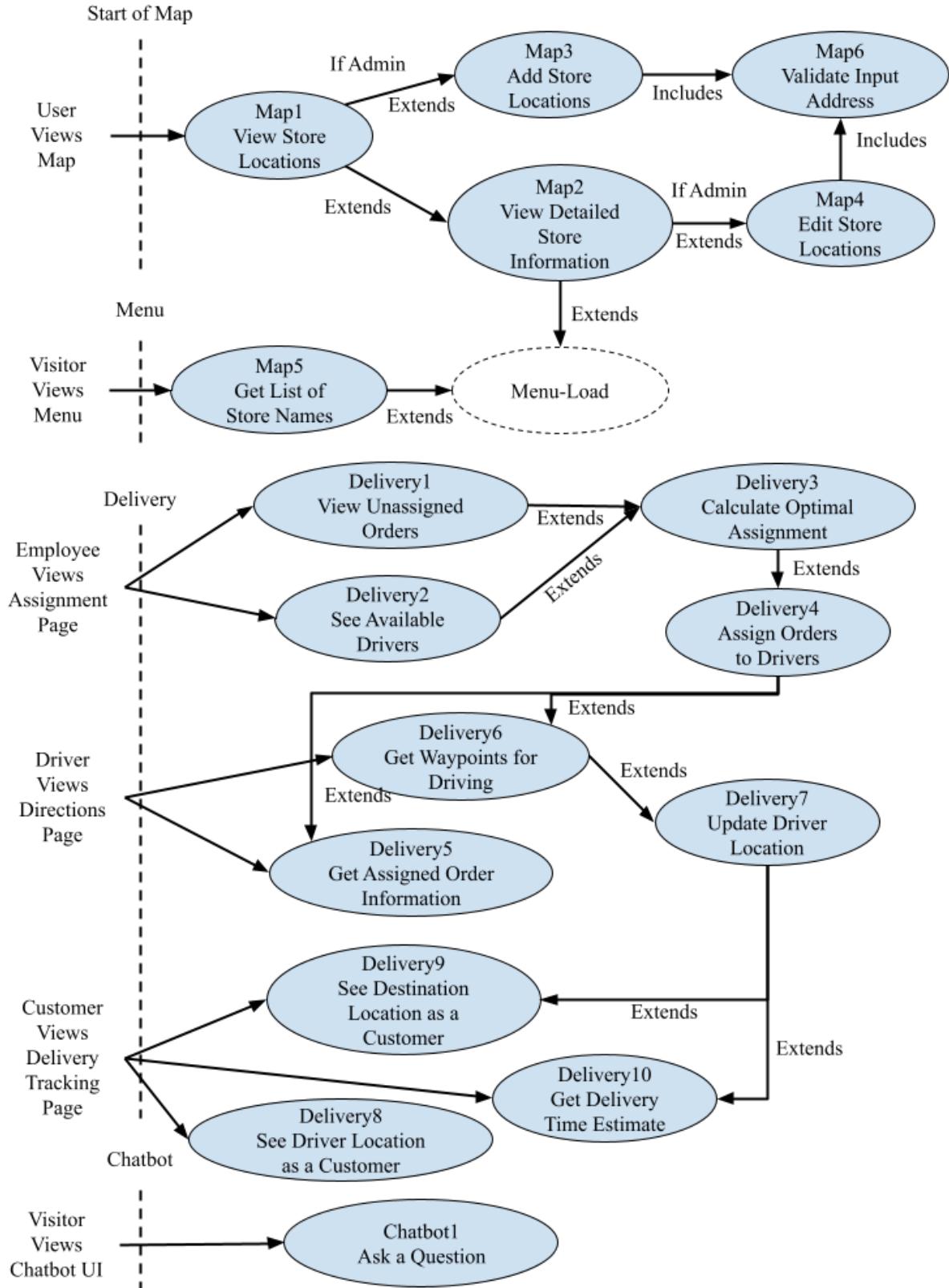
Map/Driver Use Cases:

1. Map1: View Store Locations (Ji Wu)
 - a. Description: This use case can be executed by anyone visiting the Mr.Pizza Website. The visitor will be able to view a map that has all of our store locations on it.
 - b. Preconditions: None, any visitor is able to see this.
 - c. Requirements: **REQ-Visitor-Map-1**.
2. Map2: View Detailed Store Information (Ji Wu)
 - a. Description: This use case can be executed by anyone visiting the Mr. Pizza Website. Once a visitor views a map with our store locations, they can just click on each branch and more detailed information regarding the branch will be displayed.
 - b. Preconditions: None, any visitor is able to see this.
 - c. Requirements: **REQ-Visitor-Map-2**, **REQ-Visitor-Map-3**.
3. Map3: Add Store Locations
 - a. Description: This use case can be executed if you are logged in as an admin. The purpose of this use case is to allow admins to add new store branches if there is expansion within the franchise.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Map-5**
4. Map4: Edit Store Locations
 - a. Description: This use case can only be executed if you are logged in as an admin. The purpose of this use case is to allow admins to edit store information, such as the name of the store or the location of the store.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Map-5**
5. Map5: Get list of Store Names
 - a. Description: This use case allows customers to choose the branch that they want to order from. When they are going to the menu, this list will appear as a dropdown that allows customers to choose the branch of their choice.
 - b. Preconditions: None, any visitor is able to see this and choose a branch.
 - c. Requirements: **REQ-Map-6**
6. Map6: Validate Input Address
 - a. Description: This use case utilizes the Google Maps API to check if an address inputted by the user is an address that exists. This use case is a form of input validation for when admins are adding a new store location.

- b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Map-5, REQ-Order-Creation6**
7. Delivery1: View unassigned orders:
- a. Description: This use case allows employees to obtain a list of all orders that are finished but not assigned to drivers. This use case is necessary for employees to be able to assign delivery orders to drivers.
 - b. Preconditions: Valid Auth Token for the Employee role.
 - c. Requirements: **REQ-Driver-Map-4**
8. Delivery2: See Available Drivers
- a. Description: This use case allows employees to obtain a list of all drivers who are currently idle. This use case is necessary for employees to be able to assign delivery orders to drivers.
 - b. Preconditions: Valid Auth Token for the Employee role.
 - c. Requirements: **REQ-Driver-Map4**
9. Delivery3: Calculate Optimal Assignment
- a. Description: This use case calculates the best assignment of orders to drivers. It runs an algorithm that optimizes the order assignment to drivers and makes the decision for the employee. This effectively removes the geological barrier to order assignment, meaning employees will not need to know where the addresses are or take into account how long the order has been placed for.
 - b. Preconditions: Valid Auth Token for the Employee role. For this use case to be executed, there must be orders that are ready and drivers that are available.
 - c. Requirements: **REQ-Driver-Map-2**
10. Delivery4: Assign Orders to Drivers
- a. Description: This use case is executed by employees when they want to assign orders to drivers.
 - b. Preconditions: Valid Auth Token for the Employee role. For this use case to be executed, there must be orders that are ready and drivers that are available.
 - c. Requirements: **REQ-Driver-Map5**
11. Delivery5: Get Assigned Order Information
- a. Description: This use case is executed by drivers(a type of employee) when they are assigned a batch to deliver. This allows drivers to see order information for the order(s) that they have been assigned to deliver.
 - b. Preconditions: Valid Auth Token for employee (type:driver) role.
 - c. Requirements: **REQ-Driver-Map-3**
12. Delivery6: Get Waypoints for Driving
- a. Description: This use case is executed by drivers in order to get waypoints for driving. This gives drivers the location of the orders for the batch they are assigned.
 - b. Preconditions: Valid Auth Token for employee(type:driver) role. The driver must also be currently sending a delivery order.
 - c. Requirements: **REQ-Driver-Map1, REQ-Driver-Map2**
13. Delivery7: Update Driver Location
- a. Description: This use case is executed by the driver (server runs this automatically once every 10 seconds as the driver is driving). This use case allows customers to see where the driver is in real time(or at most 10 seconds behind).

- b. Preconditions: Valid Auth Token for employee(type:driver) role. The driver must be currently sending a delivery order.
 - c. Requirements: **REQ-Map-4, REQ-Map-3, REQ-Map-2.**
14. Delivery8: See Driver Location as a Customer
- a. Description: This use case allows customers to see where their drivers are in real time. This is one of the most important features for delivery for customers.
 - b. Preconditions: Valid Auth Token for customer role. They also must have placed an order and it needs to be currently being delivered. To execute, customers go from the order status page to the view driver location page.
 - c. Requirements: **REQ-Map-4, REQ-Map3, REQ-Map-2**
15. Delivery9: See Destination Location as a Customer
- a. Description: This use case is executed by customers and it allows customers to see what destination their assigned driver is heading to. Because drivers may be assigned more than one order, it is necessary to let customers know where the driver is headed so as to give a more accurate time estimate. For example, if the driver is assigned two orders and a customer is ten minutes away, their total wait time since the driver leaves may be greater than ten minutes if the driver must send the other order first.
 - b. Preconditions: Valid Auth Token for customer role. They also must have placed an order and it needs to be currently being delivered. To execute, customers go from the order status page to the view driver location page.
 - c. Requirements: **REQ-Map-4, REQ-Map-3, REQ-Map-2.**
16. Delivery10: Get Delivery Time Estimate
- a. Description: This use case is executed by customers and it gives customers a delivery time estimate based on the driver's current location.
 - b. Preconditions: Valid auth Token for customer role. They also must have placed an order and it needs to be currently being delivered. To execute, customers go from the order status page to the view driver location page.
 - c. Requirements: **REQ-Map-4, REQ-Map-3, REQ-Map-2**
17. Chatbot1: Ask a Question
- a. Description: This use case can be executed by anyone who visits the Mr. Pizza website. This provides real-time feedback to common questions that customers may have.
 - b. Preconditions: None, anyone who visits the Mr. Pizza website can access it.
 - c. Requirements: **REQ-Chatbot-1, REQ-Chatbot-2**

Map/Driver Use Case Diagram



Map/Driver Detailed Use Cases

1. Map6: Validate Input Address
 - a. Normal Use Scenario Business Workflow
 - i. Initiating actor: customer gets to order checkout; or initiating actor: employee gets to add/edit store location.
 - ii. The user is prompted with the address validation form.
 - iii. The user inputs Address lines, City, and Zip code into the form.
 - iv. The Request is submitted to the server.
 - v. The server formats the request and sends it to the Google Maps Address Validation API.
 - vi. The server retrieves the response from the API and checks if the information is good.
 - vii. The server will return the information to the user/frontend or submit the information if correct.
 - b. Exceptions to the Workflow
 - i. Customer/Employee is not signed in
 1. The page will not display and will say “Access denied.”
 - ii. The user inputs an address that requires validation
 1. The server will return an address that was automatically completed based on the prior incorrect data requested from the Google Maps Address Validation API.
 2. The user will confirm whether the auto-filled address is correct.
 - iii. The user inputs an address that requires fixes
 1. The server will return the fields that require fixing, which visually display on the frontend.
 - iv. The Google Maps Address Validation API is unavailable
 1. The server will attempt to send a request to the API, returning an error that will be returned to the user.
 - c. Business Concepts
 - i. Restaurant Locations
 - ii. Customer Experience
 - d. Business Operations
 - i. Checkout
 - ii. Store Map
 - e. Business Rules
 - i. Visitors can view stores to choose the store they want to look at on the menu.
 - ii. Must be a valid address within a 7-mile radius of the Mr. Pizza store, if the address is valid, the customer may proceed to payment.
 2. Delivery3: Calculate Optimal Driver Assignment
 - a. Normal Use Scenario Business Workflow
 - i. Initiating actor: employee gets to assign orders to driver. They have already viewed the unassigned orders and available drivers.
 - ii. The employee sends a request to the server, designating which available drivers to assign and the maximum number of orders to assign per driver.

- iii. The server creates a request for a route matrix from the Google Maps Routes API.
 - iv. The server parses the route matrix response to get a 2D array containing the times taken to travel between each location.
 - v. The server calculates all possible ways to group orders into batches.
 - vi. The server calculates all permutations of each batch of orders.
 - vii. The server calculates the score of each permutation, in terms of total squared customer waiting time, accounting for the travel time and delivery ordering. The best permutation for each possible batch is stored.
 - viii. The server returns the grouping and ordering that results in the minimal squared customer waiting time.
 - ix. The Initiating actor receives the assignment of orders to drivers, which is visualized on the order assignment page using Google Maps Directions API.
 - b. Exceptions to the Workflow
 - i. Employee is not signed in
 - 1. The page will not display and will say “Access denied.”
 - ii. There are no orders to assign
 - 1. The page will not display any orders but still loads.
 - 2. If the user attempts to compute the optimal driver assignment, then the server will send the request, returning an error message that the orders array cannot be empty.
 - iii. There are no available drivers
 - 1. The page will not display any available drivers but still loads.
 - 2. If the user attempts to compute the optimal driver assignment, then the server will send the request, returning an error message that the drivers array cannot be empty.
 - iv. The Google Maps Routes API is unavailable
 - 1. The server will attempt to send a request to the API, returning an error that will be returned to the user.
 - c. Business Concepts
 - i. Delivery Protocol
 - d. Business Operations
 - i. Delivery
 - e. Business Rules
 - i. Drivers are to abide by the priority set by the website (overall tardiness over individual order tardiness)
3. Chatbot1: Ask the Chatbot a Question
- a. Normal Use Scenario Business Workflow
 - i. Initiating actor: Visitor will open the home page and the chatbot UI and input a question for the bot.
 - ii. The question is received by the server, which then uses the trained NLP model to classify the question and provide a response.
 - iii. The answer is returned to the user and then displayed on the chatbot UI.
 - b. Exceptions to the Workflow

- i. The model was not trained before the user inputs a question.
 - 1. The server still gets the question and uses the untrained NLP model to classify the question and provide a response.
 - 2. However, the answer will always be “bad” wherein the chatbot cannot help the user with their query.
- ii. The model fails to classify the question
 - 1. The model will simply return “Sorry I cannot help you” with the associated question from the user.
- c. Business Concepts
 - i. Customer Support
- d. Business Operations
 - i. Real-time Support
- e. Business Rules
 - i. The information is limited to that from the report and menu. The Chatbot does not handle any real-time information.

Sequence Diagrams for these detailed use cases are located in Report 2 pt 2: Updated Sequence Diagrams.

Business policies and rule enforcement:

1. All drivers are employees but not all employees are drivers
 - a. The system will give permissions to the employees depending on if they are drivers
2. Drivers will have special access to the map subsystem of the backend site. This will enable them to see map directions, special instructions, and general order information.
 - a. The system will give permissions to access these functionalities if they are drivers
3. Once an employee assigns an order batch to a driver, the status of the order will be changed to “in transit”
 - a. Only admins may alter the order status to “in transit,” “failed,” or “complete.”
4. Drivers will have one of the following statuses: *Idle*, *Delivering*, or *Returning*.
 - a. The system checks that the driver must always be in one of these statuses.
5. Drivers will be tracked in real time, allowing customers to check on their order status and the estimated delivery time.
 - a. The server will update the driver’s location every ten seconds.
6. Customers will receive notifications of their delivery status through a phone call by the driver. This will only happen when the driver needs help reaching the address, when they arrive at the destination, or when the driver cannot make it to the destination due to unforeseen circumstances.
 - a. The driver will only have access to the customer’s number when sending the order.
7. Orders will only be accepted if the address is within 7 miles of the Mr. Pizza branch.
 - a. If an address is further than 7 miles, then an employee will call the customer to alert them of the cancellation.

Policy and Rule Awareness

1. Employees will be given a PDF of business rules and policies before being hired. They must confirm that they are aware and understanding of the rules.

2. Customers are able to get rules and policies about the map/driver use cases from the Mr. Pizza Chatbot, which is trained about all the relevant information about business policies..

Use Case Violations

1. Order is not delivered due to extenuating circumstances (traffic accidents, etc).
 - a. Customers will be notified via a phone call and they will also be able to view their updated status order on the website in the status panel.
 - b. Admin/Employees will be notified by Order Dashboard
2. Google Map API is down
 - a. The store locations screen will be replaced with a maintenance screen.
 - b. Drivers will use Apple Maps and domain knowledge of surrounding areas and driving experience to navigate.

Pizza Order Pipeline

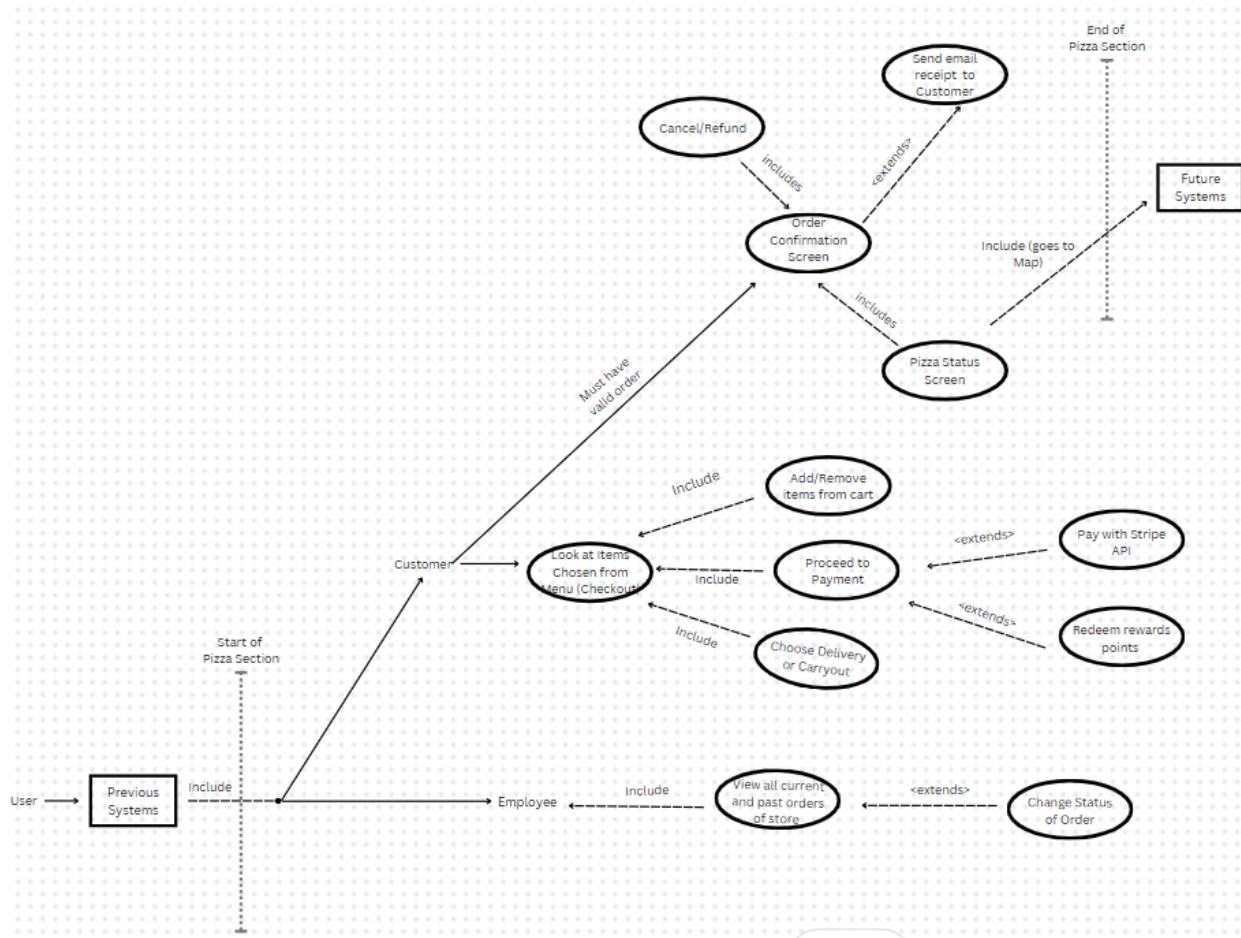
Actors: Customer, Employee, Stripe (Participating)

Developers: Arya Shetty, Vineal Sunkara, Nikash Rajeshbabu

Description:

Customers will be able to place the order of items that they have already selected. This order will be processed and paid for using the Stripe API. This order will be stored in a database, which will keep track of every order placed at each store. This will allow the employees to make, access, and possibly deliver their orders. Customers can also see the status of their order, as well as all orders they have ever placed. They can also submit a small review for items within previous orders that they have placed.

Pizza Pipeline use case diagram:



Pizza Pipeline Use Cases:

- *Order Payment* (Requirements: REQ-Order-Creation-1, REQ-Order-Creation-4, REQ-Menu-Cart-1, REQ-Customer-Rewards-1, 2, 3, 4) (Arya)
 - Preconditions: User must be signed in as customer, user must have created cart and proceeded to the order payment menu.

- Users must select whether the order is carryout or delivery.
 - *Input Delivery Address*
 - *Apply Rewards Points*
 - *Add Gratuity Points*
 - Customers can also return to the menu to edit their items or return to the home page.
 - Customer can then press Checkout to open Stripe Payment Menu
 - **Business Goal:** The order process should be as straightforward and streamlined as possible provided it is within operating hours. Customers can easily select delivery and carryout status, and apply their earned reward points. They can also begin the checkout process.
- *Input Delivery Address* (Requirements: REQ-Order-Creation-1, REQ-Order-Creation-6) (Vineal)
 - Preconditions: User must be signed in as a customer, user must have created a cart and proceeded to the order payment menu, user must have selected Delivery option.
 - After the customer selects the Delivery option, a form is opened where the customer can input a desired, valid address.
 - The form has several fields for the address.
 - Country and State pre-filled since Mr. Pizza operates in New Jersey exclusively.
 - Customers are required to fill in address line 1, city, zip code as accurately as possible.
 - Based on the information provided, the valid address is immediately or the best result is displayed, with an additional confirmation.
 - Customer can finalize the address input with a confirmation button
 - **Business Goal:** The address input validation should be as clear as possible to the customer to prevent confusion between the customer and the business.
- *Apply Rewards Points* (Requirements: REQ-Order-Creation-1, REQ-Order-Creation-7) (Vineal)
 - Preconditions: User must be signed in as a customer, user must have created cart and proceeded to the order payment menu, user must have 5 rewards points
 - Once a customer has 5 points, they can hit the apply rewards button.
 - This will waive the cost of the most expensive item in their cart. If that item has more than one quantity, only one will be waived.
 - The customer can then proceed as normal, with the savings being viewable on the orderStatus screen.
 - **Business Goal:** Rewards points are used to incentivize more orders from customers, as a free item can be redeemed once 5 orders have been placed.
- *Add Gratuity Tip* (Requirements: REQ-Order-Creation-1) (Vineal)
 - Preconditions: User must be signed in as customer, user must have created cart and proceeded to the order payment menu.
 - Customers have the ability to add a 15% tip to the order total.
 - **Business Goal:** Allows customers to tip the Mr. Pizza restaurant.
- *Stripe Payment* (Requirements: REQ-Order-Creation-1, REQ-Order-Creation-4, REQ-Order-Creation-5) (Arya and Vineal)

- Preconditions: User must be signed in as a customer, user must have created cart and proceeded to the order payment menu, user must have selected either Delivery or Account, user must have decided whether or not to use rewards points or not, user must have decided to add gratuity tip or not.
- Customers (initiating) input credentials to pay for order, then press the checkout button to begin the checkout process.
- A Stripe Payment menu will open up below.
 - The total is dependent on whether there is a tip or not.
 - Customer will be asked for the following information
 - Email
 - Credit Card Information
 - Card Holder Name
 - Zip Code
 - Optional: Customers can input their phone number to save their credentials to a Stripe Link, which saves the information for them.
 - Customers can then hit Pay on the Stripe Checkout Menu in order to fulfill the payment and be taken to the order status screen.
- The order information is updated in the database and marked as paid if stripe checkout is successful.
- Order confirmation is sent to the customer's email saved in the account information.
- **Business Goals:** Using the stripe API will allow the API itself to handle payment verification and confirmation, in order to both prevent any data liability regarding the customer's payment information and to allow the order to be processed and started more quickly.
- *View Order Status* (Requirements: REQ-Order-Status-1, REQ-Order-Status-2) (Nikash, Vineal, Arya)
 - Preconditions: User must be signed in as a customer, User must have already paid successfully using the Stripe API
 - Customer is now on the orderStatus page, where they can see the current status of the order they just paid for.
 - They can also see the amount saved when using reward points.
 - They can also return to home or interact with *Delivery-8* (See Map/Driver section for more information).
 - *Cancel Order*
 - *Refund Order*
 - **Business Goals:** Allows the customer to see the status of their order in real time, which allows for better transparency and less confusion as to the location and status of the order. The statuses are designed to be informative. Customers should be able to determine exactly what stage their order is in.
- *Cancel Order* (Requirements: REQ-Order-Creation-5, REQ-Order-PostProcess-1) (Nikash and Arya)
 - Preconditions: User must be signed in as a customer, User must have already paid successfully using the Stripe API, order must not have proceeded past the Processing/Paid status.

- Customers can press the cancel button on the orderStatus page, where they are notified that the order is canceled.
 - Customer is emailed with the notification that their order has been canceled.
 - **Business Goals:** Allows the customer to cancel the order if they realize a mistake has been made or they changed their mind in some way. This can only be done when order is still processing, in order to prevent the customer from attempting to cancel too late in the order process and causing wasted time for the employee and/or driver.
- *Refund Order* (Requirements: REQ-Order-Creation-5, REQ-Order-PostProcess-1) (Nikash and Vineal)
 - Preconditions: User must be signed in as a customer, User must have already paid successfully using the Stripe API, order must not have proceeded past the Processing/Paid status.
 - Customers can press the cancel button on the orderStatus page, where they are notified that the order is canceled.
 - Customer is emailed with the notification that their order has been canceled.
 - **Business Goals:** Allows the customer to cancel the order if they realize a mistake has been made or they changed their mind in some way. This can only be done when order is still processing, in order to prevent the customer from attempting to cancel too late in the order process and causing wasted time for the employee and/or driver.
- *View All Store Orders* (Requirements: REQ-Order-Management-1, REQ-Order-Creation-2) (Vineal and Arya)
 - Preconditions: User must be signed in as employee, ideally there would be orders to populate the page.
 - Employees can view the past orders for only the store they are assigned to.
 - They are able to view relevant information such as the store and order id of the order, date created, total price, the item number of a specific item, price of that item, and the item name.
 - They can now change the status to one of the following statuses depending on the state of the order. This will be updated accordingly for the customer.
 - **Business Goal:** This is to allow the customer to view the status in real time, as well as the employee to keep track of the state of orders currently sent to the store. It is useful to be transparent with the status of the order in order for the customer to not have any confusion, as well as allowing smooth service.
- *View All Customer Orders* (Requirements: REQ-Order-Management-2) (Arya)
 - Preconditions: User must be signed in as customer, ideally there would have already been orders to populate the page.
 - Customers can view the past orders that they have placed.
 - They are able to view relevant information such as the order id of the order, status, date created, total price, the item number of a specific item, price of that item, item name, and the current review for the item.
 - Customers can also type a small review for a specific item, and then submit it.
 - This can only be done on a specific order that is tied to a customer's account and checkout id.

- **Business Goal:** Customers should be able to view their order history at Mr. Pizza for their own reference, as well as be able to place a small review for specific items.

Pizza Pipeline detailed use case scenarios:

For each use case, we have already covered preconditions and business goals, so we will examine the workflow here, business policies and rule enforcement and use-case violations are below.

- *Order Payment*
 - Workflow
 - Firstly, the customer needs to be signed in as a registered customer, and must have prepared a cart full of items they would like to order.
 - They can do this by clicking on the view menu button, then selecting a store.
 - A list of items is displayed, where customers can select a range of available products.
 - Customers can also search for a desired product by clicking the search icon.
 - Customers can finally click the cart button to view their selected items, then the checkout button to proceed to the order payment page.
 - This will be detailed further in the Menu Section.
 - The customer is then taken to the order payment page, where they can see the list of items that were present in their cart. They can also see the toppings, size, and extras selected, as well as any surcharges resulting from that. They can also choose the quantity of a specific menu item, in case they selected more than one. They can finally see the price per item and total cost of that item, with the total order price being in the last row of the table.
 - They can also see their current rewards points. This will never go below 0 or above 5. If the rewards points are at 5, they can then make the most expensive item free of charge, and if that item has a quantity more than one, then only one instance of that item can be redeemed.
 - The customer can also choose whether to make the order a delivery or checkout.
 - If customer chooses delivery:
 - The address input form will be opened, where the customer can input their information. The address has several fields that can be inputted, as well as two fields that are already pre filled due to Mr. Pizza being only served in the state of New Jersey.
 - Country: US
 - State: New Jersey
 - Street Address 1 (required)
 - Street Address 2
 - City

- Zip Code
 - After the address is inputted, the form will correctly set the address or find the best possible match. If it needs to find the best possible match, it will ask the customer before confirming so the customer has time to re-input their address.
 - If customer chooses carryout:
 - There is no need for further action, this is the default state of the order.
- The customer has the option to add a 15% gratuity tip, which is a simple button.
 - When activated, an additional charge is added to the stripe menu. This will be displayed as an additional surcharge below the actual total. The new price is updated and displayed on that menu. This is the amount that will be charged to the customer's card.
- The customer also has the option to return to the menu, in case they changed their mind about the items they selected.
- The customer can also return to home if they would like to do another action on the Mr. Pizza website.
- Once the customer hits the checkout button, they will be shown the Stripe Payment Menu.
- Note that by default, the order is considered as a carryout order with no tip and no rewards points applied.
 - It is at the discretion of the customer to update these values based on their current needs.
- Exceptions to Workflow
 - Customer not signed in
 - Page will not display, will say access denied.
 - Customer has not made an account with Mr Pizza
 - Either the customer is not signed in as a customer, has no account, or has not signed in at all.
 - In this case, the customer is allowed to build a cart, but if they try to proceed to checkout, they will be told that they do not have access until they have signed in.
 - Menu is not accurate or updated properly
 - If there is some issue with web scraping the menu, the menu may not display properly, leaving the user unable to select items.
 - Items currently in order will not be available to purchase, which will prompt users to select new items.
 - Customer tries to apply rewards points when they do not currently have 5 rewards points
 - The customer will be notified that they need 5 rewards points to apply.
 - The customer is then allowed to proceed with normal checkout, and they will be notified every time they try to apply the rewards points.

- Customer tries to switch to an alternate type of order rapidly.
 - Any time the carryout option is clicked, the default order type is set, which is the carryout type with no tip or rewards points activated.
 - Customer can then proceed as normal
- Customer does not properly input address properly when selecting the order as delivery
 - The customer will be notified to make sure they have filled out all the required fields for the address input form.
 - If they have done so, then the form itself will do its best to find the best address depending on the input the customer has input.
 - For example, if 220 Moorvin Lane, Piscatooway, 00852 is inputted, then the form will ask if 220 Marvin Lane, Piscataway, 08854 is the right address, with the option to re-input the address or select the suggestion as the correct address.
- Business Concepts
 - REQ-Order-Creation-1, REQ-Order-Creation-4, REQ- Menu-Cart-1, REQ-Customer-Rewards-1, 2, 3, 4
 - This case is responsible for streamlining the checkout process, which occurs after the desired list of items is selected but before the payment is processed with Stripe API. Note that the cart functionality and search functionality (where customers can search for desired items in the menu) will be elaborated further in the menu section.
- Business Rules and Policies
 - Customers must have a registered account.
 - Food must be part of the menu, the stock on certain items will vary and display accurately during ordering.
 - This will be handled by the web scraping
 - Customers must input address if order is delivery
 - If a customer does not have a proper address they can send the delivery too, they will not be able to properly receive the order.
 - Rewards points can only be earned by placing orders, where every order placed is one reward point, up to 5. No points can be gained past 5. Once rewards points have been applied, the total resets to 0.
 - Customers are not required to tip, but Mr. Pizza appreciates the courteous gesture.
- View Order Status
 - Workflow
 - Once the customer has successfully paid with Stripe, they are now forwarded to the orderStatus page, where they can see the current status of the order they just paid for.
 - On this page, they can see the exact list of items that they have purchased, as well as the fact that their rewards points counter has gone up thanks to them successfully placing an order.

- There will also be an image on the right which shows the current status of the order.
 - Every status has a corresponding image
 - Depending on the type of the order, the image will have additional future statuses that will be displayed.
 - For example, if the order is a delivery, the customer will be able to see additional statuses like In-Transit, whereas this would be hidden if the order is a carryout.
 - The status image will update every interval of about 5 to 10 seconds, so the status will essentially be in real time.
- If the customer wants to cancel or refund their order, they can go through the *Cancel Order and Order Status*.
 - They can also return to home or interact with *Delivery-8 (See Map/Driver section for more information)*.
- Exceptions to Use Case
 - Customer not signed in
 - Page will not display, will say access denied.
 - Stripe had some failure, but still allowed the order to go through
 - Order will be given the status of Processing, and it is up to the Employee to examine the payment and see if there are any errors with Stripe API before marking the order as Started.
 - Status failed to update
 - The default status of Processing is assigned should there be any errors with status assignment
 - Order is unreasonable in nature or has critical errors
 - Employees have the right to reject it and allow customers to cancel/refund.
- Business Concepts
 - REQ-Order-Status-1, REQ-Order-Status-2
 - This case is responsible for displaying the order status to the customer, as well as providing them various options about their order.
- Business Rules and Policies
 - Customers must have a registered account.
 - If the order is not picked up by the end of the current day's store hours, it will be marked as rejected and the order becomes unavailable to pick up.
 - If the delivery has failed to deliver for some reason (missing customer, no one to pick up order at the address provided), the order will be marked as rejected.
 - If the order is not fulfilled by Mr.Pizza within reasonable time, the customer has the option to refund the order if it is still processing.
 - If the order is not picked within a reasonable time by the customer, the customer will still be charged full price.
 - If the order seems questionable in nature, the employee has the right to reject the order. Ex. 9999 Pizzas ordered, or orders worth tens of thousands of dollars.

- Only the employee assigned to the store where the order has been placed can verify and accept the order. If the order is not accepted due to extenuating reasons, customers will be notified of this.
- *View All Store Orders*
 - Workflow
 - Assuming there are already orders placed, customers are allowed to view the orders they have placed in the past.
 - From the main menu of the Mr. Pizza website, customers can click on Customer Order History.
 - From there, they are able to view relevant information such as the order id of the order, status, date created, total price, the item number of a specific item, price of that item, item name, and the current review for the item.
 - If there is no review, the customer will be prompted to input a review for the item, then submit it so it can be reviewed and inserted into the database.
 - Customers can also type a small review for a specific item, and then submit it.
 - This can only be done on a specific order that is tied to a customer's account and checkout id.
 - Customers can also see the status of each order that was placed.
 - Exceptions to Use Case
 - Customer not signed in
 - Page will not display, will say access denied.
 - Customers has no orders placed before
 - There will simply be no orders present on the page
 - Customers wants to submit another review after submitting one already
 - Customers can submit another review, and the most recent one will display. However, all reviews are stored for quality assurance and purposes of Mr. Pizza in order to gauge items.
 - Business Concepts
 - Requirements: REQ-Order-Management-1, REQ-Order-Creation-2
 - This case is responsible for displaying the entire order history of the customer.
 - Business Rules and Policies
 - Customers must have a registered account.
 - If a customer has any enquiries about their previous order, they can submit a help request. This will be elaborated on in the customer accounts section.

General Business policies and rule enforcement:

- Customers must have a registered account.
- Food must be part of menu, the stock on certain items will vary and display accurately during ordering
- If Stripe API Fails, users will be prompted to try again in a few minutes.
- For registered customers, the card and delivery info is already saved but there is another option to change card or delivery info
- Registered customers have the option to view past orders, past orders are also displayed on order menu for registered customers to easily add past items to cart

- Registered and guest customers both have the option to view order status, and estimated delivery time is provided to customers.
- Orders will have a unique tag, and only once payment is verified through Stripe API, it will be queued to write to the database.
- Admin's have the optional ability to use the information from the order database to compile store statistics, track revenue, and perform additional business analysis on the entire chain or on a local store level. This will be elaborated further in the customer accounts section.
- Customers are able to get rules and policies about the pizza pipeline use cases from the Mr. Pizza Chatbot, which is trained about all the relevant information about business policies.

General Use Case Violations

1. Stripe API is down
 - a. The checkout page will be replaced with maintenance page
 - b. Customers won't be able to order while stripe is down
2. Node Mailer API is down
 - a. The store will call the customer if necessary for confirmation
 - b. They should still receive credit card payment confirmation by checking their bank/card activity.

Sequence Diagrams for these detailed use cases are located in Report 2 pt 2: Updated Sequence Diagrams.

Menu Administration & Selection

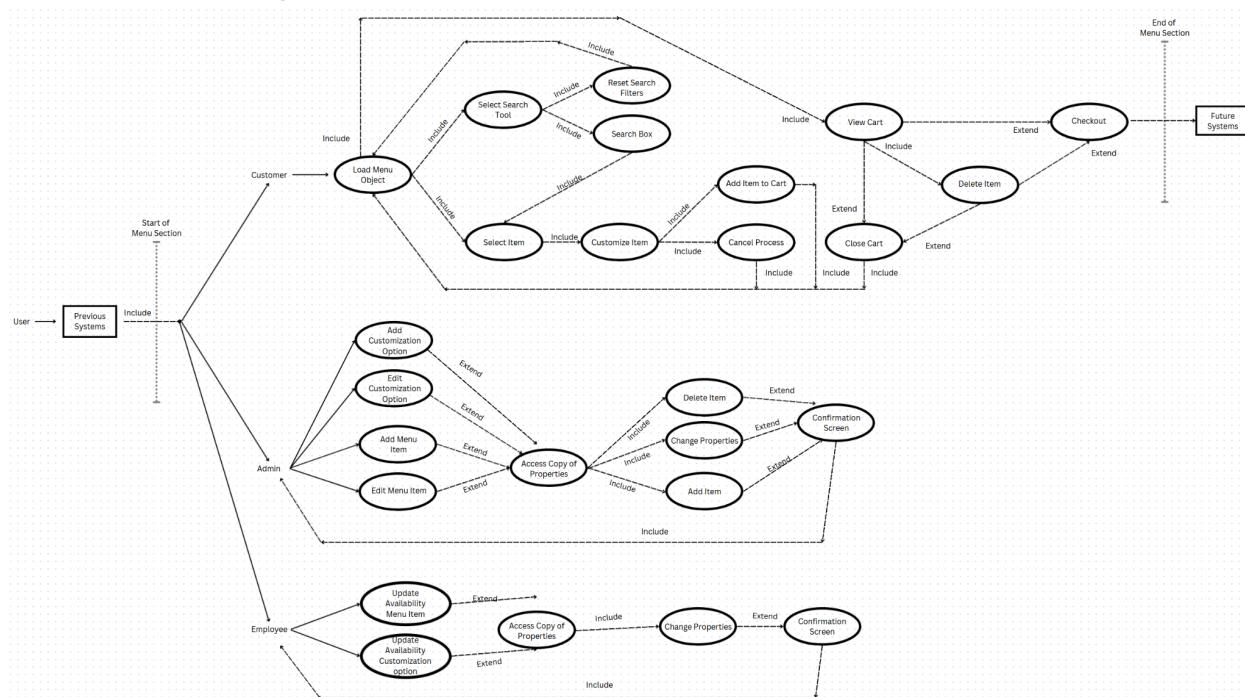
Actors: Customer and Admin

Developers: Keanu Melo Rojas, Joshua Menezes, Thomas O'Connell

Description:

Customers will be able to freely navigate the original menu or a modified version. This, with the help of a search engine that can redesign the menu to their needs. Furthermore, the Customer will be able to select menu items and access a customization menu for the respective item. Finally, the customer will have the option to add the item to their cart or cancel the process and be taken back to the menu to view more items. Admin will be able to customize and update the menu to account for current availability of menu items and ingredients. The admin will be able to give menu items a select number of customization options as to limit ingredients and toppings

Menu use case diagram:



Menu Use Cases

1. Menu-Load: Load Menu

- When directed to the menu page, the customer can view the menu for the chosen store. The Customer will be able to choose between selecting a search tool or selecting an item.
- Precondition: Customer would have to select a store, or can be redirected after selecting a location in map, in order to load a menu.
- Requirements: **REQ-Menu-Load-1.**

2. Admin-Add-1: Add New Items

- a. The admin will be given an option to add an item to the menu on the admin account. They can then choose a category the item goes under, give a basic description of the item, include a picture, add a price, and create a basic list of what is included on or with the item.
 - b. Precondition: Admin would have to click on the “Add Item” button.
 - c. Requirements: **REQ-Menu-Items-1**, **REQ-Menu-Options-2**
3. Admin-Delete-1: Delete Menu Items
- a. Precondition: Admin would have to select an item and select the “Delete Item” button.
 - b. The admin will be given an option to delete an existing menu item so that it is removed from the menu in its entirety.
 - c. Requirements: **REQ-Menu-Items-1**
4. Admin-Edit-1: Limit Customization
- a. The admin can select a menu item and can choose what customization options are included for that item. The admin can choose what ingredients to include in the customer’s customization that can be added to an item, which ingredients the customer can remove from the original menu item, and what ways the customer can have their food prepared (dietary restrictions/substitutions)
 - b. Precondition: Admin would have to click on the item they want to add customization to.
 - c. Requirements: **REQ-Menu-Options-1**, **REQ-Menu-Options-2**
5. Employee-Availability-1: Update Item Availability
- a. The employee will be able to update items on the menu for when there is a shortage of an item or an ingredient used to make an item. They can choose to mark an item as unavailable and remove missing or out of stock ingredients from the customization options.
 - b. Precondition: Employees would have to click on the item they want to update and click the “Update Item” button.
 - c. Requirements: **REQ-Menu-Items-2**, **REQ-Menu-Options-1**, **REQ-Menu-Load-1**

Menu Detailed Use Cases

1. Admin-Add-1: Add new items
- a. Normal Use Scenario Business Workflow
 - i. Initiating actor: admin goes to the “Admin Menu Tool” on the home page.
 - ii. The admin is prompted with the “Add Item” form.
 - iii. The Request is submitted to the server.
 - iv. If successful, the new item gets added to the database.
 - v. Different locations may have different items
 - b. Exceptions to the Workflow
 - i. Admin is not signed in
 - 1. The request will not be sent and a message of “You are not signed in.”
 - ii. Admin inputs invalid fields for the item
 - 1. The request will not be sent
 - c. Business Concepts

- i. Product Sales
 - ii. Restaurant Locations
 - d. Business Operations
 - i. Updating Menu
 - e. Business Rules
 - i. Admins must add items that are reasonable and have been approved
2. Employee-Availability-1: Update Item Availability
- a. Normal Use Scenario Business Workflow
 - i. Initiating actor: employee goes to the “Employee Menu Tool” on the home page.
 - ii. The employee is prompted with the “Update Item” form.
 - iii. The Request is submitted to the server.
 - iv. If successful, the item’s availability gets updated on the customer’s menu
 - v. Each location has its own updates.
 - b. Exceptions to the Workflow
 - i. Employee is not signed in
 - 1. The request will not be sent and a message of “You are not signed in.”
 - ii. Employee inputs invalid fields for the item / item does not exist
 - 1. The request will not be sent
 - c. Business Concepts
 - i. Restaurant Locations
 - ii. Customer Experience
 - d. Business Operations
 - i. Updating Item Availability
 - e. Business Rules
 - i. Employees must update the availability of items as soon as possible

Business policies and rule enforcement:

1. Admins will be provided with guidelines for adding new items, including categorization, descriptions, and image requirements.
2. Admin can make changes to the properties of the menu, but employees can not
 - a. Admin must log in with admin account in order to access these options
 - b. Admin can add and delete items from the whole menu/database
 - c. Admins will be guided on setting customization options within defined parameters, such as ingredient availability or pricing guidelines.
3. Employees can make updates to item and topping availability
 - a. Employees must log in with employee account to access these options
 - b. Only updates what is seen on the menu for customers, does not add/items from menu database
4. The admin or employee must input a store ID so that changes are made to a specific location’s menu
5. The menu loaded by the customer only shows available items from the selected store location.

Policy and Rule Awareness:

1. Employees will be informed of any restrictions or guidelines for updating menu items, such as price adjustments or ingredient changes.
2. Admins will be informed of any implications of deleting menu items, such as customer impact or inventory adjustments
3. Customers are able to get rules and policies about the menu use cases from the Mr. Pizza Chatbot, which is trained about all the relevant information about business policies.

Use Case Violations:

1. Database was not loaded correctly
 - a. Menu cannot be loaded without connecting to database
2. Login information was not correct / logged into the wrong account
 - a. Admins, employees, and customers all have different permissions and tasks that can be performed on the account
3. No available menu items for a given store
 - a. Information about no items being available for a given store will be displayed in place of the menu.

Sequence Diagrams for these detailed use cases are located in Report 2 pt 2: Updated Sequence Diagrams.

Customer account, feedback, and analytics (Customer experience)

Actors: Customer, Employees, Administrators

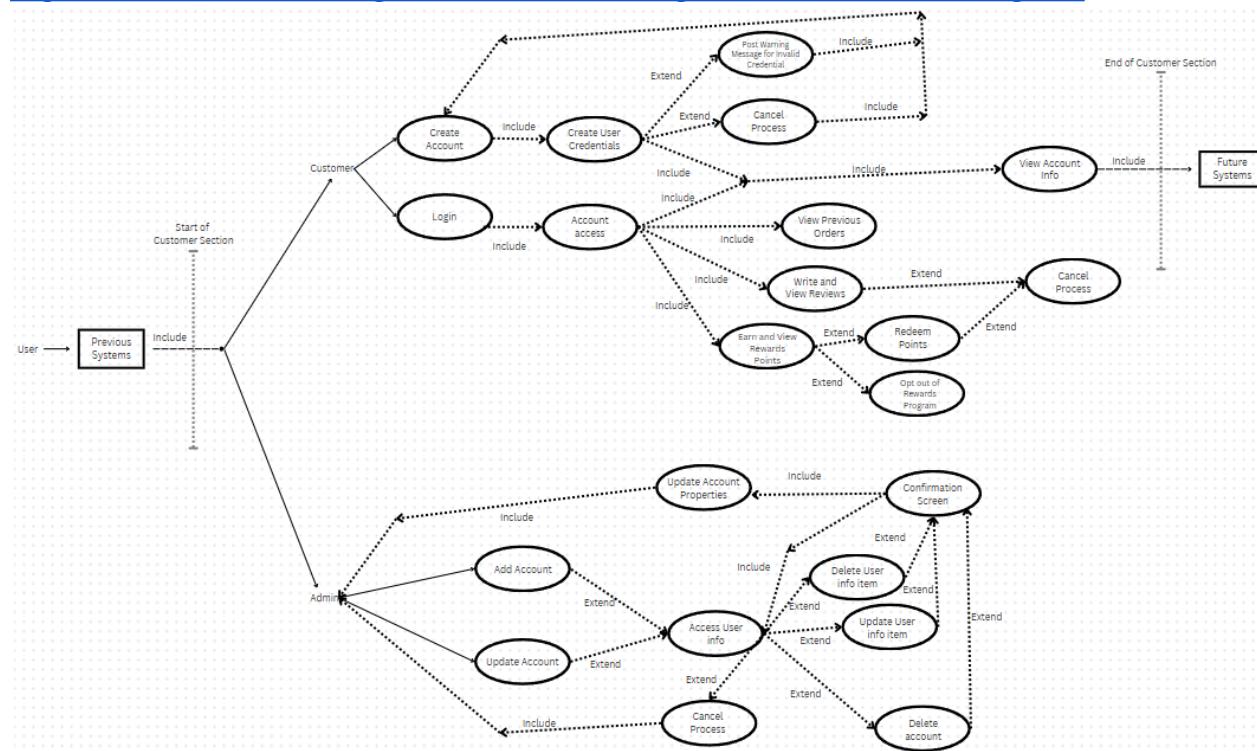
Developers: Ethan Sie, Anna Yeakel, Aman Patel

Description:

Customers will be given the option to be able to create an account with Mr. Pizza, to gain access to an ability to save and view previous orders, as well as participate in a rewards program. The created account will be stored in a MySQL database to keep track of customer information. The rewards program allows customers to earn points with each order, until eligible for a reward, generating customer loyalty and engagement. Customers will also be given the ability to leave feedback for ordered food, to provide employees with an opportunity to look at any areas in need of improvement or areas that are excelling.

Customer Accounts use case diagram:

<https://www.canva.com/design/DAF9HF35XaU/Q5glT7RAHo6eKCIIdG1lZVg/edit>



Accounts Use Cases:

1. Auth1: Create Customer Account
 - a. Description: Anyone visiting the Mr. Pizza website can register and create a customer account.
 - b. Preconditions: None, anyone visiting the website can create a customer account
 - c. Requirements: **REQ-Customer-Account1**
2. Auth2: Customer Login
 - a. Description: Anyone with a registered customer account can login to their account to have access to Mr. Pizza amenities.

- b. Preconditions: Must have a registered customer account.
- c. Requirements: **REQ-Customer-Account2**
- 3. Auth3: Employee Login
 - a. Description: If you are an employee with Mr. Pizza, you can sign into your employee account to have access to employee information.
 - b. Preconditions: Must be a valid employee with Mr. Pizza
 - c. Requirements: **REQ-Employee-Account1**
- 4. Auth4: Admin Login
 - a. Description: If you are an admin with Mr. Pizza, you can sign into your admin account to have access to admin functionalities
 - b. Preconditions: Must be a valid admin with Mr. Pizza
 - c. Requirements: **REQ-Admin-Account6**
- 5. Auth5: Get Auth Role
 - a. Description: Fetches the authentication role of the user that is signed into either a customer, employee, or admin account.
 - b. Preconditions: Must be signed into a valid Mr. Pizza account.
 - c. Requirements: **REQ-Customer-Account2**, **REQ-Admin-Account6**, **REQ-Employee-Account1**
- 6. Auth6: Logout
 - a. Description: Logs out of a signed-in Mr. Pizza account
 - b. Preconditions: Must be signed into a valid Mr. Pizza account
 - c. Requirements: **REQ-Accounts1**
- 7. Auth7: Get Employee Store Info
 - a. Description: Get information about the store the employee is assigned to
 - b. Preconditions: Must be signed into a valid employee account
 - c. Requirements: **REQ-Employee-Account1**
- 8. Accounts1: Get Customer Info
 - a. Description: Fetches a logged in customer's account information.
 - b. Preconditions: Must be logged into a valid customer account
 - c. Requirements: **REQ-Customer-Account2**
- 9. Accounts2: Edit Customer Info
 - a. Description: Updates a logged in customer's account information.
 - b. Preconditions: Must be logged into a valid customer account
 - c. Requirements: **REQ-Customer-Account5**
- 10. Accounts3: Delete Customer Account
 - a. Description: Deletes a valid customer's account from the Mr. Pizza customer account database table.
 - b. Preconditions: Must be logged into a valid customer account
 - c. Requirements: **REQ-Customer-Account6**
- 11. Accounts4: Add New Employee
 - a. Description: Adds a new employee to the Mr. Pizza employee database
 - b. Preconditions: Must be logged into a valid admin account
 - c. Requirements: **REQ-Admin-Account7**
- 12. Accounts5: Assign Existing Employee
 - a. Description: Assigns a valid employee from the Mr. Pizza database to a new store
 - b. Preconditions: Must be logged into a valid admin account

- c. Requirements: **REQ-Admin-Account7**
- 13. Accounts6: Get Employee Info
 - a. Description: Fetches a valid, logged in, employee's account information.
 - b. Preconditions: Must be logged into a valid employee account.
 - c. Requirements: **REQ-Employee-Account1**
- 14. Accounts7: Edit Employee Info as Admin
 - a. Description: Edits a valid employee's account information as an admin.
 - b. Preconditions: Must be logged into a valid admin account.
 - c. Requirements: **REQ-Admin-Account7**
- 15. Accounts8: Edit Employee Info as Employee
 - a. Description: Edits a valid employee's account information as an employee.
 - b. Preconditions: Must be logged into a valid employee account.
 - c. Requirements: **REQ-Employee-Account2**
- 16. Support1: Create Support Ticket
 - a. Description: This use case can be executed by any customers visiting the Mr. Pizza website. The customer will be able to create a support ticket to ask a question about the website.
 - b. Preconditions: Valid Auth Token for the Customer role.
 - c. Requirements: **REQ-Customer-Support1**
- 17. Support2: Respond to Support Ticket
 - a. Description: This use case can be executed by any employees using the Mr. Pizza website. The employee will be able to respond to an unanswered support ticket which the customer will be able to see.
 - b. Preconditions: Valid Auth Token for the Employee role.
 - c. Requirements: **REQ-Employee-Support1**
- 18. Support3: View All Tickets
 - a. Description: This use case can be executed by any employees using the Mr. Pizza website. The employee will be able to see all previous support tickets.
 - b. Preconditions: Valid Auth Token for the Employee role.
 - c. Requirements: **REQ-Employee-Support1**
- 19. Support4: View Past Tickets
 - a. Description: This use case can be executed by any customers visiting the Mr. Pizza website. The customer will be able to see their past support tickets including employee responses if there are any.
 - b. Preconditions: Valid Auth Token for the Customer role.
 - c. Requirements: **REQ-Customer-Support2**
- 20. Support5: View Unanswered Tickets
 - a. Description: This use case can be executed by any employees using the Mr. Pizza website. The employee will be able to see tickets that do not currently have a response.
 - b. Preconditions: Valid Auth Token for the Employee role.
 - c. Requirements: **REQ-Employee-Support1**
- 21. Analytics1: Get Total Revenue By Date
 - a. Description: Gets the total company revenue from a specific time period.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics1**

- 22. Analytics2: View Employee Count By Store
 - a. Description: View the Employee count by their store/location.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics3**
- 23. Analytics3: View Total Customers
 - a. Description: View the total amount of current customers.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics4**
- 24. Analytics4: View Total Employees
 - a. Description: Views the total amount of employees.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics5**
- 25. Analytics5: View Menu Items Sorted By Popularity
 - a. Description: Sorts menu items by the total amount of orders made for that specific item.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics6**
- 26. Analytics6: View Revenue Total Per Menu Item
 - a. Description: Tells the total revenue generated by a specific menu item.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics7**
- 27. Analytics7: View Toppings Sorted By Popularity
 - a. Description: Views the most popular toppings by amount ordered.
 - b. Preconditions: Valid Auth Token for the Admin role.
 - c. Requirements: **REQ-Analytics8**

Accounts Detailed Use Cases

- 1. Accounts1: View Customer Info
 - a. Normal Use Scenario Business Workflow
 - i. Initiating actor: customer gets to Mr. Pizza website login page.
 - ii. Customer prompted to provide username and password
 - iii. Authentication process submitted to the server
 - iv. Server validates the provided credentials and logs the customer into their account
 - v. On logging into the account, the server fetches the customer's account information from their cid value.
 - vi. Server returns the fetched account information and displays it to the logged in customer
 - b. Exceptions to the Workflow
 - i. Customer provides incorrect credentials
 - 1. Server will alert the customer of incorrect username or password
 - ii. Customer not logged in
 - 1. The page will not display and will say "You are not signed in"
 - c. Business Concepts
 - i. Account Access

- ii. Authentication
 - d. Business Operation
 - i. Input Credentials
 - ii. Validate Credentials
 - iii. Access Profile
 - e. Business Rules
 - i. Customer account must be a valid account in the Mr. Pizza database.
2. Support1: Create a Support Ticket
- a. Normal Use Scenario Business Workflow
 - i. Initiating actor: customer gets to Mr. Pizza “Need Help?” button in profile page.
 - ii. Customer prompted to enter a description of their question
 - iii. Description gets added to a newly-created database entry for the ticket
 - iv. Customer has a view of their previous support tickets with a response section containing an employee response if there is one (Support4 use case)
 - b. Exceptions to the workflow
 - i. Customer is not signed in
 - 1. Returns a 401 error code saying “You are not signed in”
 - c. Business Concepts
 - i. Account Access
 - ii. Customer Support
 - d. Business Operation
 - i. Input credentials
 - ii. Validate credentials
 - iii. Take description input
 - e. Business Rules
 - i. Customer must have a valid account in the Mr. Pizza database
 - ii. Customer must be signed in
3. Analytics1: View Total Company Revenue
- a. Normal Use Scenario Business Workflow
 - i. Initiating actor: Admin opens up the Mr.Pizza website login page.
 - ii. Admin prompted to provide username and password
 - iii. Authentication process submitted to the server
 - iv. Server validates the provided credentials and logs the admin into the account
 - v. Upon logging in Admin clicks on ‘Store Analytics’
 - vi. Then specifies a specific time frame using the calendar options for a beginning point and endpoint
 - vii. Server returns the fetched revenue information for how much revenue was generated during that time period.
 - b. Exceptions to the workflow
 - i. Customer attempts to login
 - ii. Returns a 401 and a message saying Authorization denied

- c. Business Concepts
 - i. Account Access
 - ii. Authentication
 - iii. Company revenue
- d. Business Operation
 - i. Input Credentials
 - ii. Validate Credentials
 - iii. Revenue generated

Business policies and rule enforcement:

- Passwords will be encrypted to provide security
- Users must have an account to leave a food review
- Users must be signed in to accrue reward points (PP)
- Should a rule be broken by an admin, the user(s) affected will be notified of suggested actions via email
- Customers are able to get rules and policies about the account use cases from the Mr. Pizza Chatbot, which is trained about all the relevant information about business policies.

Overall Contributions Table			
	Member 1	Member 2	Member 3
Subteam 1	Ji Wu(33.33)	Damon Lin(33.33)	Brandon Cheng(33.33)
Subteam 2	Arya Shetty (33.33%)	Nikash Rajeshbabu (33.33%)	Vineal SUNKARA (33.33%)
Subteam 3	Keanu Melo Rojas (33.33%)	Joshua Menezes (33.33%)	Thomas O'Connell (33.33%)
Subteam 4	Aman Patel (33.33%)	Anna Yeakel (33.33%)	Ethan Sie(33.33%)

Report 2 Pt 1: Domain Models

Actor Roles

The actor roles that will be accessing our APIs will be Visitors, Customers, Employees, and Admins.

Map/Driver Sub-group Domain Model:

API Operations

- Map1-View Store Locations
 - Entities: Stores (read)
 - Services & Validation: Google Maps API will be used to display the map of store locations
- Map2-View Detailed Store Information
 - Entities: Stores (read)
 - Services & Validation: None
- Map3: Add Store Locations
 - Entities: Stores (write)
 - Services & Validation: Google Maps Address Validation API will be used to validate that the inputted address is valid. Admin authentication will be validated.
- Map4: Edit Store Locations
 - Entities: Stores(write)
 - Services & Validation: Google Maps Address Validation API will be used to validate that the inputted address is valid. Admin authentication will be validated.
- Map5: Get List of Store Names
 - Entities: Stores (read)
 - Services & Validation: None
- Map6: Validate Input Address
 - Entities: None
 - Services & Validation: Google Maps Address Validation API will be used to check the validity of the inputted address. This will sort the inputted address into the accepted, verify_required, and fix_required. We will also ensure that the desired address is limited to the US and NJ, as outlined in our business policies.
- Delivery1:View Unassigned Orders
 - Entities: Orders(read)

- Services & Validation: Google Maps Routes API will be used to approximate the time it will take to drive to each unassigned order. Employee authentication and store assignment will be validated.
- Delivery2: See Available Drivers
 - Entities: Employee Accounts(read), Delivery Batches(read)
 - Services & Validation: Employee authentication and store employment will be validated. Validates availability of drivers based on participation in current deliveries and store assignments.
- Delivery3: Calculate Optimal Assignment
 - Entities: Employee Accounts(read), Orders(read), Stores(read)
 - Services & Validation: Google Maps Routes API will be used to calculate the route matrix for trips between the orders and the store. The server will use an algorithm to calculate the optimal assignment and ordering of deliveries to return the assignment that aligns with our business goals. Employee authentication and store assignment will be validated. Driver availability and order status will be validated.
- Delivery4: Assign Orders to Drivers
 - Entities: Employee Accounts(write), Delivery Batches(write)
 - Services & Validation: Employee authentication and store employment will be validated. Driver availability and store assignment will be validated.
- Delivery5: Get Assigned Order Information
 - Entities: Employee Accounts(read), Delivery Batches(read), Orders(read)
 - Services & Validation: Google Maps API will be used to display the map of order locations. Employee authentication, driver status, and store employment will be validated.
- Delivery6: Get Waypoints for Drivers
 - Entities: Employee Accounts(read), Delivery Batches(read), Orders(read)
 - Services & Validation: Google Maps API will be used to display the driving waypoints and human-readable directions. Employee authentication, driver status, and store employment will be validated.
- Delivery7: Update Driver Location
 - Entities: Employee Accounts(read), Delivery Batches(write)
 - Services & Validation: The Mozilla Geolocation API will be used to determine the driver's location. Employee authentication, driver status, and store employment will be validated.
- Delivery8: See Driver's Location
 - Entities: Orders(read), Delivery Batches(read)
 - Services & Validation: The Google Maps API will be used to display the location on a map. Order ownership and status will be validated.

- Delivery9: See My Delivery Destination (Customer)
 - Entities: Orders(read)
 - Services & Validation: The Google Maps API will be used to display the location on a map. Order ownership and status will be validated.
- Delivery10: Get Delivery Time Estimate
 - Entities: Orders(read), Delivery Batches(read)
 - Services & Validation: The Google Maps Routes API will be used to calculate the time estimate between the driver's current position and the order's delivery address. Order ownership and status will be validated.
- Chatbot1: Ask the Chatbot a Question
 - Entities: Chatbot Data(read)
 - Services & Validation: NLP.js will be used to interpret the user's question. This will ensure that the question is a valid question and respond accordingly, or respond appropriately if the question is invalid.

Example Data Generation for Collections

- Stores
 - We have a set of hand-picked stores for our example store data.
- Delivery Batch and Orders
 - We have a script that generates random orders with various items, times, locations, and statuses. From these example orders, we can assign them into batches using Delivery3: Calculate Optimal Assignment and Delivery4: Assign Orders to Drivers.
 - Once the orders are assigned to a driver, we can use our driving simulation script to simulate the driver's position approaching the customer's location
 - We can view the updating delivery position as a customer in real-time.
- Chatbot Data
 - Chatbot training data comes from two sources: processed report information and menu data.
 - Processed report information is generated by using a Generative Pretrained Transformer model with our report as the input, which generates a corpus file of questions, intent classifications, and answers.
 - Menu data is generated through our scraped menu data from real-world pizza websites, which is then formatted into a corpus file of questions, intent classifications, and answers.

Pseudocode

```
Pseudo Code for validating the store input address:

// User gets to one of the predefined scenarios that prompts them to fill out the address validation form
// The request is sent to the backend which calls this function:
function validateAddress(req, res) {
    // retrieves the information from the address validation form and checks if the formatting is correct
    if missing_fields || !correctRequestBody+ContentType
        return;

    // formats the quest to send it to the Google Maps Address Validation API
    let requestBody = {
        // formats it
    }

    // calls the runValidation function to send it to the API
    let res = runValidation(requestBody);

    return res;
}

function runValidation (requestBody) {
    // fetches the response from Google Maps Address Validation API
    let res = fetch

    // checks for the verdict for bad fields and returns the bad ones if any
    if ( verdict.hasBadProperties() ) {
        output = { some properties }
    }

    return { good or require verification }
}
```

This pseudocode utilizes the Google Address Validation API to check whether the address entered by the user is valid, and the server will return the status of the validation request.

Pseudocode for assignment of order to drivers:

```
if addressIsValid:  
    return "Invalid Address"  
if addressIsValid:  
    nextDriver = None  
    longestWaitTime = -1  
    for driver in drivers:  
        if driver.status == 'idle' and driver.waitTime > longestWaitTime:  
            longestWaitTime = driver.waitTime  
            nextDriver = driver  
  
    if nextDriver is not None:  
        # Proceed with assigning the nextDriver  
        return nextDriver  
    else:  
        return "No available driver"
```

This pseudocode picks the next driver based on who has been “idle” for the longest time after first verifying that the address is correct.

Pseudocode for asking the chatbot a question:

```

// Assuming the admin already trained the chatbot with information from the menu and report
and that the user opens the chatbot UI
// Any message the user inputs is passed as a request to the backend into this function:
function handlerChatbotQuestion(req, res) {
    // retrieves the information from the request and checks if the formatting is correct
    if missing_fields || !correctRequestBody+ContentType
        return;

    let question = get question from request

    // uses the trained NLP model to classify the question and provide a response
    res = nlp.process(question)

    // checks if the response provides no help and returns
    if (res == "I cannot help")
        return res

    // otherwise just return the response
    return res
}

```

This pseudocode utilizes the NLP module to train the model and classify the user's question and generate a response to their question.

Detailed Central Use-Case: Delivery3

Related Requirements	REQ-Driver-Map-2, REQ-Driver-Map-3
Initiating Actor	Any of type Employee
Actor's Goal	To successfully fetch the optimal driver assignment for the available drivers and unassigned orders using the Google Maps API.
Participating actors	Employee, Google Maps Routes API
Preconditions	There are available drivers and unassigned orders in the database, and there exists an optimal assignment that minimizes overall tardiness.

Post Conditions		The optimal driver assignment is returned such that the overall tardiness is optimized.
Flow of Events for Main Success Scenario		
1.	The employee sends a request to the server, designating which available drivers to assign and the maximum number of orders to assign per driver.	
2.	The server creates a request for a route matrix from the Google Maps Routes API.	
3.	The server parses the route matrix response to get a 2D array containing the times taken to travel between each location.	
4.	The server calculates all possible ways to group orders into batches.	
5.	The server calculates all permutations of each batch of orders.	
6.	The server calculates the score of each permutation, in terms of total squared customer waiting time, accounting for the travel time and delivery ordering. The best permutation for each possible batch is stored.	
7.	The server returns the grouping and ordering that results in the minimal squared customer waiting time.	
8.	The Initiating actor receives the assignment of orders to drivers, which is visualized on the order assignment page using Google Maps Directions API.	

Pseudocode for central use case:

```

function optimalAssignment(req, res, storeId) {
    // retrieves the information from the request and checks if the formatting is correct
    if missing_fields || !correctRequestBody+ContentType
        return;

    // checks that the drivers array are non-empty
    if (drivers is empty)
        return

    // ensures that drivers are available and exist
    availableDrivers = queryAvailableDrivers(storeId)
    for i : length(availableDrivers) {
        checks that drivers[i] = availableDrivers[i]
        returns if not
    }
}

```

```
// max number of orders; maxPerDriver is passed through the JSON body
maxOrders = length(drivers)*maxPerDriver

// gets the unassigned orders
orders = queryUnassignedOrders(storeId, maxOrders)

// gets the route matrix from the queried orders
routeMatrix = queryRouteMatrix(storeId, orders)

calculates all the assignments through some assign function such that each driver will try to
reach maxOrders

routeMatArr = []
for i : length(routeMatrix) {
    extracts the time estimates and stores it in routeMatArr
}

// optimizes overall tardiness over individual tardiness
calculates the table of route permutations to customer delay for all the sets that we need

return output containing the best assignment for the drivers and orders
}
```

Pizza Ordering Pipeline Sub-Group Domain Model:

API Operations

- Places an order from items in cart
 - Entities: Store (Write), Order (Write), Customer Account (Write), Employee Account (Write)
 - Services & Validation:
 - Customer will be taken to the Order Confirmation page when payment is processed successfully and receipt is emailed to Customer.
 - Stripe API will handle the payment processing and make sure inputted payment information is valid.
 - Node Mailer will be responsible for sending a receipt to the Customer.
 - Need to be logged in as customer
- View an order's status
 - Entities: Order (Read)
 - Services & Validation:
 - Need to be logged in as customer
 - Need to have successfully paid for order
- View current and completed orders of specified customer
 - Entities: Customer Account (Read)
 - Services & Validation:
 - Need to be logged in as customer
- Cancel an order
 - Entities: Store (Write), Order (Write), if order is delivery, Delivery Batch (Write), Customer Account (Write), Employee Account (Write)
 - Services & Validation: Stripe API will handle payment cancellation and refunds. These will follow previous business rules.
 - The customer will be immediately charged after placing the order to deter fake orders, but users are still able to fully refund as long as it is before the order is marked as **in transit** or **ready**. If it is after this stage, they will have to contact Mr. Pizza to receive a refund.
- (Employees) View all orders for store
 - Entities: Store (Read)
 - Services & Validation: Logged in as employee
- (Employee) Can set status for orders for their current store
 - Entities: Store (Write), Order (Write), Customer Account (Write), Employee Account (Write)
 - Services & Validation: Logged in as employee
- (Employee) Can reject the customer's order due to certain circumstances
 - Entities: Store (Write), Order (Write), Customer Account (Write), Employee Account (Write)
 - Services & Validation: Logged in as employee
- (Admin) Compile information from database for statistics
 - Entities: Store (Read), Customer Account (Read), Employee Account (Write), Review (Read), Help Ticket (Read)

- Services & Validation: Logged in as admin

Example Data Generation for Collections

- Stores
 - Automatically generated list of addresses/locations around New Jersey for the store locations.
 - We will use sample images and information associated with each location.
- Customer and Employee Accounts
 - We will generate a list of customer and employee accounts for the database to have on file, to use for test orders.
- Menu Items
 - We will create a predefined list of all menu items, for customer accounts to be able to choose and order from, these items are webscraped
 - Each menu item will have images, descriptions, and prices.
- Order Database
 - We will have a development script that randomly generates orders from registered example customer accounts.
 - Each order will go through certain stages of the pizza ordering pipeline. Some will make it all the way through to delivery while others may be canceled or refunded based on the randomization of the script. This process will show how the customer payment process and employee order creation process will be handled.
 - Our database will also include past order history for certain registered customers

Pseudocode

```
Pseudocode for placing an order:

if StripeApiPaymentProcessing
    placeOrder(creditCardLast4Digits, totalPrice, isDelivery, deliveryAddress, cart)
    //deliveryAddress will not have a value if this is not delivery order
else
    return;

function placeOrder (creditCard, totalPrice, isDelivery, deliveryAddress, DTCreated, cart) {
    const order = {
        creditCard = this.creditCard
        status = pending
        if isDelivery
            deliveryAddress = this.deliveryAddress
        DTCreated = Date.now() //get current timestamp
        orderId = generateOrderId() //generates a unique id for order
        cart = this.cart()
    }
    //sends to Database
    Try {
```

```

sendOrderQuery = INSERT INTO Orders (credit, status, DTcreated, oid, cart)
VALUES (order.credit, order.status, order.DTCreated, order.orderID, order.cart)
result = await quer(sendOrderQuery)
}
Catch { (error)
    OUTPUT error sending query
    Status(500)
}

}

```

Pseudocode for View all orders

```

if (viewAll = "selected")
    Try {
        viewAllQuery = SELECT * FROM Orders JOIN Customers on Orders.cid =
        customers.cid;
        result = await query(viewAllQuery);
        response.status(200);
        response.write(JSON(result));
    }
    Catch {
        OUTPUT error sending query
        Status(500)
    }
else
    return;

```

Detailed Central Use-Case - Payment Menu:

Use-Case	Payment Menu
Description	After confirming their order through the cart, the customer will access the payment menu. Here, they will be prompted to enter their payment information. They will also be able to see their order in a simple text format. Some of this functionality will be handled by the Stripe API, such as verification of valid payment info and actual payment processing. The customer may then input this information, use previously saved info if they have an account, or return to the cart if they

	would like to change their order. After inputting the payment information, customers can hit a Pay Now With Stripe interactable element. If it succeeds, customers will be taken to the Order Confirmed menu and the order will be sent to the database.
Related Requirements	REQ-Order-Creation-1: As a customer, I can create an order of menu items on the website to have the food items delivered or picked up. REQ-Order-Creation-4: As a customer I can pay for my order through the Stripe API.
Initiating Actor	Customer (any User who is placing an order)
Actor's Goal	To confirm and pay for their order by inputting payment information.
Participating Actors	Customer Stripe API Node Mailer API
Preconditions	Cart must have at least one item that can be paid for by rewards or online payment. Customers can only progress to the payment menu from the cart menu.
Postconditions	If in cart and the customer clicks on checkout, will progress to payment page Will progress to the order confirmed page if and only if Stripe API successfully verifies and processes the payment and the order is sent to the database. Will send notification/receipt to customer through Node Mailer API
Flow of Events for Main Success Scenario	<ol style="list-style-type: none"> 1. Customer confirms order 2. Customer views payment menu <ul style="list-style-type: none"> a. Can return back to cart, if so, repeat step 1. 3. Customer double-checks order, then inputs payment information. 4. Customer double-checks payment information, then confirms payment with Stripe API after checking out (through some interactable element).

	<ol style="list-style-type: none"> 5. Stripe API confirms payment info and processes payment. 6. Order is updated to paid in the database 7. Customer is sent a receipt through email. 8. Customer taken to order confirmed screen where they can view status.
--	--

```

import StripeApi
import NodeMailerApi
/*
★ Main function that will begin the display of the payment menu.
★ Will be run after customer confirms order in the cart menu
*/
function main() {
    try:
        //Will display Payment Menu Window, which contains a preview of the order: Step 1-2
        displayPaymentMenuWindow();
    catch Exception as e:
        // Handle any errors that occur while displaying the quantity selection window
        logError(e);
        displayErrorMessage("An error occurred while trying to display the payment menu.
Please try again later.");
    finally:
        /* Sends a receipt and allows the customer to proceed to order confirmation. Since this is
technically not part of payment menu, we will assume these methods work as described Steps
6-8 */
        proceedToOrderConfirmation();
    }

/*
★ Display main payment menu window, which creates a item preview, and also calls
Stripe API to create a checkout menu.
*/
function displayPaymentMenuWindow() {
    try:
        //Code will iterate through order and list preview of items. Step 3.
        for item in order {
            itemPreview = item.getItemName() + item.getItemQty() + item.getItemPrice()
            orderPreview.push(itemPreview)
        }
    catch abnormalItemQtyPrice:
        logError(e);
}

```

```

        displayErrorMessage('Item and qty price abnormally high.');
    catch Exception as e:
        logError(e);
        displayErrorMessage('There was an error displaying the order preview. Please return to
        cart and try again.');
    finally:
        //Code will create a mini menu where payment information can be inserted by the user.
        /* We will use the prebuilt checkout function found in the StripeAPI to insert and verify
        to do this. For the purpose of this pseudocode, we will consider this as some function that
        handles all the work. This will include inserting payment information and processing payment
        information. https://docs.stripe.com/payments/checkout */ Step 3-5
        callStripeAPICheckout();
    }

function callStripeAPICheckout() {
    //code will create a new checkout session where it will be embedded into the payment info
    page. It will take some parameters, which will be verified using the API. If customer has
    account and accessed the payment menu with an account, they will have option to auto-fill
    their information
    createNewCheckoutSession {
        firstname:
        middlename:
        lastname:
        email:
        deliveryAddress:
        cardNumber:
        expDate:
        cvc:
    }
    try:
        if (customer has account) {
            promptUserUseSavedInformation();
            createNewPayNowButton();
        }
    catch Exception as e:
        logError(e);
        displayErrorMessage("Stripe API error. Please try again in a few minutes.");
    }

    //Code will send the confirmed payment order to the database. It will then prompt the user to
    enter the order confirmation page, where they will be able to see the status of their order, check
    their possible delivery, and possibly refund their order
    function proceedToOrderConfirmation(order, paymentConfirmation) {
        if (payment confirmation == true ) {
            //Order database will be initialized somewhere else
            //The database will be stored and queried through mysql, var mysql = require('mysql');
            con.connect(function(err) {

```

```

if (err) throw err;
console.log("Connected!");
var sql = order;
con.query(sql, function (err, result) {
  if (err) throw err;
  console.log("1 confirmed order entered");
});
});
}
else {
  displayErrorMessage("Transaction not successful");
}
//Since this is technically not part of payment menu, we will assume this methods work as
described Step 8 */
displayOrderConfigurationMenu();
}

```

Menu Sub-Group Domain Model:

API Operations

- (Visitor) Load Menu
 - Entities/Resources: Menu (read)
 - Services: None
- (Visitor) Restart Filtering
 - Entities/Resources: Menu (read)
 - Services: None
- (Employee) Update Availability
 - Entities/Resources: Menu Item (write), Toppings (write)
 - Services: None
- (Admin) Add Menu Item
 - Entities/Resources: Menu Item (write), Toppings (write)
 - Services: None
- (Admin) Update Menu Item (delete, toppings,price)
 - Entities/Resources: Menu Item (write), Toppings (write)
 - Services: None

Example Data Generation for Collections

- Menu Items and Customizations

- Menu Data will be generated by using a web scraping bot on a real-life online pizza store. This bot will use two main steps, which are repeatedly alternating between using Puppeteer to interact with the live webpage, and using Cheerio to parse the html contents of the page.
- To determine the items available, we will iterate through all the categories of the menu and record the entries for each category.
- To read the prices for all of the items from this pizza website, we will need to add all of the items to the cart and check the checkout page to read the prices.
- To figure out the customizations available for each item and the prices associated with each one, we will need to read all the options from the “edit item” page on the checkout screen, and then try each option and record the price change that occurs from trying each option.
- Menu Item
 - Generation: Menu items will be populated from a predefined list taken from a web scraper.
 - MID: A unique identifier that will be persisted in the database to keep track of all the items Mr. Pizza offers in its menu.
 - Name: A string with the item’s name.
 - Price: A value with the cost of the item.
 - Image: URL or file path to the image of the item.
 - Description: A brief description of the menu item including the name, ingredients, calories, and style.
 - Category: A string with the item’s unique category.
 - Available: A boolean value that represents the availability of an item in the specified store.
 - Toppings: JSON of available toppings for the item.
 - Size: JSON of available sizes for the item.
 - Serving Options: JSON of available serving options for the item.
 - Sauces: JSON of available sauces for the item.
 - Crust: JSON of available crust for the item.
 - Meats: JSON of available meats for the item.
 - Non-meats: JSON of available non-meats for the item.
 - Cheese: JSON of available cheeses for the item.
 - Example:
 - MID: 66
 - Name: ExtravaganZZa
 - Price: A value with the cost of the item.
 - Image:
https://cache.dominos.com/olo/6_130_2/assets/build/market/US/_en/images/img/products/larges/S_ZZ.jpg
 - Description: Pepperoni, ham, Italian sausage, beef, fresh onions, fresh green peppers, fresh mushrooms and black olives, all sandwiched between two layers of provolone and cheese made with 100% real mozzarella.
 - Category: Specialty Pizzas
 - Available: 1
 - Toppings: []

- Size: [[Object], [Object], [Object]]
 - Serving Options: []
 - Sauces: []
 - Crust: [[Object], [Object], [Object]]
 - Meats: [[Object], [Object], [Object]]
 - Non-meats: [[Object], [Object], [Object]]
 - Cheese: [[Object], [Object], [Object]]
- Topping
 - Generation: Toppings will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the topping.
 - MID: Id of the topping.
 - Price: Cost of the topping.
 - Available: Availability of the topping.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:
 - Name: No American Cheese
 - MID: 62
 - Price: 0
 - Available: 1
 - Mutual Exclusive: 0
 - isDefault: 0
- Sizes
 - Generation: Sizes will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the size.
 - MID: Id of the size.
 - Price: Cost of the size. Negative value if the size is smaller than default for future operations.
 - Available: Availability of the size.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:
 - Name: 10" Small
 - MID: 77
 - Price: -5
 - Available: 1
 - Mutual Exclusive: 1
 - isDefault: 0
- Serving Options
 - Generation: Serving options will be populated from a predefined list taken from a web scraper.
 - Attributes:

- Name: Name of the serving option.
 - MID: Id of the serving option.
 - Price: Cost of the serving option.
 - Available: Availability of the serving option.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
- Example:
 - Name: 16-Piece
 - MID: 20
 - Price: 12
 - Available: 1
 - Mutual Exclusive: 1
 - isDefault: 0
- Sauces
 - Generation: Sauces will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the sauce.
 - MID: Id of the sauce.
 - Price: Cost of the sauce.
 - Available: Availability of the sauce.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:
 - Name: Alfredo Sauce
 - MID: 56
 - Price: 0
 - Available: 1
 - Mutual Exclusive: 1
 - isDefault: 1
- Crusts
 - Generation: Crusts will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the crust.
 - MID: Id of the crust.
 - Price: Cost of the crust.
 - Available: Availability of the crust.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:
 - Name: Gluten Free Crust
 - MID: 66
 - Price: 0
 - Available: 1
 - Mutual Exclusive: 1

- isDefault: 0
- Meats
 - Generation: Meats will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the meat.
 - MID: Id of the meat.
 - Price: Cost of the meat.
 - Available: Availability of the meat.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:
 - Name: Bacon
 - MID: 66
 - Price: 2
 - Available: 1
 - Mutual Exclusive: 0
 - isDefault: 0
- Non-Meats
 - Generation: Non-meats will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the non-meat.
 - MID: Id of the non-meat.
 - Price: Cost of the non-meat.
 - Available: Availability of the non-meat.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:
 - Name: Banana Peppers
 - MID: 69
 - Price: 2
 - Available: 1
 - Mutual Exclusive: 0
 - isDefault: 0
- Cheeses
 - Generation: Cheeses will be populated from a predefined list taken from a web scraper.
 - Attributes:
 - Name: Name of the cheese.
 - MID: Id of the cheese.
 - Price: Cost of the cheese.
 - Available: Availability of the cheese.
 - Mutual Exclusive: Exclusiveness of object with objects of its same type.
 - isDefault: Value that assigns the default topping for menu item.
 - Example:

- Name: Extra
- MID: 71
- Price: 2.5
- Available: 1
- Mutual Exclusive: 1
- isDefault: 0

Pseudocode:

```

function loadMenu(storeId):
    try:
        // Retrieve menu items for the specified store ID from the database
        menuItemsJson = getMenuItemsFromDatabase(storeId)

        // Parse the JSON data to get menu items
        menuItems = parseJson(menuItemsJson)

        // Display menu on the website
        displayMenuOnWebsite(availableItems)
    catch Exception as e:
        // Handle any errors that occur during menu loading
        displayErrorMessageOnWebsite("An error occurred while loading the menu: " + e.message)

function getMenuItemsFromDatabase(storeId):
    try:
        // Connect to the database
        databaseConnection = connectToDatabase()

        // Execute SQL query to retrieve menu items for the specified store ID
        queryResult = executeQuery(databaseConnection, "SELECT * FROM menu_items WHERE
store_id = ?", storeId)

        // Convert query result to JSON format
        menuItemsJson = convertQueryResultToJson(queryResult)

        // Close the database connection
        closeDatabaseConnection(databaseConnection)

        return menuItemsJson
    catch Exception as e:
        // Handle any errors that occur while retrieving menu items from the database
        throw e

function filterAvailableItems(menuItems):
    // Filter out unavailable items and get only available toppings for each available item

```

```

availableItems = []
for item in menuItems:
    if item.available:
        availableToppingsJson = filterAvailableToppings(item.toppings)
        availableItems.push({"description": item.description, "MID": item.MID, "price": item.price, "image": item.image, "toppings": availableToppingsJson})
return availableItems

function filterAvailableToppings(toppings):
    // Filter out unavailable toppings
    availableToppings = []
    for topping in toppings:
        if topping.available:
            availableToppings.push({"name": topping.name, "price": topping.price})
    return availableToppings

```

Detailed Central Use-Case:

Central Use-Case (M-1)	Load Menu
Description	Customers will load the original Mr. Pizza's menu upon entry and when redirected by another process or by a button. For loading the menu, the program should extract the available menu items from the database, update items with available customization, and then display these selectable items on screen with an image and description.
Related Requirements	REQ-Menu-Options4, REQ-Menu-Search1
Initiating Actor	Customer
Actor's Goal	View the menu including all the available items.
Pre-Conditions	Click on the menu button or be redirected to the menu from another process.
Post-Conditions	Search or select available menu item
Flow of Events	<ol style="list-style-type: none"> Extract available items. Update the items with their available customization options.

	3. Display the menu items on screen for the customer to view and select.
--	--

Customer Sub-Group Domain Model:

API Operations

- Auth1: Create Customer Account
 - Entities: Customer accounts (write)
 - Services: None
- Auth2: Customer Login
 - Entities: Customer accounts (read)
 - Services: Bcrypt will be used to check the cryptographic hash of the password, Cookie and JWT will be used to store the authentication token.
- Auth3: Employee Login
 - Entities: Employee accounts (read)
 - Services: Bcrypt will be used to check the cryptographic hash of the password, Cookie and JWT will be used to store the authentication token.
- Auth4: Admin Login
 - Entities: Admin accounts (read)
 - Services: Bcrypt will be used to check the cryptographic hash of the password, Cookie and JWT will be used to store the authentication token.
- Auth5: Get Auth Role
 - Entities: Admin accounts (read), Employee Accounts (read), Customer Accounts (read)
 - Services: Cookie and JWT will be used to read the authentication token.
- Auth6: Logout
 - Entities: None
 - Services: Bcrypt will be used to check the cryptographic hash of the password, Cookie and JWT will be used to store the authentication token.
- Auth7: Get Employee Store Info
 - Entities: Employee accounts (read)
 - Services: Cookie and JWT will be used to read the authentication token.
- Support1: Create Support Ticket
 - Entities: Support ticket (write), Customer Accounts(read)
 - Services: Support ticket ownership will be validated.
- Support2: Respond to Support Ticket
 - Entities: Support ticket (write), Employee Accounts(read)
 - Services: None
- Support3: View All Tickets
 - Entities: Support ticket (read), Employee Accounts(read)
 - Services: None
- Support4: View Past Tickets
 - Entities: Support ticket (read), Customer Accounts(read)
 - Services: Support ticket ownership will be validated.
- Support5: View Unanswered Tickets
 - Entities: Support ticket (read), Employee Accounts(read)
 - Services: None
- Accounts1: Get Customer Info

- Entities: Customer Accounts (read)
 - Services: Account ownership will be validated.
- Accounts2: Edit Customer Info
 - Entities: Customer Accounts (write)
 - Services: Account ownership will be validated.
- Accounts3: Delete Account
 - Entities: Customer Accounts (write)
 - Services: Account ownership will be validated.
- Accounts4: Add new Employee
 - Entities: Emmployee Accounts (write)
 - Services: Admin authentication will be validated.
- Accounts5: Assign Existing Employee
 - Entities: Emmployee Accounts (write)
 - Services: Admin authentication will be validated.
- Accounts6: Get Employee Info
 - Entities: Emmployee Accounts (read)
 - Services: Employee account ownership will be validated.
- Accounts7: Edit Employee Info as Admin
 - Entities: Emmployee Accounts (write)
 - Services: Admin authentication will be validated.
- Accounts8: Edit Employee Info as Employee
 - Entities: Emmployee Accounts (read)
 - Services: Employee account ownership will be validated.
- Analytics1: Get Total Revenue By Date
 - Entities: Orders(read)
 - Services: None
- Analytics2: View Employee Count By Store
 - Entities: Orders(read)
 - Services: None
- Analytics3: View Total Customers
 - Entities: Orders(read)
 - Services: None
- Analytics4: View Total Employees
 - Entities: Orders(read)
 - Services: None
- Analytics5: View Menu Items Sorted By Popularity
 - Entities: Orders(read)
 - Services: None
- Analytics6: View Revenue Total Per Menu Item
 - Entities: Orders(read)
 - Services: None
- Analytics7: View Toppings Sorted By Popularity
 - Entities: Orders(read)
 - Services: None

Example Data Generation for Collections

- Customer Accounts
 - We will have a script that will create randomly generated customer accounts with names, emails, and usernames.
- Analytics Data
 - Our script will also generate hundreds of example orders, each with randomly selected menu items and customizations. The prices for all of these orders will be calculated accordingly, and this data will be available for querying in the database for when we run our analytics. With this example data, we can test our business analytics to sort items, view statistics about revenue and account numbers.
- Reviews
 - We will have example reviews that can be viewed and responded to.

Pseudocode

```

function handleCreateProfileButtonClick():
try:
    // Display account creation form
    displayAccountCreationForm()
except Exception as e:
    // Handle any errors that occur while displaying the account creation form
    logError(e)
    displayErrorMessage("An error occurred while trying to display the account creation
form. Please try again later.")

function handleAccountCreationFormDataSubmission(customerData):
try:
    // Validate the customer's input data
    validateInputData(customerData)
    // Create a new customer profile in the database
    customerId = createCustomerProfile(customerData)
    // Send a confirmation email to the customer
    sendConfirmationEmail(customerData.email)
    displayConfirmationMessage("Account successfully created. A confirmation email has
been sent.")
except InputDataValidationException as e:
    // Handle input data validation errors
    logError(e)
    displayErrorMessage("Invalid input data: " + e.message)
except EmailAlreadyExistsException as e:
    // Handle the case where the email is already in use
    logError(e)
    displayErrorMessage("The email address is already in use. Please log in or recover your
password.")

```

```

except Exception as e:
    // Handle any other unexpected errors
    logError(e)
    displayErrorMessage("An unexpected error occurred while creating your account. Please try again later.")

// Main program
function main():
    // Listen for customer's click on the "Create Profile" button
    while true:
        try:
            if customerClickedOnCreateProfileButton():
                // Handle click on the "Create Profile" button
                handleCreateProfileButtonClick()
        except Exception as e:
            // Handle any errors that occur during the main program execution
            logError(e)
            displayErrorMessage("An unexpected error occurred. Please try again later.")

// Listen for account creation form submission
while true:
    try:
        if customerSubmittedAccountCreationForm():
            customerData = getCustomerDataFromForm()
            handleAccountCreationFormDataSubmission(customerData)
    except Exception as e:
        // Handle any errors that occur while listening for form submission
        logError(e)
        displayErrorMessage("An unexpected error occurred. Please try again later.")

```

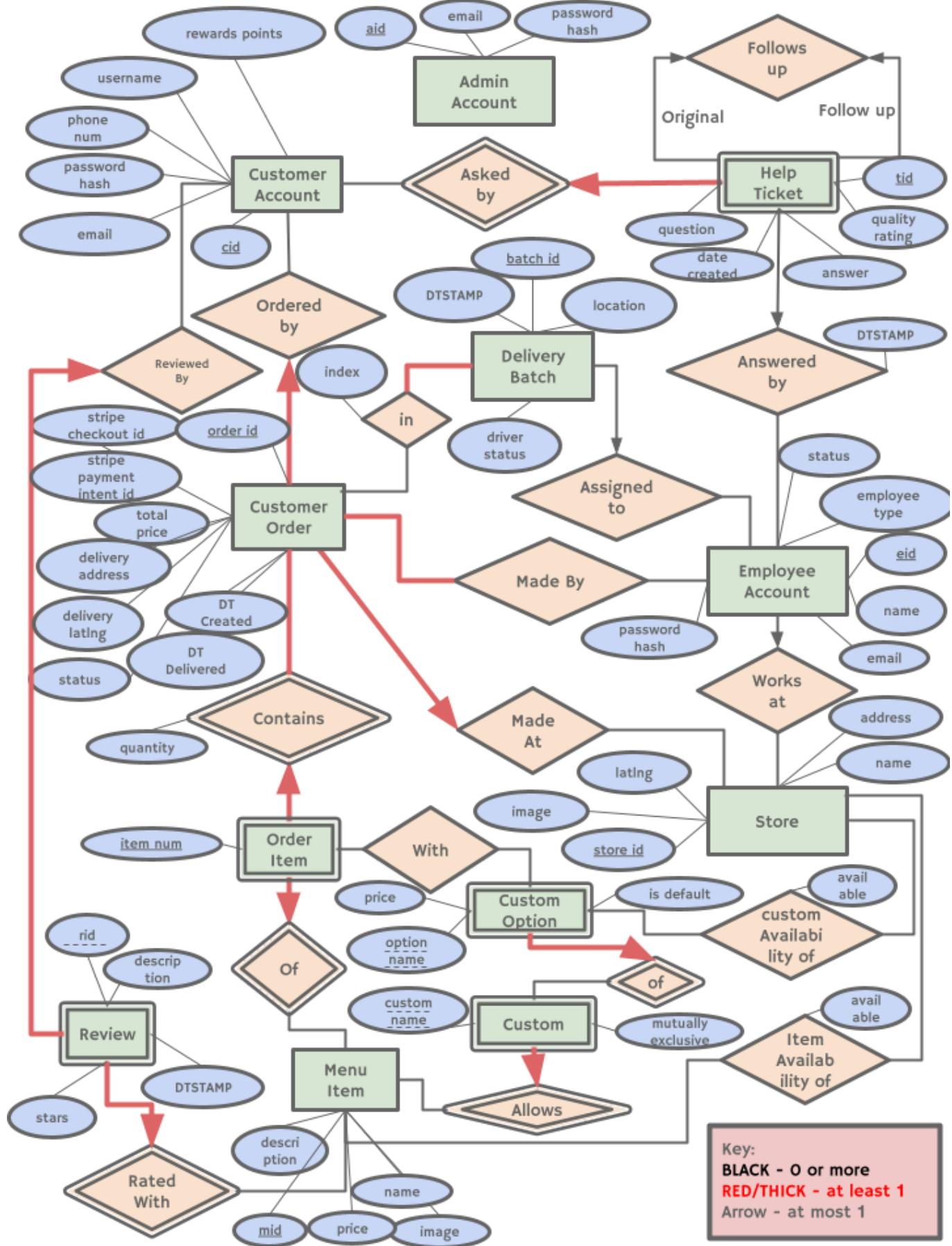
Detailed Central Use Case - Customer Account

Use-Case	Customer Account
Description	Customers will be able to create an account by providing their information such as their name, address, phone number, email, and authentication credentials. The customer account will be stored in a database and associated with a unique customer identifier. Upon account creation, customers will be able to view their account information and will gain access to the rewards program, an ability to

	<p>view previous orders, and participation in giving order feedback as well as Mr. Pizza service feedback after the completion of an order. Customers will also be able to update their account information, which will be reflected by writing the new information to the database in which the customer accounts are stored.</p>
Related Requirements	<p>REQ-Customer-Account1: As a potential customer, I want to create an account in which I can view my order history and accumulate reward points</p> <p>REQ-Customer-Account3: As a returning customer, I want to view my previous orders, so that I can easily repurchase items or track my spending.</p> <p>REQ-Customer-Feedback1: As a customer, I can offer feedback on the ordering process via a popup after completing an order, so that I can help improve the ordering experience for others.</p>
Initiating Actor	Customer
Actor's Goal	To successfully create a Mr. Pizza account to gain access to rewards and feedback participation, and to have the ability to see previous orders.
Participating Actors	Customer, Administrators
Preconditions	Customers must have provided valid account credentials in order for the account to be successfully created. Invalid credentials will lead to an error in the account creation and will prompt the user to re-enter credentials.
Postconditions	Customer account creation will successfully complete if and only if, credentials are valid and customer identification in the database is unique for each specific customer. The account will be up and running and the customer will have access to the rewards program, feedback participation, and have the ability to see previous orders.

Flow of Events for Main Success Scenario	<ol style="list-style-type: none">1. Customer is prompted to create an account with Mr. Pizza.2. Customers are prompted to create a unique username and a password for the account.3. Once the customer provides a username and password for the account, information such as first and last name, phone number, email address, delivery address, and payment method will be entered by the customer.4. Customer information and credentials is validated and then sent into a database where all customer accounts are stored.5. The customer is then notified of a successful account creation and has the ability to view the account and use its features.
--	--

Report 2 Pt 1: ER Diagram and Schema



MySQL Schema

```

create table customer_account(cid int primary key auto_increment, username varchar(30),
    phone_num varchar(15), password_hash varchar(200), email varchar(40), rewards_points
    int DEFAULT 0);

create table admin_account(aid int primary key auto_increment, email varchar(40),
    password_hash varchar(200));

create table store (store_id int primary key auto_increment, name varchar(100), address
    varchar(200), latlng point not null, image_url varchar(1000));

create table employee_account(eid int primary key auto_increment, name varchar(50),
    employee_type varchar(20), email varchar(50), password_hash varchar(200), status
    varchar(50), works_at int,
    foreign key(works_at) references store(store_id) on delete set null);

create table help_ticket(tid int auto_increment primary key, asked_by int not null, answered_by
    int, date_created date, question varchar(200), answer varchar(200), quality_rating
    boolean, original_tid int, DTSTAMP datetime,
    foreign key(asked_by) references customer_account(cid) on delete cascade,
    foreign key(answered_by) references employee_account(eid) on delete cascade);
alter table help_ticket add constraint original_tid_references_help_ticket foreign
    key(original_tid) references help_ticket(tid) on delete set null;

create table customer_order(order_id int primary key auto_increment, status varchar(20),
    total_price float, delivery_address varchar(100), delivery_latlng point, DT_created
    datetime, DT_delivered datetime, ordered_by int not null, made_at int not null,
    stripe_checkout_id varchar(100), stripe_payment_intent_id varchar(100),
    foreign key(ordered_by) references customer_account(cid) on delete cascade,
    foreign key(made_at) references store(store_id) on delete cascade);

create table menu_item(mid int primary key auto_increment, item_name varchar(100), price
    float, image_url varchar(1000), description varchar(1000), category varchar(100));

create table order_item(item_num int auto_increment, order_id int not null, mid int not null,
    primary key(item_num, order_id),
    foreign key(mid) references menu_item(mid) on delete cascade,
    foreign key(order_id) references customer_order(order_id));

create table custom(custom_name varchar(50), mid int, mutually_exclusive boolean not null,
    primary key(mid, custom_name),
    foreign key(mid) references menu_item(mid) on delete cascade);

create table custom_option(custom_name varchar(50), mid int, option_name varchar(50), price
    float not null, isDefault boolean not null,
    primary key(mid, custom_name, option_name),

```

```

foreign key(mid, custom_name) references custom(mid, custom_name) on delete
cascade);

create table with_custom(order_id int, item_num int, mid int, custom_name varchar(50),
option_name varchar(50),
primary key(order_id, item_num, mid, custom_name, option_name),
foreign key(order_id, item_num) references order_item(order_id, item_num) on delete
cascade,
foreign key(mid, custom_name, option_name) references custom_option(mid,
custom_name, option_name) on delete cascade);

create table delivery_batch(batch_id int primary key auto_increment, current_latlng point,
DT_stamp datetime, driver_status varchar(20), assignedToEmp int,
foreign key(assignedToEmp) references employee_account(eid) on delete cascade);

create table in_batch(order_id int, batch_id int, order_index int,
primary key(order_id, batch_id),
foreign key(order_id) references customer_order(order_id) on delete cascade,
foreign key(batch_id) references delivery_batch(batch_id) on delete cascade);

create table review(mid int not null, rid int auto_increment primary key, description
varchar(1000), DT_stamp datetime, stars float,
reviewedBy int not null,
foreign key(mid) references menu_item(mid) on delete cascade,
foreign key(reviewedBy) references customer_account(cid));

create table made_by(order_id int, eid int,
primary key(order_id, eid),
foreign key(order_id) references customer_order(order_id),
foreign key(eid) references employee_account(eid));

create table item_availability(mid int, store_id int, available boolean not null,
primary key(mid, store_id),
foreign key(mid) references menu_item(mid) on delete cascade,
foreign key(store_id) references store(store_id) on delete cascade);

create table custom_availability(mid int, custom_name varchar(50), option_name varchar(50),
store_id int, available boolean not null,
primary key(mid, custom_name, option_name, store_id),
foreign key(mid, custom_name, option_name) references custom_option(mid,
custom_name, option_name) on delete cascade,
foreign key(store_id) references store(store_id) on delete cascade);

```

Database Access Permissions

Legend: x - No access, r - Read, w - Write, + - Management

Table	Visitor	Customer	Employee	Admin
customer_account	x	rw	r	rw+
admin_account	x	x	x	rw+
store	r	r	r	rw+
employee_account	x	x	rw	rw+
help_ticket	x	rw	rw	rw+
customer_order	x	rw	rw	rw+
menu_item	r	r	r	rw+
order_item	x	rw	rw	rw+
custom	r	r	r	rw+
custom_option	r	r	r	rw+
with_custom	x	rw	rw	rw+
delivery_batch	x	r	rw	rw+
in_batch	x	r	rw	rw+
review	x	rw	rw	rw+
made_by	x	r	rw	rw+
item_availability	r	r	rw	rw+
custom_availability	r	r	rw	rw+

Part 1 Contributions	
Groups	Percentage/Part
Map/Driver	Ji Wu (100%)
Pizza Pipeline	Arya Shetty (33%), Vineal Sunkara (33%), Nikash Rajeshbabu (34%)
Menu and Items	Keanu Melo Rojas (33%), Joshua Menezes (34%), Thomas O'Connell (33%)
Customer/Accounts	Ethan Sie (34%), Anna Yeakel(33%), Aman Patel(33%)
ER Diagram + Schema	Brandon Cheng(34%), Damon Lin(33%), Arya Shetty(33%)

Report 2 Pt 2: API Specification

Authentication

All APIs which require authentication will make use of JWT (JSON Web Tokens) to ensure authentication. If authentication is required, it is denoted in the inputs as JWT(actor role). The process to get this token is as follows:

When an account is created:

- The server receives the username/email and password
- A random salt of a fixed length is generated
- The salt is prepended to the password
- A slow hash function, such as Bcrypt, is used to hash the salt+password
- The random salt is prepended to the hash, and then stored in the database.

When a login is requested:

- User visits the sign-in page, and inputs their credentials (username/email, password)
- The data is sent to the server securely using Https
- The server retrieves the salt+hash stored in the database
- The fixed length of the salt is taken from the beginning and prepended to the received password
- The salt+password is hashed with Bcrypt, and the resulting hash is checked on the second half of the database entry
- If this matches, then a JWT with the appropriate credentials is generated and returned to the user
- The JWT is stored as a HttpOnly, secure browser cookies of the user, to defend against XSS and CSRF attacks.

This uses hashing to prevent storing plaintext passwords or credentials, so in the event of a database breach, users and employees will not have their accounts stolen and access will not be granted to protected, sensitive business information. To prevent rainbow attacks, where an attacker hashes a known database of a lot of potential passwords and checks if they match the hashes, we use a random salt which is appended before the hash is generated. Lastly, to prevent brute force attacks, we use a special hash function that is designed to be slow

Map/Driver API Specification

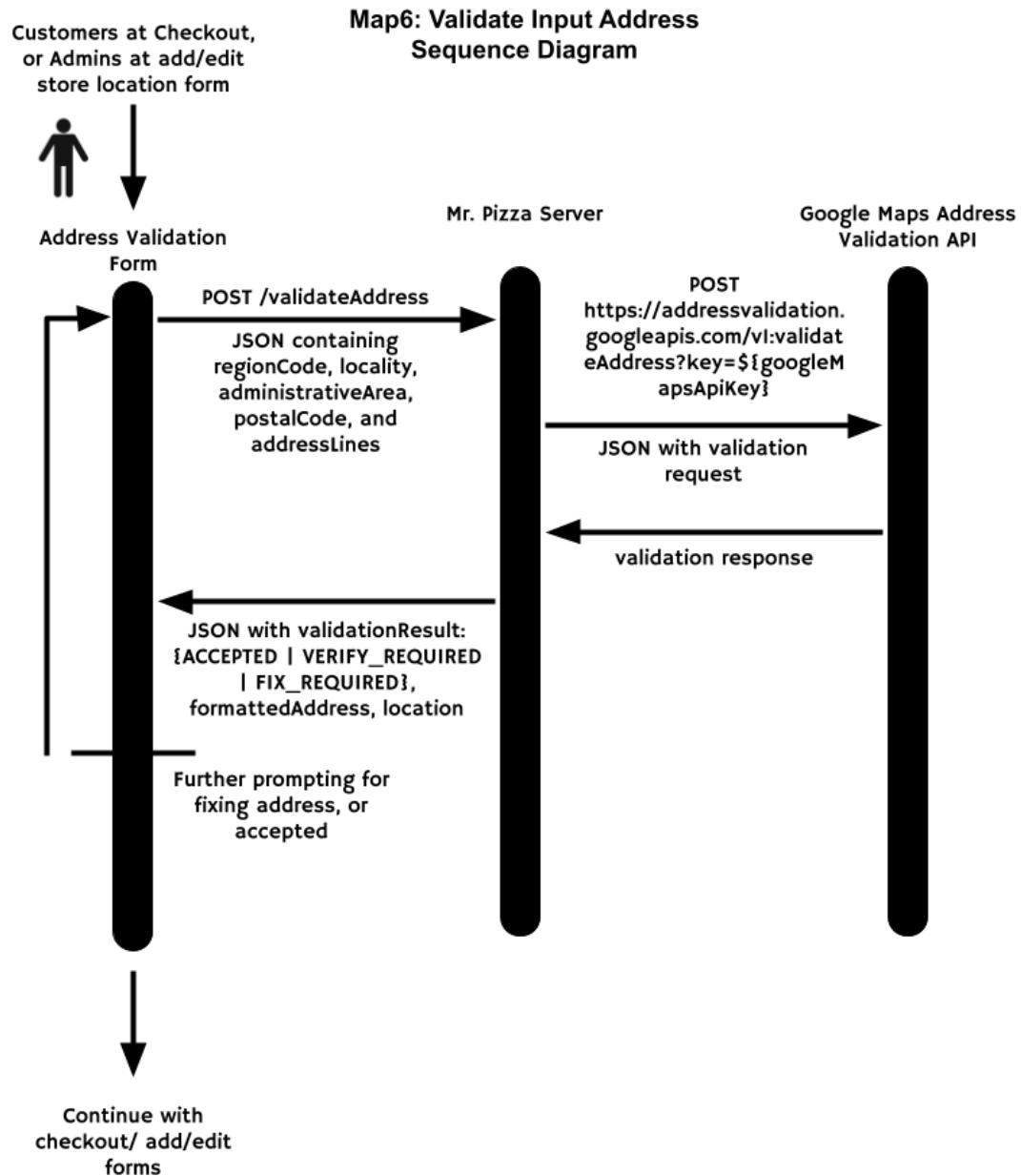
Operation	Resource URI	Method	Inputs and validation	Returns	HTTP Response Codes	Use Case
View store locations	/stores	GET	N/A	JSON list Store Locations	200, 404, 405	Map1-View Store Locations
View detailed store information	/stores/{store_id}	GET	N/A	JSON of Detailed Store Info	200, 404, 405	Map2-View Detailed Store Information
Add Store	/stores/add	PUT	JWT(admin), Store Location [valid location], Store Name [between 0 and 50 characters]	JSON of Success/failURI message, store_id	201, 400, 401, 405, 415	Map3-Add Store Locations
Edit store information	/stores/{store_id}/edit	POST	JWT(admin), JSON of updated information [Must specify location, image with valid value]	JSON Success/failure message	200, 400, 401, 404, 405, 415	Map4: Edit Store Locations
Get List of Store Names	/storeNames	GET	N/A	JSON list of store names and ids	200, 405	Map5: Get List of Store Names
Validate Input Address	/validateAddress	POST	JSON with regionCode, locality, administrativeArea, postalCode, and addressLines regionCode must be "US", administrativeArea must be "NJ"	JSON error or success with validationResult. If FIX_REQUIRE_D, also includes warning_fields. Otherwise, includes formattedAddress and location.	200, 400, 405, 415, 500	Map6: Validate Input Address

View list of unassigned orders	/unassigned/{store_id}	GET	JWT(employee)	JSON list of unassigned orders	200, 401, 405	Delivery1: View Unassigned Orders
See list of available drivers	/availableDrivers/1	GET	JWT(employee)	JSON of available drivers with their email and current status	200, 401, 405	Delivery2: See Available Drivers
Calculate optimal driver assignment	/optimalAssignment}/{store_id)	POST	JWT(employee)	JSON list of coordinates	200, 400, 401, 405, 415	Delivery3: Calculate Optimal Assignment
Assign orders to Drivers	/assignOrders/{store_id}	POST	JWT(employee), Driver eids, Order batches.	JSON of success/fail URL message	200, 400, 401, 405, 415	Delivery4: Assign Orders to Drivers
See driver's assigned order	/directions/assignedOrder	GET	JWT(driver)	JSON of batch id, order information	200, 401, 405	Delivery5: Get Assigned Order Information
Get waypoints for drivers	/directions/waypoints/{eid}	GET	JWT(driver), assigned orders.	JSON of waypoints	200, 400, 401, 405	Delivery6: Get Waypoints for Drivers
Update Driver's Location	/geolocation/updatePos	POST	JWT(driver), must be assigned a delivery batch	confirmation or error JSON	200, 400, 401, 404, 405, 415	Delivery7: Update Driver Location
See Driver's Location	/geolocation/fetchDriverPosCustomer	GET	JWT(Customer), customer must have an In-Transit order.	JSON of driver's position, or error	200, 401, 404, 405	Delivery8: See Driver's Location
See Delivery Destination (Customer)	/geolocation/fetchCustomerPos	GET	JWT(customer), customer must have an In-Transit order.	JSON of delivery_latlng, or error	200, 401, 404, 405	Delivery9: See My Delivery Destination (Customer)

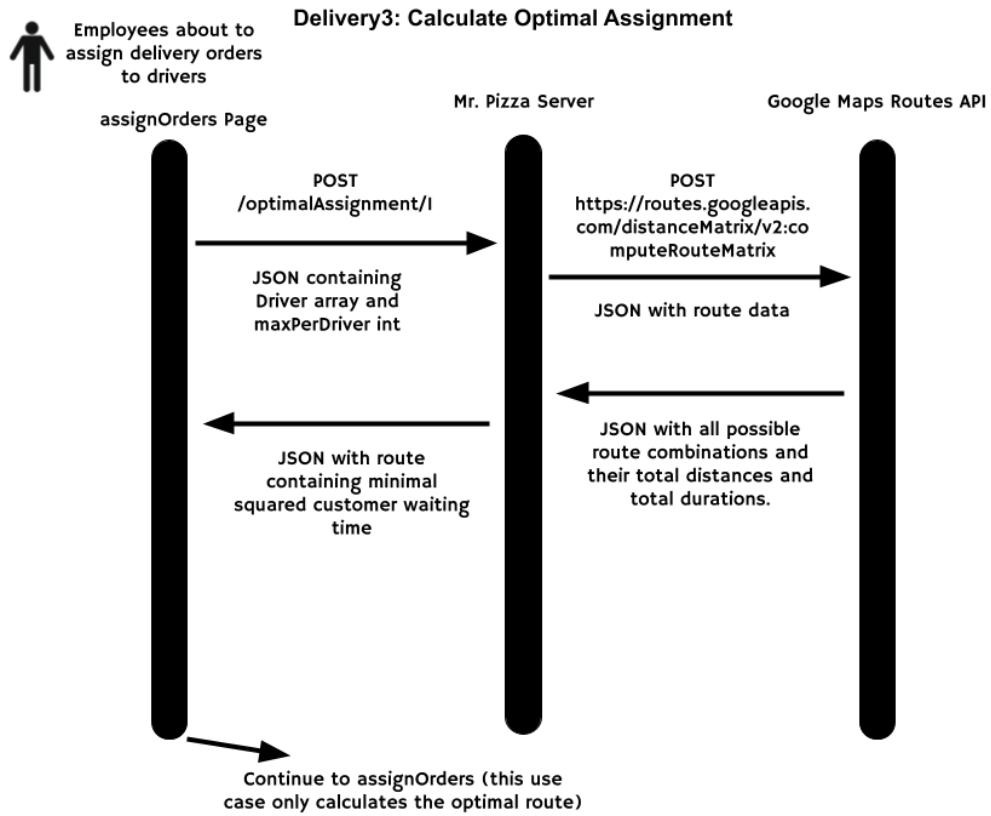
Get Delivery Time Estimate	/timeEst/getOrderTimeEst	GET	JWT(customer), customer must have an In-Transit order.	JSON of time estimate, or error	200, 401, 404, 405	Delivery10: Get Delivery Time Estimate
Ask the Chatbot a Question	/chatbot	POST	JSON containing question.	JSON containing error or answer	200, 400, 405, 415	Chatbot1: Ask the Chatbot a Question

Updated Map/Driver Sequence Diagrams

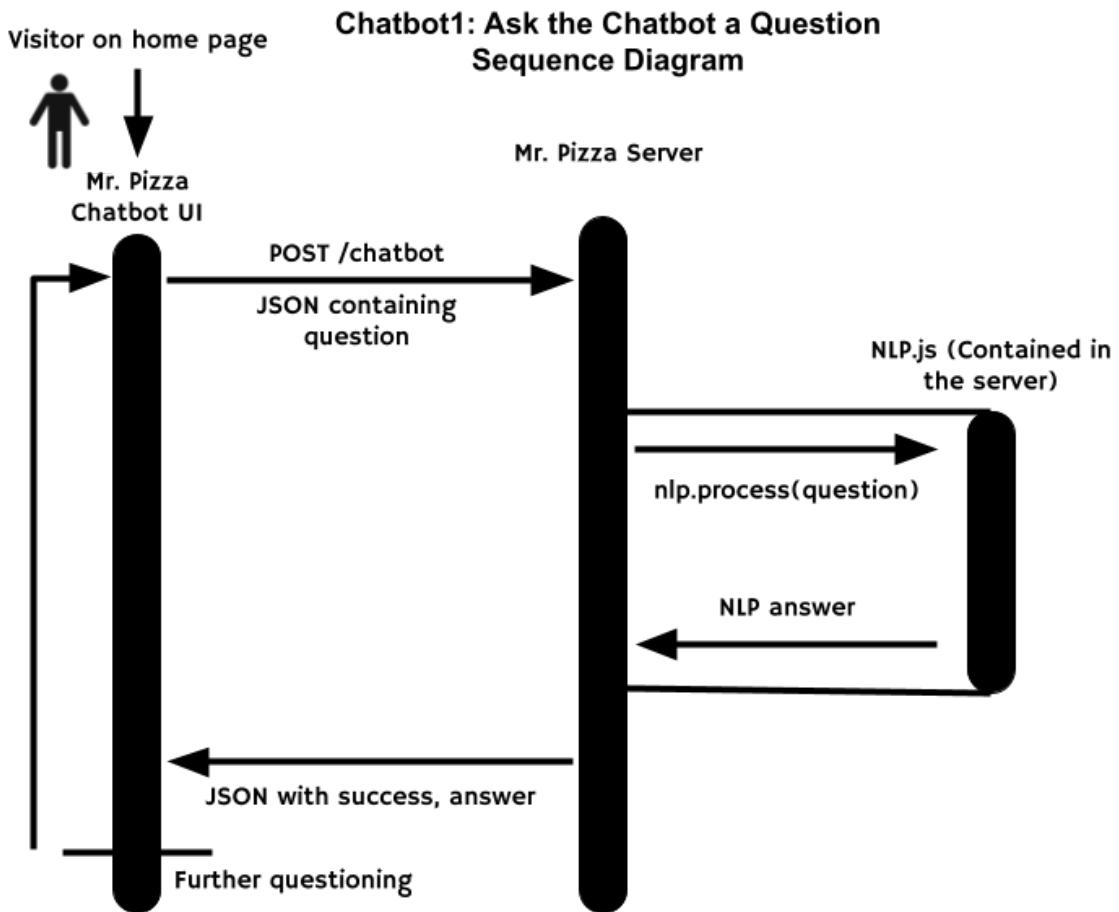
Use Case: Map6: Validate Input Address Sequence Diagram



Use Case: Delivery3: Calculate Optimal Assignment



Use Case: Chatbot1: Ask the Chatbot a Question



Pizza Ordering Pipeline API Specification

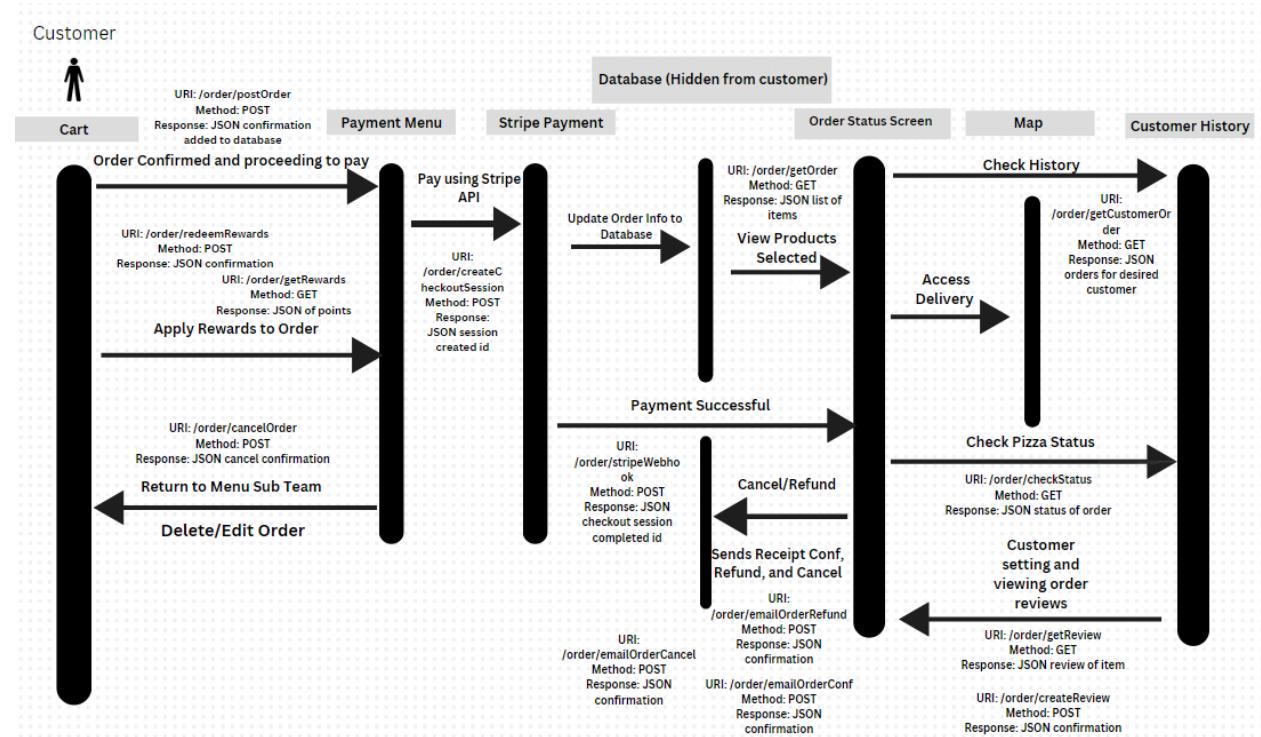
Operation	Resource URI	Method	Inputs and validation	Returns	HTTP Response Codes	Use Case
Gets order for current order id being displayed in order status page	/order/getOrder	GET	Validation, logged in as customer for jwtBody	JSON of results from query (order that is found from order id) or error message	200, 500, 404, 400	View Order Status
Sends the order to the database	/order/postOrder	POST	JSON of cart data split into orderData (for customerOrder table), isDelivery to determine whether there is address or not, orderItemdata (for order_item table)	Return order id of the new order inserted or error message	200, 400, 500, 422	Order Payment
Cancel the order and set the status of the order to cancelled in the database	/order/cancelOrder	POST	Logged in as customer for jwtBody	Success/failure message (canceled or failed to cancel order)	200, 500, 404	Cancel Order
Check status of an order	/order/checkStatus	GET	Logged in as customer for jwtBody	Returns status of order id in body or fail message	200, 500	View Order Status

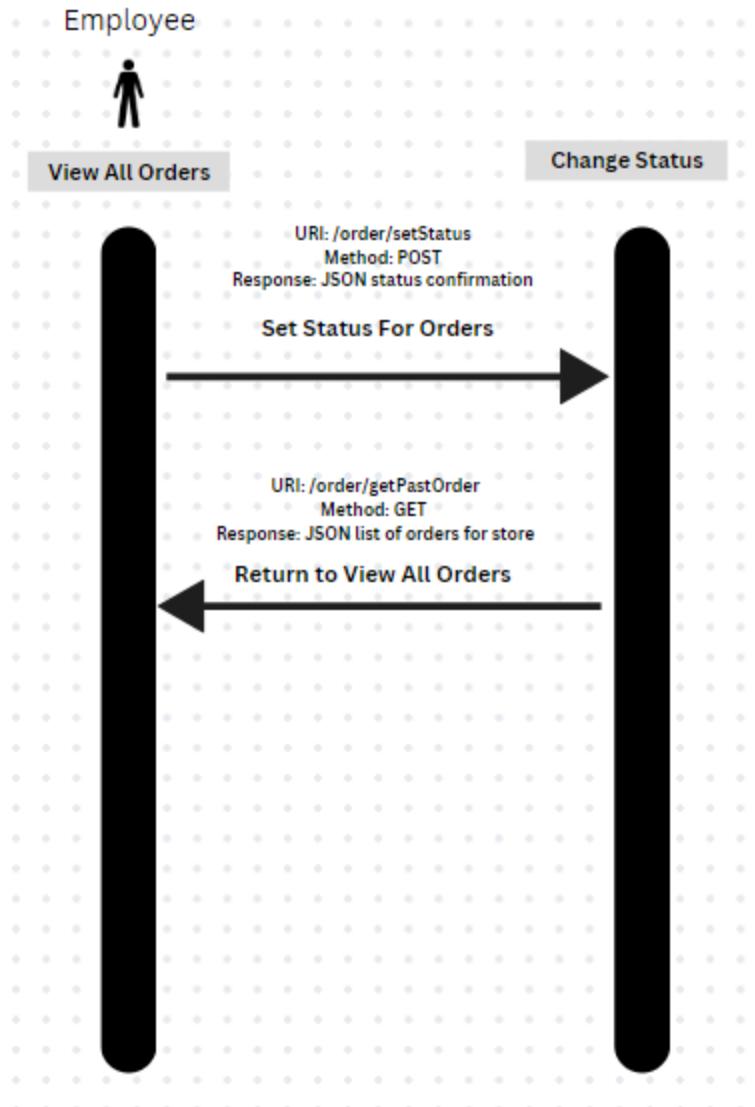
Creates a stripe session with user information to pay for current order	/order/createCheckoutSession	POST	JSON of total, tipEnabled, and tipAmount, to calculate the stripe total	Returns the stripe session created when paying for the order	200, 400, 500	Stripe Payment, Add Gratuity tip
Get the order id of the current customer id logged in	/order/getOID	GET	Logged in as customer for jwtBody	JSON of order ids associated with customer id	200, 500, 404	View all customer orders
Gets all orders sent to a particular store based on employee logged in	/order/getPastOrders	GET	Logged in as employee, for jwtBody	JSON of results from query (all orders that are found in database) or error message	200, 404, 500	View all store orders
Updates the status of a particular order	/order/setStatus	POST	JSON of order id and new status	Success/Failure messages based on order missing or error updating statuses	200, 400, 404, 500	View Order Status
Sends the email cancellation to customer account	/order/emailOrderCancel	POST	Logged in as customer, to get jwtBody	Return a success message or failure if email was sent	200, 404, 500, 400	Cancel Order

Sends the email refund to customer account	/order/email OrderRefund	POST	Logged in as customer, to get jwtBody	Return a success message or failure if email was sent	200, 404, 500, 400	Refund Order
Gets the logged in customer's current reward points	/order/getRewards	GET	Logged in as customer, to get jwtBody	JSON of reward points customer has, or failure message	200, 404, 500	Apply Rewards Points
Uses customer's points to redeem an item in their order for free	/order/redeemRewards	POST	Logged in as customer, to get jwtBody	Success/Failure message if points were used or not	200, 500	Apply Rewards Points, Stripe Payment
Check if customer clicked carryout or delivery	/order/check Option	GET	Logged in as customer, to get jwtBody	Success/Failure message is correctly checked or not, whether delivery is null or not	200, 404, 500	Input Delivery Address
Create review for selected item customer purchased	/order/create Review	POST	JSON of review description and item_num the review is for	Success/Failure message of whether the review was created or not	201, 500	View all customer orders
Get review for selected item customer purchased	/order/getReview	POST	JSON of item_num, for the review parsing	Success/Failure message of whether the review was retrieved or not	201, 500	View all customer orders

Gets all orders of the logged in customer	/order/getCustomerOrder	GET	Logged in as customer, for jwtBody	JSON of results from query (all orders that are found in database) or error message	200, 404, 500	View all customer orders
Verifies the customer checkout order, and returns payment, deducts reward point if needed, and sends email	/order/handleRefund	POST	JSON of order id	Success or failure message of order being properly refunded	200, 404, 500	Refund Order
Verifies that the checkout session has been created and paid for before proceeding and updating the database order to paid, prompts to send email and increment reward point	/order/stripe Webhook	POST	JSON of checkout session id	Success or failure message, of properly processing Stripe payment	200, 400	Stripe Payment

Updated Pizza Pipeline Sequence Diagrams





Menu API Specification

Considerations:

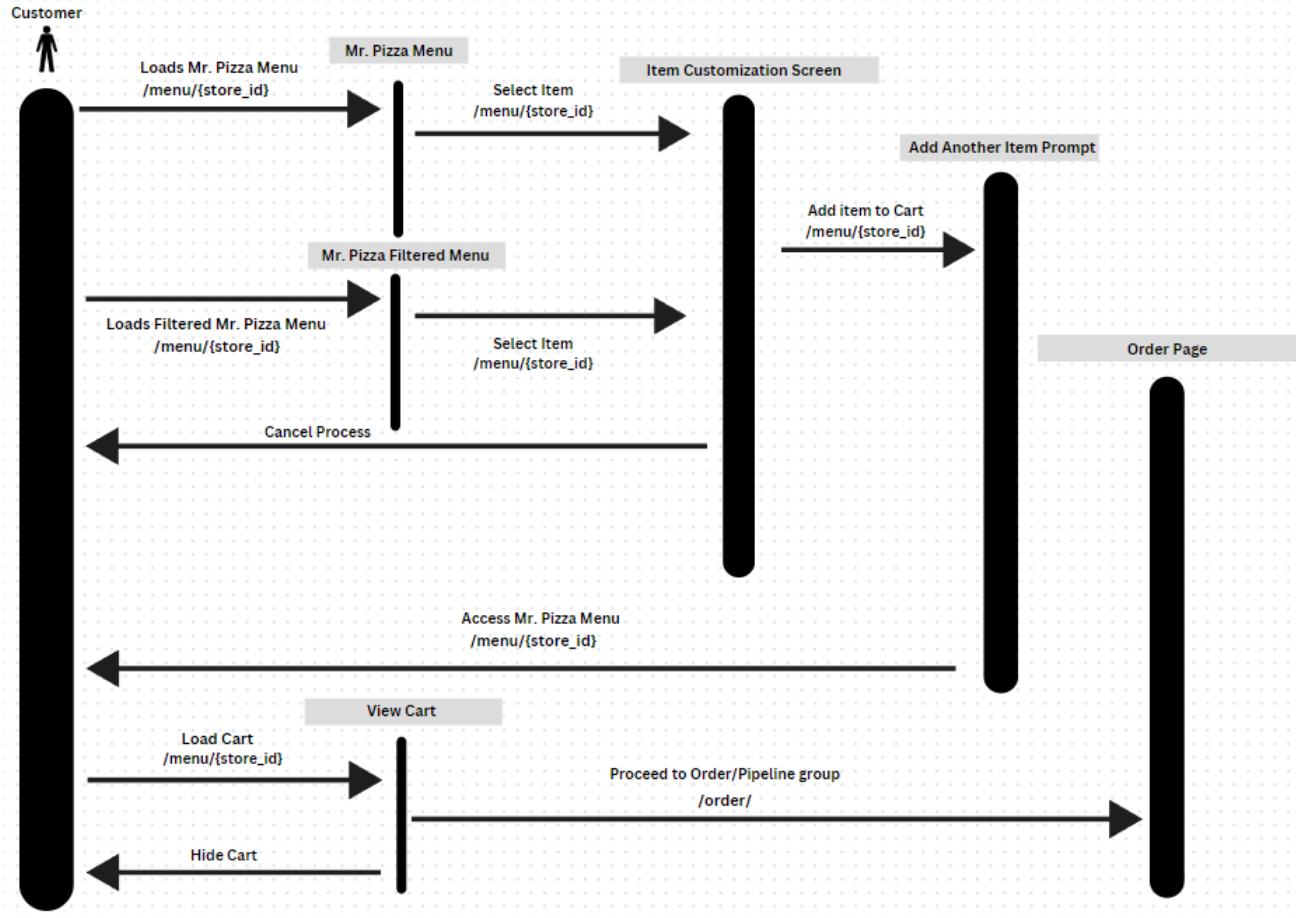
APIs included are operations that need access to server information, which includes loading the menu, viewing menu items and their customization options as a customer, and viewing and updating the menu as an employee or admin of the server.

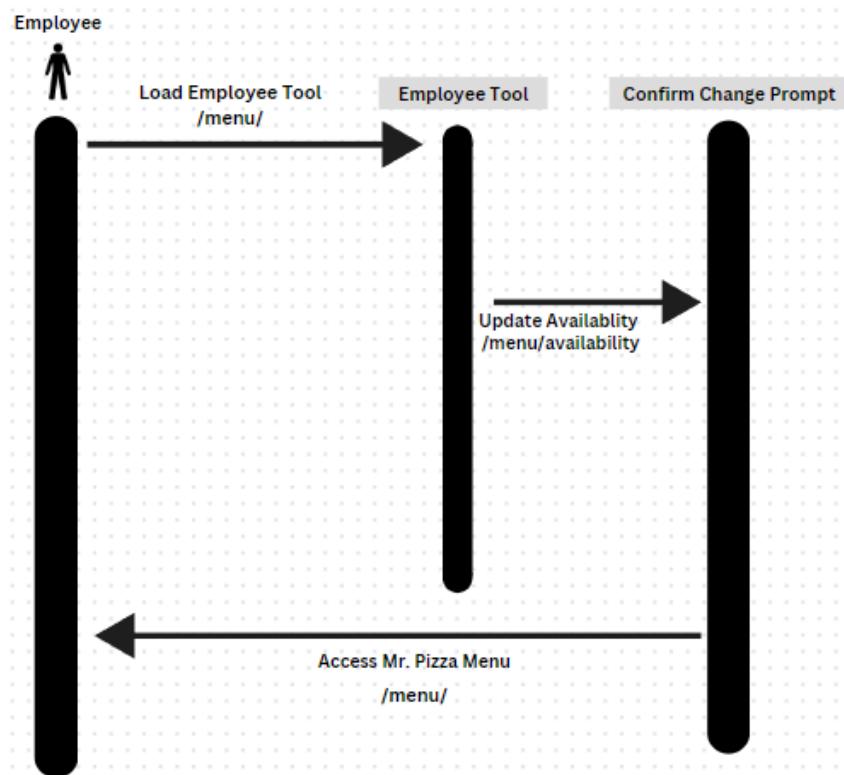
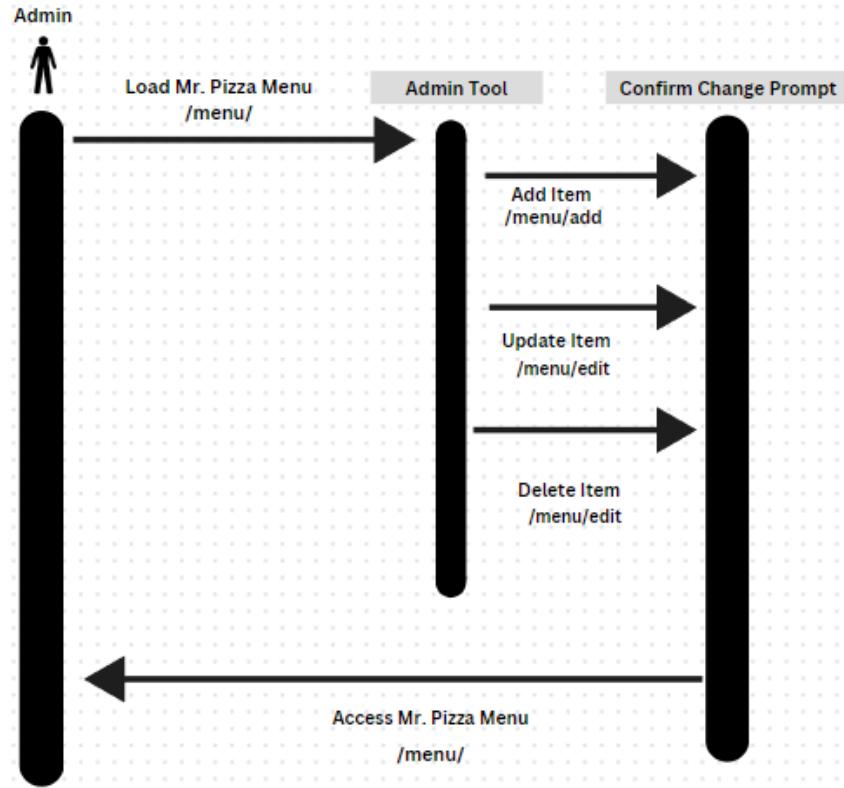
Most menu operations for ordering are handled client side. When a customer adds an item to their order, the information of the item will be written to a JSON file. Operations such adding an item to the cart or customizing an item will update the JSON with the new information for the order. When a customer places their order, the JSON with the order information will be handled by the Pizza Ordering Pipeline to process it. These operations do not need access to the server until the order is placed, which is why they are not listed as API calls.

Operation	Resource URI	Method	Inputs	Returns	HTTP Response Codes	Use Case
(Visitor) Load menu items and their respective options.	/menu/{store_id}	GET	NONE	JSON list of all menu items and their properties	200, 404	Menu-Load
(Visitor) Restart Filter Search	/menu/{store_id}	GET	NONE	JSON list of all original menu items and their properties	200, 404	Menu-Search Menu-Load
(Employee) Update Availability	/menu/availability	POST	Menu_item_id, update_details	JSON confirmation	201, 404	Employee-Availability-1 Employee-Availability-2
(Admin) Add Menu Item	/menu/add	POST	update_details	JSON confirmation	201, 404	Admin-Add-1
(Admin) Edit Menu Item (delete, toppings, price)	/menu/edit	POST	menu_item_id, update_details	JSON confirmation	201, 404	Admin-Edit-1 Admin-Edit-2
(Admin) Add Customization Option	/menu/add	POST	update_details	JSON confirmation	201, 404	Admin-Add-2

(Admin) Edit Customization Option (delete, toppings,price)	/menu/edit	POST	menu_item_id, update_details	JSON confirmation	201, 404	Admin-Edit-1 Admin-Edit-2
--	------------	------	------------------------------	-------------------	----------	------------------------------

Updated Menu System Sequence Diagrams





Customer/Accounts API Specification

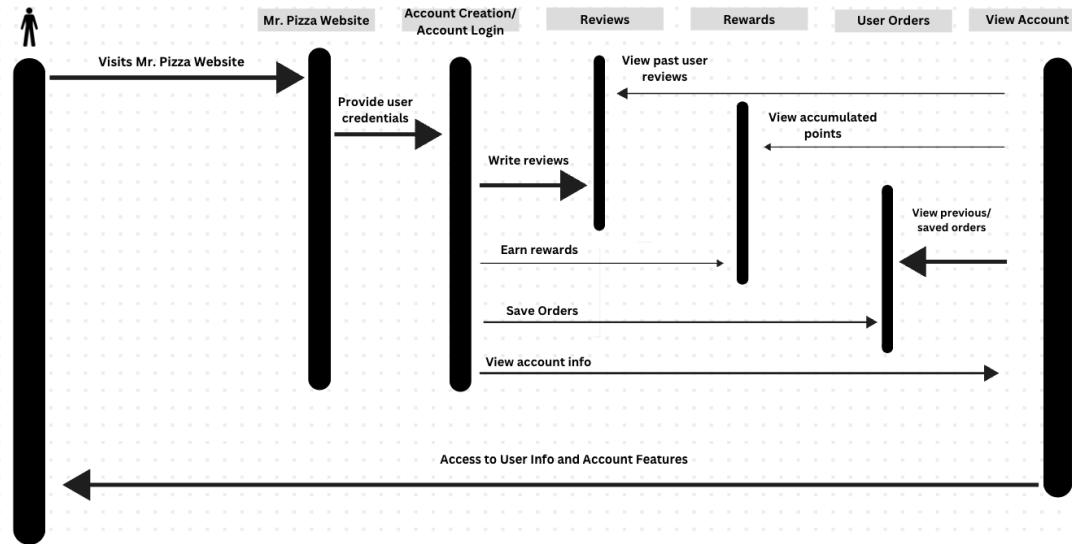
Operation	Resource URI	Method	Inputs	Returns	HTTP Response Codes	Use-Case
Customer Create Account	/customer/register	POST	Username, password	Account ID, JSON account creation confirmation	201, 404	Auth1: Create Customer Account
Customer Login	/customer/login	POST	Username, password	JSON message of successful/failed log in action	200, 404	Auth2: Customer Login
Employee Login	/employee/login	POST	Email, password	JSON message of successful/failed log in action	200, 404	Auth3: Employee Login
Admin Login	/admin/login	POST	Email, password	JSON message of successful/failed log in action	200, 404	Auth4: Admin Login
Get Auth Role	/authRole	GET	JWT	JSON message of auth role	200	Auth5: Get Auth Role
Log out of account	/logout	POST	NONE	JSON message of successful/failed log out of account action	200, 404	Auth6: Logout
Get Employee Store Info	/employeeStoreInfo	GET	Employee Account	JSON Employee's Store info	200, 404	Auth7: Get Employee Store Info
Create support ticket	/support/createTicket	POST	Description	JSON message of ticket creation	201, 400	Support1: Create Support Ticket
Respond to support ticket	/support/respondToTicket	POST	Response	JSON message of response addition	200, 401	Support2: Respond to Support Ticket
View all support tickets	/support/tickets	GET	JWT(Employee)	JSON message of all support tickets and attributes	200, 401	Support3: View All Tickets

View past support tickets	/support/past Tickets	GET	JWT(Customer)	JSON message of all of customer's support tickets	200, 401	Support4: View Past Tickets
View unanswered support tickets	/support/unansweredTickets	GET	JWT(Customer)	JSON message of all currently unanswered support tickets	200, 401	Support5: View Unanswered Tickets
Get Customer Account Info	/getCustomer Info	GET	JWT(Customer)	JSON of Account information	200, 404	Accounts1: Get Customer Info
Edit account info	/account/{accountID}/edit	PATCH	JWT(Customer), Account information to be altered	JSON message of successful/failed updated info, and JSON of edited account information	200, 404	Accounts2: Edit Customer Info
Delete account	/account/delete	DELETE	JWT(Customer)	JSON message of successful/failed deletion of account action	200, 404	Accounts3: Delete Account
Add New Employee	/admin/addNewEmployee	POST	JWT(Admin), First and last name, and store ID of employee to be added	JSON of successful new employee add	201, 404	Accounts4: Add new Employee
Assign Employee to New Store	/admin/assignExistingEmployee	POST	JWT(Admin), Employee ID and new store ID of employee to be assigned to store	JSON of successful new assignment of employee	200, 404	Accounts5: Assign Existing Employee
Get Employee Info	/employee/getInfo	GET	JWT(Employee)	JSON of employee account information	200, 400	Accounts6: Get Employee Info

Edit Employee Account Info as Admin	/admin/editEmployee	POST	JWT(Admin), Employee ID, status, and type for employee to edit.	JSON of successful employee account edit.	200, 404	Accounts7: Edit Employee Info as Admin
Edit Employee Account Info as Employee	/employee/accountInfo	POST	JWT(Employee), New email and password for employee account	JSON of successful employee account edit	200, 404	Accounts8: Edit Employee Info as Employee
Get Total Company Revenue	/getTotalCompanyRevenue	POST	JWT(Admin), Specified date time period.	JSON of total company revenue for that time period.	200, 401	Analytics1: Get Total Revenue By Date
Get Employee Count by Store	/getEmployeeCountByStore	GET	JWT(Admin)	JSON of total amount of employees at stores.	200, 401	Analytics2: View Employee Count By Store
Get Total Amount of Customers	/getTotalCustomers	GET	JWT(Admin)	JSON of total amount of customers.	200, 401	Analytics3: View Total Customers
Get Total Amount of Employees	/getTotalEmployees	GET	JWT(Admin)	JSON of total amount of employees.	200, 401	Analytics4: View Total Employees
Get the Menu Items sorted by times ordered	/getMenuItemsSortedByPopularity	GET	JWT(Admin)	JSON of menu items which are sorted by times ordered from greatest to least.	200, 401	Analytics5: View Menu Items Sorted By Popularity
Get the revenue generated by Menu Items sorted by greatest to least	/getRevenueTotalByMenuItem	GET	JWT(Admin)	JSON of menu items sorted by the highest amount of revenue generated to least.	200, 401	Analytics6: View Revenue Total Per Menu Item

Get times Toppings are ordered sorted by greatest to least.	/getMostPopularToppings	GET	JWT(Admin)	JSON of menu toppings sorted by times ordered from greatest to least.	200, 401	Analytics7: View Toppings Sorted By Popularity
---	-------------------------	-----	------------	--	-------------	---

Updated Customer/Accounts Sequence Diagrams



Third-Party APIs

REST-based APIs

API	API URL	API Functionality	Relevant Use Cases
Google Maps - Address Validation	https://developers.google.com/maps/documentation/address-validation	-Validates any particular human-readable address to place a store location -Shares the driver's location along his/her route to the customer -Provides the geographic locations of each Mr. Pizza store	Map3 Map4 Map6
Google Maps - Routes	https://developers.google.com/maps/documentation/routes	-Handles the time approximation from store location to customer -Generates an (optimal) path for the driver's orders	Delivery1 Delivery3 Delivery10
Stripe	https://stripe.com/	-Handles payment for a customer placing their order at Mr. Pizza. -Handles refunds for eligible customers.	Order-Creation4, Order-PostProcess1

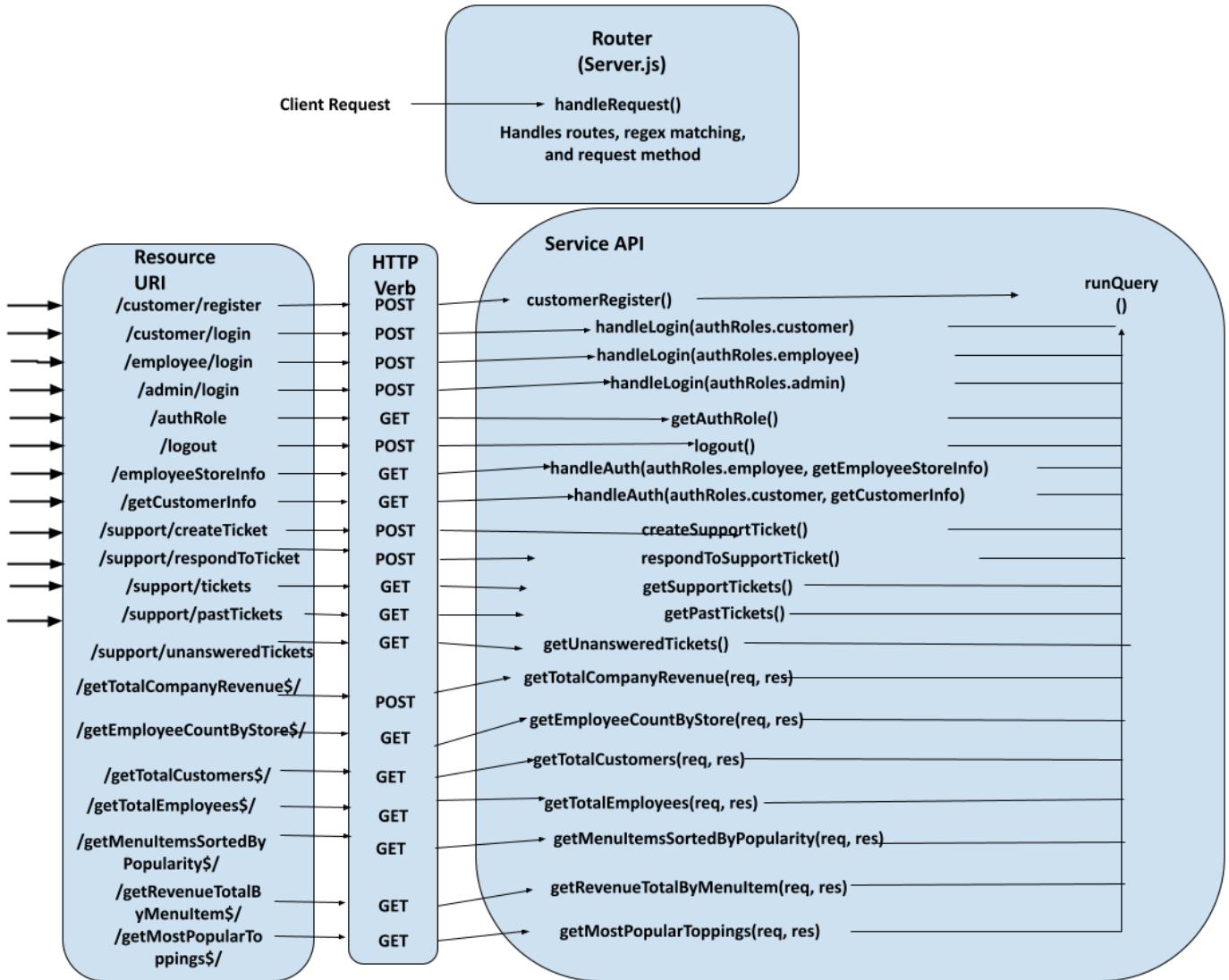
Non REST-based APIs / Libraries

API/Library	API URL	API Functionality	Relevant Use Cases
NLP.js	https://github.com/alpha-group/nlp.js/blob/master/docs/v3/README.md	<ul style="list-style-type: none"> - Handles processing of natural language input. - Learns how to classify inputs based on training from example data of questions and answers. 	Chatbot1
Bcrypt	https://www.npmjs.com/package/bcrypt	<ul style="list-style-type: none"> - Handles securely hashing passwords for storing in the database. - Allows checking an input to see if the hash matches. 	Auth1 Auth2 Auth3 Auth4
Cheerio	https://www.npmjs.com/package/cheerio	<ul style="list-style-type: none"> - Parses HTML contents after webscraping to read parts of the page and find menu information. 	Menu1
Cookie	https://www.npmjs.com/package/cookie	<ul style="list-style-type: none"> - Formats data into the form of a cookie for user authentication. - Secures the cookie to prevent cross site scripting attacks and request forgery attacks. 	Auth1 Auth2 Auth3 Auth4 Auth5 Auth6 Auth7
Jsonwebtoken	https://www.npmjs.com/package/jsonwebtoken	<ul style="list-style-type: none"> - Formats data into the form of a JSON Web Token, which is encrypted and prevents users from editing their credentials. 	Auth1 Auth2 Auth3 Auth4 Auth5 Auth6 Auth7
Nodemailer	https://www.nodemailer.com/	<ul style="list-style-type: none"> - Facilitates sending emails to customers to notify them of their order status changes and confirmations. 	Order-Creation-1
Puppeteer	https://pptr.dev/	<ul style="list-style-type: none"> - Performs web scraping by manipulating Google Chrome for Testing to navigate Domino's website. 	Menu1

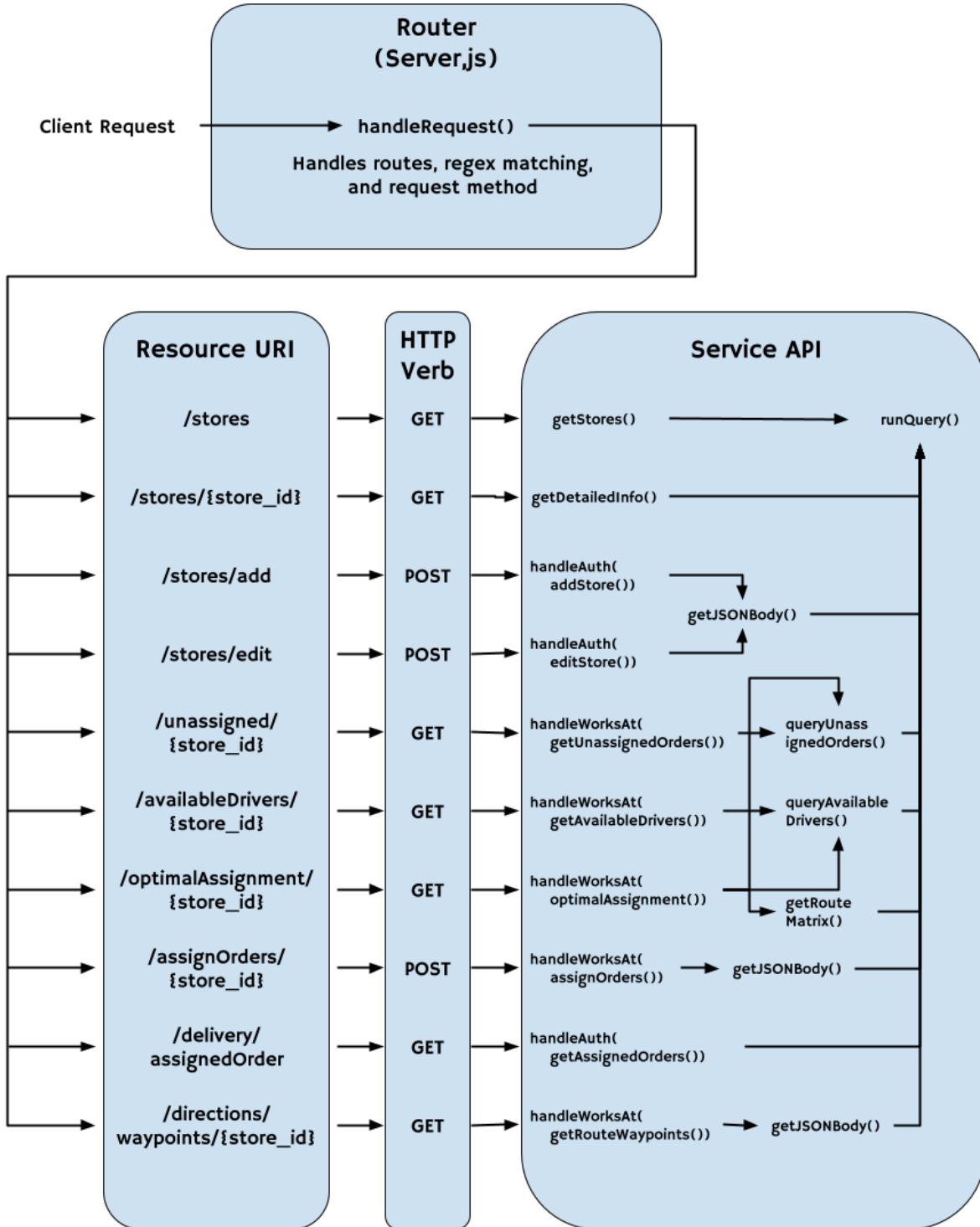
Part 2 Contributions	
Groups	Percentage/Part
Authentication	Brandon Cheng (100%)
Map/Driver API Spec	Brandon Cheng(33%), Damon Lin(34%), Ji Wu (33%)
Pizza Pipeline API Spec	Arya Shetty (33%), Vineal Sunkara (33%), Nikash Rajeshbabu (34%)
Menu and Items API Spec	Keanu Melo Rojas (33%), Joshua Menezes (34%), Thomas O'Connell (33%)
Customer/Accounts API Spec	Ethan Sie (34%), Anna Yeakel(33%), Aman Patel(33%)
Third-Party APIs	Ethan Sie (50%), Damon Lin (50%)

Report 3: Server Side Routing Diagram

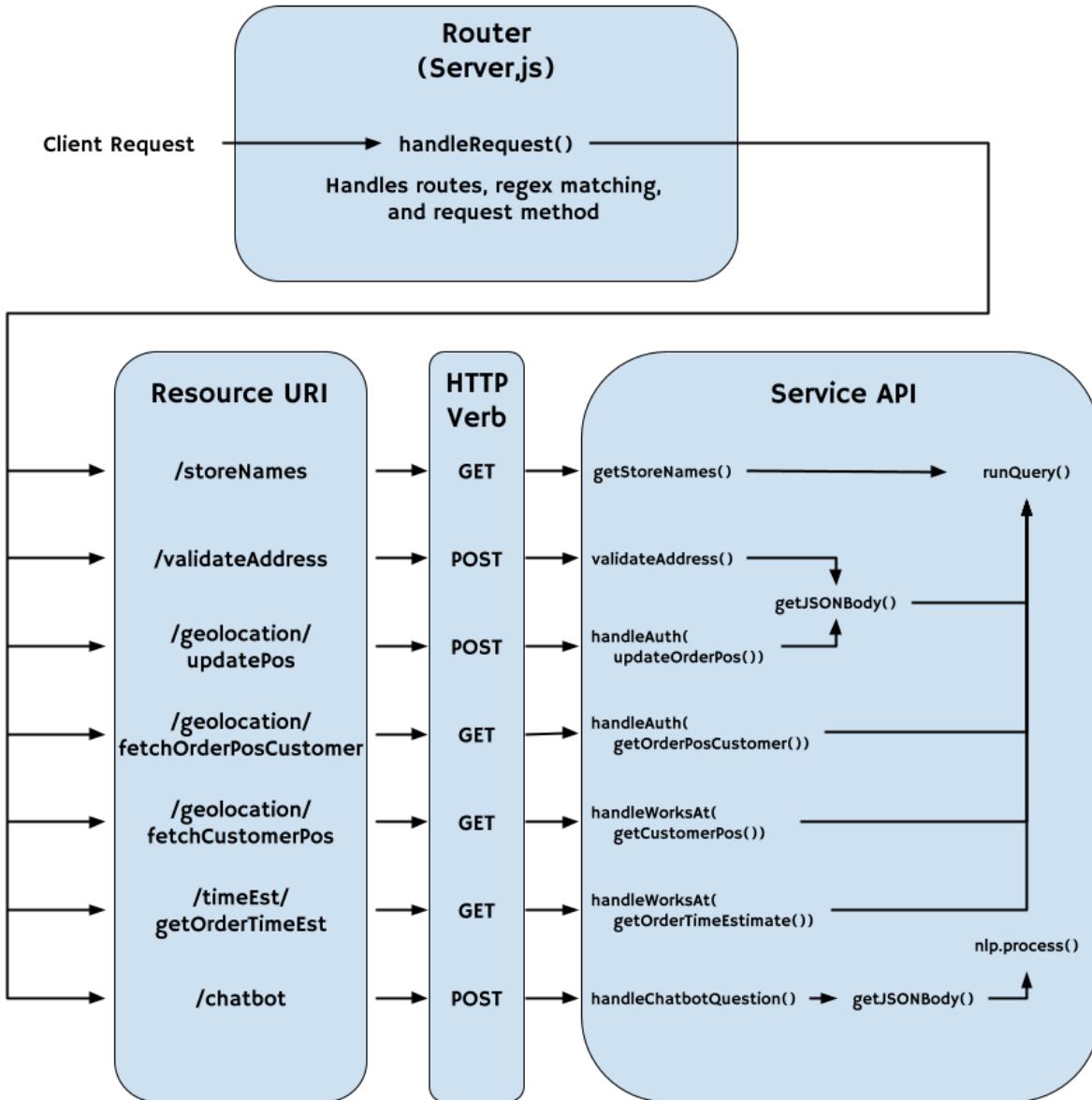
Customer/Accounts Subgroup



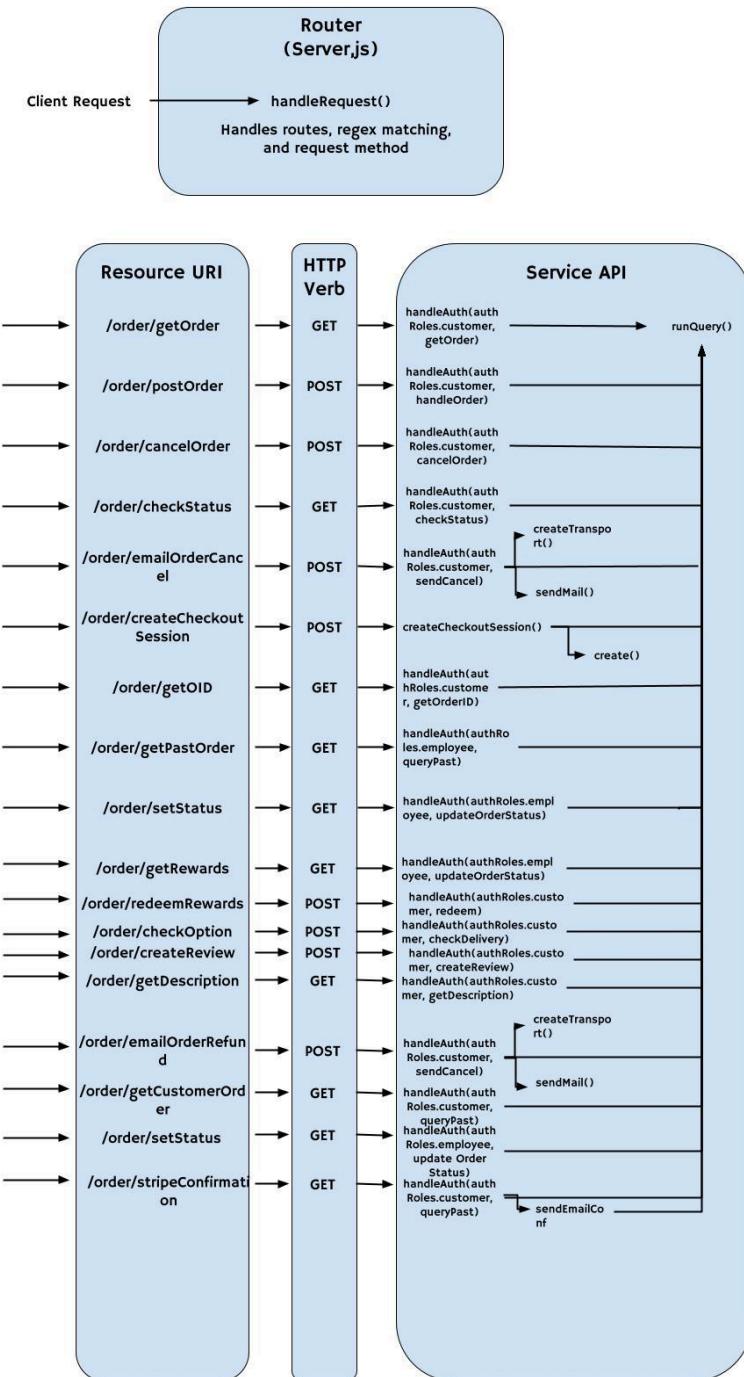
Map/Driver Subgroup



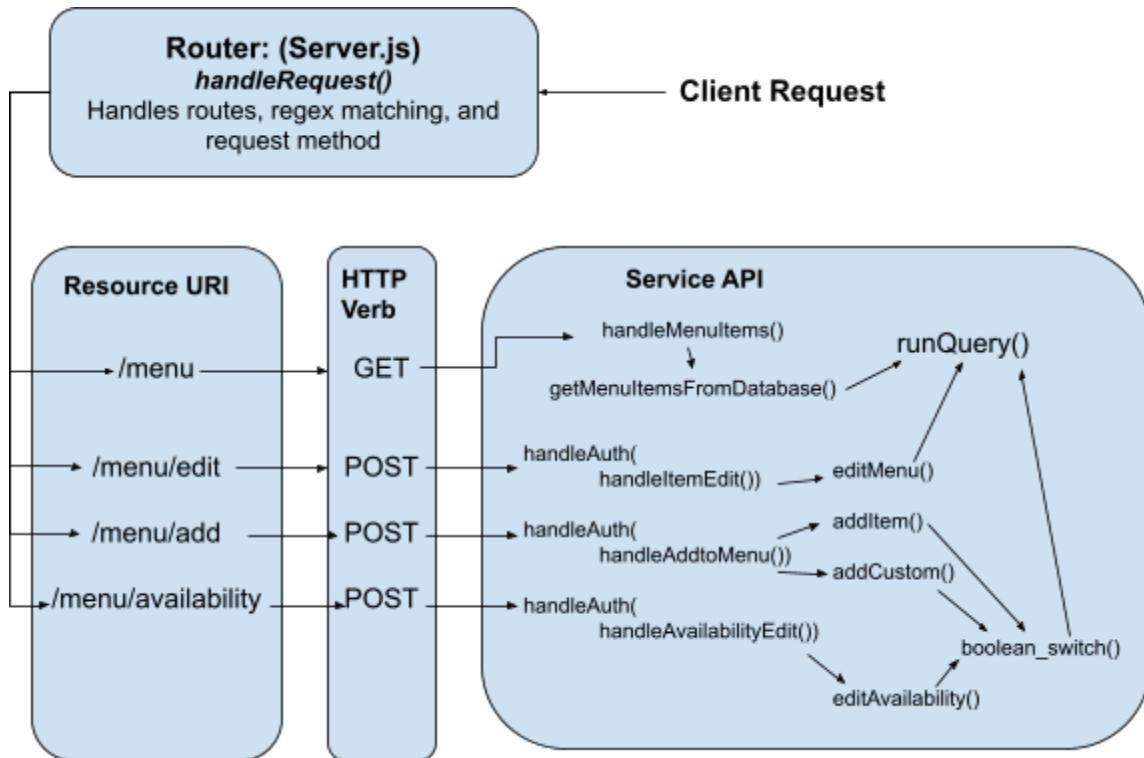
Map/Driver Subgroup Continued



Pizza Pipeline Subgroup



Menu Subgroup



Report 3: Steps to Run Server

Detailed instructions on how to set up the server are provided in the project's README. An abbreviated version will be included below.

Clone Repository

- run `git clone https://github.com/Wingo206/Mr-Pizza.git`

Setup Node Environment

- cd into project directory (`/Mr-Pizza`)
- run `npm ci`

Setup config.js

You will need to make a copy of `/util/config_template.js`, name it `/util/config.js`, and insert proper values. The following config file can be used instead, which contains all sensitive API keys as well.

```
/util/config.js

module.exports = {
  googleMapsApiKey: 'AIzaSyD0Y7j1X8Ug2psarQtGmCnwuRYdz0O-ENk',
  jwtSecretKey: 'thisisasecret',
  database: 'mrpizza',
  unitTestDatabase: 'mrpizzaUnitTest',
  sqlHost: 'localhost',
  sqlAuthUser: 'mrPizzaAuth',
  sqlAuthPw: 'mrPizzaAuthPassword00!',
  sqlVisitorUser: 'mrPizzaVisitor',
  sqlVisitorPw: 'coolPassword001!uhkjn',
  sqlCustomerUser: 'mrPizzaCustomer',
  sqlCustomerPw: 'iujkb7868YTHGJBr65!!@22',
  sqlEmployeeUser: 'mrPizzaEmployee',
  sqlEmployeePw: 'nkjabsdf876909&*^hkjNIU',
  sqlAdminUser: 'mrPizzaAdmin',
  sqlAdminPw: 'TUYGH^6899i09iojlknuyftyjuh',
}
```

Setup MySQL Database

- cd into `/models/`
- run `node substitute.js`
- open `/models/temp/setup.sql` in MySQL workbench and run it (or run `source ./setup.sql` in MySQL CLI).
- cd into `/lib/`
- run `node initData.js`

Setup SSL Certificate

You will need an SSL certificate to run the server with HTTPS. Copy the two files listed below and place them in their respective places, or run the following command to generate your own: `openssl req -x509 -newkey rsa:2048 -nodes -sha256 -subj '/CN=localhost' -keyout server.key -out server.crt` (Note: may not work on windows).

```
/ssl/server.crt
```

```
-----BEGIN CERTIFICATE-----
```

```
MIIIDCTCCAfGgAwIBAgIUFKhqvMnceeIoS5NMaViHBwus08wDQYJKoZIhvvcNAQEL  
BQAwFDESMBAGA1UEAwJbG9jYWxob3N0MB4XDTI0MDMwNzE3NTIwNVoxDTI0MDQw  
NjE3NTIwNVowFDESMBAGA1UEAwJbG9jYWxob3N0MIIBIjANBgkqhkiG9w0BAQE  
AAOCAQ8AMIIBCgKCAQEA2Mf59H7S4tBj5UDfxBCkQKw9s07Q4yvDfnjrlntg5LGw  
efQ7KMUshXuWq92GLEYMdVxLuE1+xNv9MQqCitRJf12A+N0S6PSIlwHgGvrpbVu  
g/kmAis7ifclNq+jsVXBVLYE7/EulHxxXkKSb450W1GtFKjoRrAZV2l1eroangX  
uu7nu9ajwkgb6XZ1B0YUdRDp2GpWfpwnzgPzL6xn9ubpV/QASt67LBhZyvaXGj/3  
ayiZl7un8lljNuQSjNtPGJFBzt3ZH8pxpjAD0gF5LP0zp9gfacAjclYyPnzu6L4  
iZtBh3th+6dzgj5DOABRtyzFuaZnVEHsifm05/u0QwIDAQABo1MwUTAdBgNVHQ4E  
FgQUSeAJZtcmk1T5g6/a4Qewm+rRevEwHwYDVR0jBBgwFoAUSpAJZtcmk1T5g6/a  
4Qewm+rRevEwDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAguUZ  
HL8eAHX5r3apkM4BpCgpsfpagy7evZFdpVAd3z0WLUrMTvegB8Ea4tclKHQa86zT  
ApKIgPUN3mr5WjAtaQgo8250w27kSyLVfVLE0qr6eC5UUXC0gddbwm/a10ed9Pmk  
dpx0+uANDMuFDho+g6oJnlOTIar8nwLBjX0bwdxkh7LV93Qd3yYmf9K6LMG90CWY  
lOEzwTYeBop8U/Th/23B6w89e7PUrxrGfwGCEJs5be8RbLP4hrh31zuYYmJEqpkb  
IqSNXKhvg1A9lbRa90oj8Me+0TB8gzNJCpmgsB74UHHzpYPmCmvKh//5LeykQED/  
DaXo/eLZmODSVhFasg==
```

```
-----END CERTIFICATE-----
```

```
/ssl/server.key
```

```
-----BEGIN PRIVATE KEY-----
```

```
MIIEvjIBADANBgkqhkiG9w0BAQEFAASCBKgwgSkAgEAAoIBAQDYx/n0ftLi0GPL
QN/EEKRArD2zTtDjK8N+e0uWe2DksbB59DsoxSyFe5ar3YaURgx1xEu4TX7E2/0x
CoKK1El/XYD43RLo9IiXAeAa+ul9tW6D+SYCKzuJ9yU2r60xVcFWgTv8S6UfHFe
QpJhvjk5bUa0Uq0hGsBlXaXV6uhqeBe67ue71qPCSBvpdnUHRhR1EOnYalZ8/Cf0
A/MvrGf25ulX9ABK3rssGFnK9pcaP/drKJmXu6fyWWM25BK208YkUH03dkfynGm
OMAM6AXks870n2B9pwCNyVjI+f07oviJm0GHe2H7p30CPkM4AFG3LMW5pmdUQeyJ
+bTn+6hDAgMBAACggEApxPOWgSSZyIz0eGf4dGmLA4VpN+0cSCbt1RAuZ4Z+a2
Y4M4v6lHiG1AXZo8UrFJ+PnPJuuPYoei0R3B2kXGt6RcCxIIpBpPGEYfi6gD8b2z
UfcRQHyVZSHhNGAoPonMnPnxLwxLV94ATXC7m2Gs60zFmvYJeix+AgHQaA9anG5y
0ATfmZReq02z6/BZ9lQhhN2orNvo02saLIqKGzt7lT7+1Z2ja/MhrrcnYVaGup/4
x6j4lA07a/d5z8CZHRhpzCEg8JLbKF83DnjW01trNPkXAxPCjh/R04YSqfKe/EYk
MPe/SwNLZWACJC7lTiKt9ngMfQTZew4t73jxVrhAQKBgQD6I0Zfr5Qipuu+r6Zd
Qn/7/ZUIKpwKMC75+RT67SSJrZTSB0fdJ01SG+RHUTzaWzHY02qlBzmGo7Q6F0fQ
U3LdYHrqPfcQSgT82ukQZZp4YvFGDCIEJ12q6CuQikLQB0jdkErNlNp+ZIckLM1G
QZ5XYUXcAMlcSrIaTScIzAGgEwKBgQDd3JNCr4kNbdXrP981g0a2h+rQwnz58ghp
b4UUNGWzGq3MWULNykOH3LqzNDzVkBvLeeFl8YULqJBgKic73x3saDMZkAaj0W
6zlrzKsI2WXPRQEty6Ncf0qoGR7EIr4PfCFxVvPz0dcaz16Xx8QtPiTUtfTV2gL
8LGNfsa9EQKBgD52Ao0wR1Bj2ChUmjlderIE/2r7oN1WbNSU4y1Jfd847hmHihKK
Vt9zAYzhda4YaZkYKeAtrqq1RDLhT1hxDhTDm83UTVZu9VHipCIpD0qU4mS4c0IS
pmwf74j3Txm+5+4TsuoLCm5XUfuxiEzL4+sz6grm02FNca3pqB6l0p6rAoGBALF7
36KdYnqHv3Yu5pWaxPaIz0SsceZSJUCTolaDhEg/s4GidrrCcDuj7QenZnNDuCqS
HzIsm26MRm2PjVQAW5e+q/gQGWInk5XMH2eJl9nAcJ3bihGRgAJjIopPbjlhwcIJ
f4d4FDgjsV0VP4sUmG/JIxW3amIZwdvJQYD8RmjBAoGBAJKeRm0So02Esp2q/hx0
MJThbOHALQfDNK0FRAZAdY2iP39E7qrqCzIGygsj78grdt74Xs534uiH2/Lsqicg
i7VuEC6zc8zflec79v/CSVf6S9vD/7bb0QAcUdlA0US19MHDH++xLwy/vhPY5Juyk
tIXAXSOX0W4v9RTwB8ignngR
-----END PRIVATE KEY-----
```

Running the Server

- Cd to /Mr-Pizza
- Run node server.js
- View the home page at <https://127.0.0.1:8080/home/home.html>

Report 3: Backend Implementation Details

Routing

All API URIs are routed using a dynamic router which recursively reads the backend files in the lib/ directory. The router checks the exports of each javascript file for an array of routes, which are specified via a JSON object containing the method, path, and handler. When a file exports routes, they are validated and then added to a dictionary of registered routes. Then, when the server receives a request, the dictionary of routes is checked. If there is a match, then the handler is run. Each handler takes in req and res, which allows each API's implementation to be completed without worrying about how it will be called. Once it is registered, the dynamic router will handle it.

Some URIs require an ID of a resource to be present in the path. To handle this, the router allows the path for a route to be a regex. If the path of a request doesn't match any paths in the dictionary, it matches it with any of the regex paths. The first route with a matching regex will have its handler run.

The router handles basic errors such as 404 Resource not Found, as well as 405 Method not Allowed. Because of this, we will not show tests in our API documentation covering these.

Config File

To facilitate better security, all potentially sensitive variables, such as usernames and passwords for SQL users, or API keys are kept in a config file that is not uploaded to git. There is a template file that is uploaded instead, with sensitive fields blank.

SQL Template Substitution

To set up the database with the supplied usernames and passwords from the config file, we have our schema (models/template_setup.sql) with template values. When a contributor sets up the database on their local machine, they first fill out the config file and then run the substitute.js script. This takes the values from their config and substitutes them into the template file wherever designated. Then, this SQL script can be run through the contributor's own root connection to the database, which is never supplied to the server, protecting root access to the database in the case of server breaches. In addition, this generated SQL script is never uploaded to git.

In addition to creating the database, the substitute script also creates a copy of the main database for use in unit testing, which is safe to reset and set up in the state required during unit testing. The substitute script also generates SQL GRANT commands to give the appropriate permissions for each user on each table. Whenever the database is used by the server, the developer of that use case designates which SQL user to use via the provided connection pools in /lib/util/databaseUtil.js. This prevents circumvention of intended access in the case that something goes wrong. The permissions are visible in the substitute script, or in the generated script in /models/temp/setup.sql.

Input Validation Utilities

All APIs that require a JSON body use a helper function called `getJSONBody`, which is available from the `lib/inputValidationUtil.js` file. This function checks for JSON content-type header, proper JSON body, and required properties of the body. If any of these fail, the appropriate error codes will be sent in the response. Because of this, most of our test cases will not display tests for wrong content-type or a malformed JSON body, but we will show the missing required property errors.

Report 3: API Documentation

Customer Accounts

Auth1: Create Customer Account

SQL Queries

```
let queryRes = await runQuery(authPool, `select username from
customer_account c where c.username =
"${decodedData.username}"`);
```

Auth1a: Successful registration

Resource URI: POST /customer/register

Request Header: None

Request Body:

```
{
  "username": "test2",
  "password": "1234"
}
```

Response Code: 201

Response Body:

```
Successfully registered.
```

Developed and tested by Brandon Cheng

Auth1b: Username in use

Resource URI: POST /customer/register

Request Header: None

Database Status: User with username “test2” is already registered

Request Body:

```
{  
    "username": "test2",  
    "password": "1234"  
}
```

Response Code: 409

Response Body:

```
User with username test2 is already registered.
```

Developed and tested by Brandon Cheng

Auth1c: Missing required properties

Resource URI: POST /customer/register

Request Header: None

Request Body:

```
{  
}  
}
```

Response Code: 400

Response Body:

```
{  
    "error": "Missing required properties",  
    "missingProperties": [  
        "username",  
        "password"  
    ]  
}
```

Developed and tested by Brandon Cheng

Auth2: Customer Login

SQL Queries

```
query = `select cid as id from customer_account c where c.username =
"${decodedData.username}"`
```

Auth2a: Successful login

Resource URI: POST /customer/login

Request Header: None

Database Status: User with username “test2” is registered

Request Body:

```
{
  "username": "test2",
  "password": "1234"
}
```

Response Code: 200

Response Headers:

```
set-cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjEyLCJpYXQiOjE3MTIwMzY4NjF9.k_BKYXgV2EST7yYy3wBlaXFPkkr3c5UM9X12
8BbLhkU; Max-Age=86400; Path=/; HttpOnly; Secure
```

Response Body:

```
Successful Login, setting cookie
```

Developed and tested by Brandon Cheng

Auth2b: Missing username property
Resource URI: POST /customer/login

Request Header: None

Database Status: User with username “test2” is registered

Request Body:

```
{  
}  
}
```

Response Code: 400

Response Body:

```
Email or Username properties required.
```

Developed and tested by Brandon Cheng

Auth2c: Missing password property
Resource URI: POST /customer/login

Request Header: None

Database Status: User with username “test2” is registered

Request Body:

```
{  
    "username": "test2"  
}
```

Response Code: 400

Response Body:

```
Password property required.
```

Developed and tested by Brandon Cheng

Auth2d: Incorrect username and/or password
Resource URI: POST /customer/login

Request Header: None

Database Status: User with username “test2” is registered

Request Body:

```
{  
    "username": "test2",  
    "password": "12345"  
}
```

Response Code: 401

Response Body:

```
Incorrect username or password.
```

Developed and tested by Brandon Cheng

Auth2e: User does not exist
Resource URI: POST /customer/login

Request Header: None

Database Status: User with username “test2” is registered

Request Body:

```
{  
    "username": "test3",  
    "password": "1234"  
}
```

Response Code: 401

Response Body:

```
Incorrect username or password.
```

Developed and tested by Brandon Cheng

Auth3: Employee Login

SQL Queries

```
query = `select eid as id from employee_account e where e.email =
"${decodedData.email}" and e.password_hash = "${decodedData.password}"`
```

Auth3a: Successful login

Resource URI: POST /employee/login

Request Header: None

Database Status: Employee with email "employee1@mrpizza.com" is registered

Request Body:

```
{
  "email": "employee1@mrpizza.com",
  "password": "employee1"
}
```

Response Code: 200

Response Headers:

```
set-cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRoUm9sZSI6ImVtcGxveWV
lIiwiWQiOjEsImlhdCI6MTcxMjAzNzMzMX0.xEMTiIMOTq7NE8Vh-DoCbXnWDWxWJCTYwud4L
W6G42M; Max-Age=86400; Path=/; HttpOnly; Secure
```

Response Body:

```
Successful Login, setting cookie
```

Developed and tested by Brandon Cheng

Auth3b: Missing email property
Resource URI: POST /employee/login

Request Header: None

Database Status: Employee with email “employee1@mrpizza.com” is registered

Request Body:

```
{  
}  
}
```

Response Code: 400

Response Body:

```
Email or Username properties required.
```

Developed and tested by Brandon Cheng

Auth3c: Missing password property
Resource URI: POST /employee/login

Request Header: None

Database Status: Employee with email “employee1@mrpizza.com” is registered

Request Body:

```
{  
    "email": "employee1@mrpizza.com"  
}
```

Response Code: 400

Response Body:

```
Password property required.
```

Developed and tested by Brandon Cheng

Auth3d: Incorrect email and/or password
Resource URI: POST /employee/login

Request Header: None

Database Status: Employee with email “employee1@mrpizza.com” is registered

Request Body:

```
{  
    "email": "employee1@mrpizza.com",  
    "password": "employee12345"  
}
```

Response Code: 401

Response Body:

```
Incorrect username or password.
```

Developed and tested by Brandon Cheng

Auth3e: Employee does not exist
Resource URI: POST /employee/login

Request Header: None

Database Status: Employee with email “employee1@mrpizza.com” is registered

Request Body:

```
{  
    "email": "employee2@mrpizza.com",  
    "password": "employee12345"  
}
```

Response Code: 401

Response Body:

```
Incorrect username or password.
```

Developed and tested by Brandon Cheng

Auth4: Admin Login

SQL Queries

```
query = `select aid as id from admin_account a where a.email =
"${decodedData.email}" and a.password_hash = "${decodedData.password}"`
```

Auth4a: Successful login

Resource URI: POST /admin/login

Request Header: None

Database Status: Admin with email "admin@mrpizza.com" is registered

Request Body:

```
{
  "email": "admin@mrpizza.com",
  "password": "admin"
}
```

Response Code: 200

Response Headers:

```
set-cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImFkbWluIiwiaWQiOjEsImlhdCI6MTcxMjAzNzY3MX0.iDIk2shidv8XT15Fn2t89A5CVxct_5nzcCw0LBOMQV0; Max-Age=86400; Path=/; HttpOnly; Secure
```

Response Body:

```
Successful Login, setting cookie
```

Developed and tested by Brandon Cheng

Auth4b: Missing email property
Resource URI: POST /admin/login

Request Header: None

Database Status: Admin with email “admin@mrpizza.com” is registered

Request Body:

```
{  
}  
}
```

Response Code: 400

Response Body:

```
Email or Username properties required.
```

Developed and tested by Brandon Cheng

Auth4c: Missing password property
Resource URI: POST /admin/login

Request Header: None

Database Status: Admin with email “admin@mrpizza.com” is registered

Request Body:

```
{  
    "email": "admin@mrpizza.com"  
}
```

Response Code: 400

Response Body:

```
Password property required.
```

Developed and tested by Brandon Cheng

Auth4d: Incorrect email and/or password
Resource URI: POST /admin/login

Request Header: None

Database Status: Admin with email “admin@mrpizza.com” is registered

Request Body:

```
{  
    "email": "admin@mrpizza.com",  
    "password": "admin212"  
}
```

Response Code: 401

Response Body:

```
Incorrect username or password.
```

Developed and tested by Brandon Cheng

Auth4e: Admin does not exist
Resource URI: POST /admin/login

Request Header: None

Database Status: Admin with email “admin@mrpizza.com” is registered

Request Body:

```
{  
    "email": "admin1@mrpizza.com",  
    "password": "admin223"  
}
```

Response Code: 401

Response Body:

```
Incorrect username or password.
```

Developed and tested by Brandon Cheng

Auth5: Get Auth Role

SQL Queries: None

Auth5a: Visitor/not signed in

Resource URI: GET /authRole

Request Headers: No Auth Token

Database Status:

User with username “test2” is registered

Employee with email “employee1@mrpizza.com” is registered

Admin with email “admin@mrpizza.com” is registered

Request Body: None

Response Code: 200

Response Body:

```
{  
    "authRole": "none"  
}
```

Developed and tested by Brandon Cheng

Auth5b: Signed in as customer

Resource URI: GET /authRole

Request Headers: Valid Auth Token, signed in as customer

Database Status:

User with username “test2” is registered

Employee with email “employee1@mrpizza.com” is registered

Admin with email “admin@mrpizza.com” is registered

Request Body: None

Response Code: 200

Response Body:

```
{  
  "authRole": "customer",  
  "id": 12,  
  "iat": 1712036861  
}
```

Developed and tested by Brandon Cheng

Auth5c: Signed in as employee

Resource URI: GET /authRole

Request Headers: Valid Auth Token, signed in as employee

Database Status:

User with username “test2” is registered

Employee with email “employee1@mrpizza.com” is registered

Admin with email “admin@mrpizza.com” is registered

Request Body: None

Response Code: 200

Response Body:

```
{  
  "authRole": "employee",  
  "id": 2,  
  "iat": 1712003937  
}
```

Developed and tested by Brandon Cheng

Auth5d: Signed in as admin

Resource URI: GET /authRole

Request Headers: Valid Auth Token, signed in as admin

Database Status:

User with username “test2” is registered

Employee with email “employee1@mrpizza.com” is registered

Admin with email “admin@mrpizza.com” is registered

Request Body: None

Response Code: 200

Response Body:

```
{  
  "authRole": "admin",  
  "id": 1,  
  "iat": 1712037671  
}
```

Developed and tested by Brandon Cheng

Auth6: Logout

SQL Queries Used: None

Auth6a: Successful logout

Resource URI: POST /logout

Request Headers: No Auth Token or Valid Auth Token (signed in as any role)

Request Body: None

Response Code: 200

Response Header:

`set-cookie: authToken=`

Response Body:

`Successful Logout, setting cookie`

Developed and tested by Brandon Cheng

Auth7: Get Employee Store Info

SQL Queries Used:

```
let queryRes = await runQuery(employeePool,
  `select *
   from employee_account e
   join store s on e.works_at = s.store_id
   where e.eid = ${jwtBody.id}`);
```

Auth7a: Successful fetch for employee store info

Resource URI: GET /employeeStoreInfo

Request Headers: Valid Auth Token, signed in as employee who works at a store

Database Status:

Employee with email “employee1@mrpizza.com” is registered

Employee with email “d1@mrpizza.com” is registered

d1 works at store_id 1

Request Body: None

Response Code: 200

Response Body:

```
{
  "eid": 2,
  "name": "d1",
  "employee_type": "driver",
  "email": "d1@mrpizza.com",
  "password_hash": "password",
  "status": "idle",
  "works_at": 1,
  "store_id": 1,
  "address": "busch student center",
  "latlng": {
    "x": 40.523421858838276,
    "y": -74.45823918823967
  },
  "image_url": "https://scheduling.rutgers.edu/sites/default/files/
    study-space-photos/IMG_5929.jpg"
}
```

Developed and tested by Brandon Cheng

Auth7b: Not logged in as an employee
Resource URI: GET /employeeStoreInfo

Request Headers: None

Database Status:
Employee with email “employee1@mrpizza.com” is registered
Employee works at store_id 1

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Brandon Cheng

Auth7c: Signed as an employee that is not assigned to store

Resource URI: GET /employeeStoreInfo

Request Headers: Valid Auth Token, signed in as an employee not working at a store

Database Status:
Employee with email “employee1@mrpizza.com” is registered
Employee with email “d1@mrpizza.com” is registered
d1 works at store_id 1

Request Body: None

Response Code: 401

Response Body:

Invalid employee.

Developed and tested by Brandon Cheng

Accounts1: Get Customer Info

SQL Queries Used:

```
let query = `SELECT * FROM customer_account WHERE cid = ${jwtBody.id}`;
```

Accounts1a: Successful fetch of customer info

Resource URI: GET /getCustomerInfo

Request Headers: Valid Auth Token, signed in as customer

Database Status:

User with username “test2” is registered

Request Body: None

Response Code: 200

Response Body:

```
{
  "cid": 12,
  "username": "test2",
  "default_delivery_address": null,
  "phone_num": null,
  "password_hash": "1234",
  "email": null,
  "default_credit_card": null
}
```

Developed and tested by Anna Yeakel

Accounts1b: Not signed in/wrong user role

Resource URI: GET /getCustomerInfo

Request Headers: None

Database Status:

User with username “test2” is registered

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Anna Yeakel

Accounts2: Edit Customer Info

SQL Queries Used:

```
// Query is generated through the following code, which uses the fields in
the request body.

. . .

try {
    let updateQuery = `UPDATE customer_account SET `;
    let updates = [];

    for (const key in decodedData) {
        if (decodedData.hasOwnProperty(key) && decodedData[key].length > 0){
            updates.push(` ${key} = ${decodedData[key]}`);
        }
    }
    if (updates.length == 0) {
        res.writeHead(400, {"Content-Type": "text/plain"});
        res.end("No data provided");
        return;
    }

    updateQuery += updates.join(', ');
    updateQuery += ` WHERE cid = ${jwtBody.id}`;
}
. . .
```

Accounts2a: Successful edit of customer information

Resource URI: POST /customer/accountInfo

Request Headers: Valid Auth Token, signed in as customer

Database Status: Currently signed-in user exists in the database

Request Body:

```
{
    "default_delivery_address" : "test address",
    "phone_num" : "1234567890",
    "email" : "example@gmail.com",
    "default_credit_card" : "1234567890123456"
}
```

Response Code: 200

Response Body:

Successfully updated

default_credit_card to be removed from database schema

Developed and tested by Anna Yeakel

Accounts2b: Not signed into a customer account
Resource URI: POST /customer/accountInfo

Request Headers: None

Database Status: Currently signed-in user exists in the database

Request Body:

```
{  
    "default_delivery_address" : "test address",  
    "phone_num" : "1234567890",  
    "email" : "example@gmail.com",  
    "default_credit_card" : "1234567890123456"  
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Anna Yeakel

Accounts3: Delete Account

SQL Queries Used:

```
try{
    let deleteQuery = await runQuery(customerPool, `DELETE FROM
customer_account WHERE cid = ${jwtBody.id}`);
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.end("Account successfully deleted");
}
```

Accounts3a: Deletes a registered account

Resource URI: POST /deleteAccount

Request Headers: Valid Auth Token, signed in as customer

Database Status: Currently signed-in user exists in the database

Request Body:

```
{
  "id" : 107
}
```

Response Code: 200

Response Body:

```
Account successfully deleted
```

Developed and tested by Anna Yeakel

Accounts4: Add new Employee

SQL Queries Used:

```
// Query implemented using the following code

try {
    let eid = await addEmployee(authRoles.employee, email, password);
    console.log(eid[0].id);
    await runQuery(adminPool, `UPDATE employee_account SET name =
`${name}`, works_at = ${works_at} where eid = ${eid[0].id}`)
    res.writeHead(201, {"Content-Type": "text/plain"})
    res.end("Successfully added employee");
}

async function addEmployee(authRole, userEmail, password) {
    let hash = await new Promise(resolve => {
        bcrypt.hash(password, saltRounds, (err, hash) => {
            resolve(hash);
        });
    })
    let res;
    if (authRole == authRoles.employee) {
        res = await runQuery(authPool, `insert into
employee_account(email, password_hash) values
(` + userEmail + `, ` + hash + `)`)
    } else {
        throw new Error('Invalid auth role');
    }
    let newQuery = `select eid as id, password_hash from employee_account e
where e.email = ${userEmail}`;
    let result = await runQuery(employeePool, newQuery, [userEmail,
password]);
    console.log(result);
    return result;
}
```

Accounts4a: Adds a new employee to the database
Resource URI: POST /admin/addNewEmployee

Request Headers: Valid Auth Token, signed in as an admin

Database Status: Currently signed-in admin exists in the database

Request Body:

```
{  
    "employeeFirstName" : "Test",  
    "employeeLastName" : "Name",  
    "employeeStoreId" : 1  
}
```

Response Code: 201

Response Body:

```
Successfully added employee
```

Developed and tested by Anna Yeakel

Accounts5: Assign Existing Employee

SQL Queries Used:

```
try{
    let updateRes = await runQuery(adminPool, `UPDATE employee_account
SET works_at = ${decodedData.newEmployeeStoreId} WHERE eid =
${decodedData.employeeId}`);
}
```

Accounts5a: Assigns an existing employee to a new store

Resource URI: POST /admin/assignExistingEmployee

Request Headers: Valid Auth Token, signed in as admin

Database Status: Currently signed-in admin exists in database

Request Body:

```
{
  "employeeId" : "2",
  "newEmployeeStoreId" : "2"
}
```

Response Code: 200

Response Body:

```
Successfully assigned employee
```

Developed and tested by Anna Yeakel

Accounts5b: Invalid employee ID

Resource URI: POST /admin/assignExistingEmployee

Request Headers: Valid Auth Token, signed in as admin

Database Status: Currently signed-in admin exists in database

Request Body:

```
{  
    "employeeId" : "200",  
    "newEmployeeStoreId" : "2"  
}
```

Response Code: 400

Response Body:

```
No employee found with the id 200
```

Developed and tested by Anna Yeakel

Accounts5c: Invalid store ID

Resource URI: POST /admin/assignExistingEmployee

Request Headers: Valid Auth Token, signed in as admin

Database Status: Currently signed-in admin exists in database

Request Body:

```
{  
    "employeeId" : "1",  
    "newEmployeeStoreId" : "10"  
}
```

Response Code: 404

Response Body:

```
No stores with id 10
```

Developed and tested by Anna Yeakel

Accounts6: Get Employee Info

SQL Queries Used:

```
// Query implemented using the following code

try{
    let updateQuery = `UPDATE employee_account SET`;
    let updates = [];

    for (const key in decodedData) {
        if (decodedData.hasOwnProperty(key) && decodedData[key].length >
0){
            updated.push(`$${key} = ${decodedData[key]}`);
        }
    }
    if (updated.length == 0){
        res.writeHead(400, {"Content-Type": "text/plain"});
        res.end("No data provided");
        return;
    }

    updateQuery += updated.join(', ');
    updateQuery += ` WHERE eid = ${jwtBody.id}`;

    await runQuery(employeePool, updateQuery);
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.end("Successfully updated");
}
```

Accounts6a: Fetches employee account info.

Resource URI: GET /employee/getInfo

Request Headers: Valid Auth Token, signed-in as employee

Database Status: Currently signed-in employee exists in database

Request Body:

None

Response Code: 200

Response Body:

Successfully updated

Developed and tested by Anna Yeakel

Accounts7: Edit Employee Info as Admin

SQL Queries Used:

```
// Query implemented using the following code

try {
    let updateRes = await runQuery(employeePool, `UPDATE
employee_account SET status = '${decodedData.employeeStatus}', employee_type
= '${decodedData.employeeType}' WHERE eid = ${decodedData.employeeId}`);
    if (updateRes.affectedRows == 0) {
        res.writeHead(400, {"Content-Type": "text/plain"});
        res.end("No employee found with the id " +
decodedData.employeeId);
    }
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.end("Successfully edited employee");
}
```

Accounts7a: Edit employee account information as an admin

Resource URI: POST /admin/editEmployee

Request Headers: Valid Auth Token, signed-in as admin

Database Status: Currently signed-in admin exists in database

Request Body:

```
{
  "employeeId" : 1,
  "employeeStatus" : "active",
  "employeeType" : "driver"
}
```

Response Code: 200

Response Body:

```
Successfully edited employee
```

Developed and tested by Anna Yeakel

Accounts7a: Edit employee with invalid employee ID

Resource URI: POST /admin/editEmployee

Request Headers: Valid Auth Token, signed-in as admin

Database Status: Currently signed-in admin exists in database

Request Body:

```
{  
    "employeeId" : 400,  
    "employeeStatus" : "active",  
    "employeeType" : "driver"  
}
```

Response Code: 400

Response Body:

```
No employee found with the id 400
```

Developed and tested by Anna Yeakel

Accounts8: Edit Employee Info as Employee

SQL Queries Used:

```
// Query implemented using the following code

try{
    let updateQuery = `UPDATE employee_account SET `;
    let updates = [];

    for (const key in decodedData) {
        if (decodedData.hasOwnProperty(key) && decodedData[key].length >
0){
            updates.push(` ${key} = '${decodedData[key]}'`);
        }
    }
    if (updates.length == 0){
        res.writeHead(400, {"Content-Type" : "text/plain"});
        res.end("No data provided");
    }

    updateQuery += updates.join(', ');
    updateQuery += ` WHERE eid = ${jwtBody.id}`;

    await runQuery(employeePool, updateQuery);
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.end("Successfully updated");
}
```

Accounts8a: Edit employee account information
Resource URI: POST /employee/accountInfo

Request Headers: Valid Auth Token, signed-in as employee

Database Status: Currently signed-in employee exists in database

Request Body:

```
{  
    "email" : "johnDoe@mrpizza.com",  
    "password_hash" : "321"  
}
```

Response Code: 200

Response Body:

```
Successfully updated
```

Developed and tested by Anna Yeakel

Support1: Create Support Ticket

SQL Queries Used:

```
const insertQuery = `INSERT INTO help_ticket (asked_by, question) VALUES (?, ?)`;
```

Support1a: Successful creation of a support ticket

Resource URI: POST /support/createTicket

Request Headers: Valid Auth Token, signed in as customer

Database Status: Ticket gets added to the database

Request Body:

```
{
    "description": "need help ordering"
}
```

Response Code: 201

Response Body:

Support ticket created successfully.

Developed and tested by Aman Patel

Support1b: Not logged in

Resource URI: POST /support/createTicket

Request Headers: None

Request Body:

```
{
    "description": "need help ordering"
}
```

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Aman Patel

Support2: Respond to Support Ticket

SQL Queries Used:

```
const updateQuery = `UPDATE help_ticket SET answer = ?, answered_by = ?
WHERE tid = ?`;
```

Support2a: Successful employee response to an existing support ticket

Resource URI: POST /support/respondToTicket

Request Headers: Valid Auth Token, signed in as employee

Database Status: Adds response to database entry of ticket with appropriate ticketId

Request Body:

```
{
  "ticketId":2,
  "response":"there is a button on the home page"
}
```

Response Code: 200

Response Body:

```
Response added to support ticket successfully.
```

Developed and tested by Aman Patel

Support2b: Not logged in

Resource URI: POST /support/respondToTicket

Request Headers: None

Request Body:

```
{
  "ticketId":2,
  "response":"there is a button on the home page"
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Aman Patel

Support3: View All Tickets

SQL Queries Used:

```
const query = `SELECT * FROM help_ticket`;
```

Support3a: Returns all tickets in the database

Resource URI: GET /support/tickets

Request Headers: Valid Auth Token, signed in as employee

Request Body: None

Response Code: 200

Response Body:

```
[
  {
    "tid": 1,
    "asked_by": 1,
    "answered_by": 1,
    "date_created": null,
    "question": "need help ordering",
    "answer": "there is a button on the home page",
    "quality_rating": null,
    "original_tid": null,
    "DTSTAMP": null
  },
  {
    "tid": 2,
    "asked_by": 107,
    "answered_by": 2,
    "date_created": null,
    "question": "need help ordering",
    "answer": "there is a button on the home page",
    "quality_rating": null,
    "original_tid": null,
    "DTSTAMP": null
  }
]
```

Developed and tested by Aman Patel

Support3b: Not logged in
Resource URI: GET /support/tickets

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Aman Patel

Support4: View Past Tickets

SQL Queries Used:

```
const query = `SELECT tid AS id, question AS description, answer AS response
FROM help_ticket WHERE asked_by = ?`;
```

Support4a: Returns signed in customer's tickets

Resource URI: GET /support/pastTickets

Request Headers: Valid Auth Token, signed in as customer

Request Body: None

Response Code: 200

Response Body:

```
[
  {
    "id": 2,
    "description": "need help ordering",
    "response": "there is a button on the home page"
  }
]
```

Developed and tested by Aman Patel

Support4b: Not logged in

Resource URI: GET /support/pastTickets

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Aman Patel

Support5: View Unanswered Tickets

SQL Queries Used:

```
const query = `SELECT *, tid AS id, asked_by, question AS description FROM help_ticket WHERE answer IS NULL`;
```

Support5a: Returns unanswered tickets for employees

Resource URI: GET /support/unansweredTickets

Request Headers: Valid Auth Token, signed in as employee

Request Body: None

Response Code: 200

Response Body:

```
[
  {
    "tid": 3,
    "asked_by": 107,
    "answered_by": null,
    "date_created": null,
    "question": "how to review",
    "answer": null,
    "quality_rating": null,
    "original_tid": null,
    "DTSTAMP": null,
    "id": 3,
    "description": "how to review"
  },
  {
    "tid": 4,
    "asked_by": 107,
    "answered_by": null,
    "date_created": null,
    "question": "need help ordering",
    "answer": null,
    "quality_rating": null,
    "original_tid": null,
    "DTSTAMP": null,
    "id": 4,
    "description": "need help ordering"
  }
]
```

Developed and tested by Aman Patel

Support5b: Not logged in
Resource URI: GET /support/unansweredTickets

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Aman Patel

Analytics1: Get Total Revenue By Date

SQL Queries Used:

```
const query = 'SELECT IFNULL(SUM(total_price), 0) AS total_revenue FROM customer_order WHERE DT_CREATED BETWEEN ? AND ?';
```

Analytics1a: Gets the total company revenue from a specific time period

Resource URI: POST /getTotalCompanyRevenue

Request Headers: Valid Auth Token, Signed in as admin/employee

Database Status: None

Request Body:

```
{  
    "startDate": "2024-01-01",  
    "endDate": "2024-12-31"  
}
```

Response Code: 200 OK

Response Body:

```
{  
    "total_revenue": 9737.909856081007  
}
```

Developed and tested by Ethan Sie and Brandon

Analytics1b: Not logged in.

Resource URI: POST /getTotalCompanyRevenue

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

```
{  
    "startDate": "2024-01-01",  
    "endDate": "2024-12-31"  
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Ethan Sie and Brandon

Analytics2: View Employee Count By Store

SQL Queries Used:

```
const query = `SELECT s.address, COUNT(e.eid) AS numEmployees
FROM store s
LEFT JOIN employee_account e ON s.store_id = e.works_at
GROUP BY s.store_id, s.address;
`;
```

Analytics2a: View the Employee count by their store/location

Resource URI: GET /getEmployeeCountByStore

Request Headers: Valid Auth Token, Signed in as Admin

Database Status: None

Request Body:

None

Response Code: 200

Response Body:

```
[
  {
    "address": "604 Bartholomew Rd, Piscataway, NJ 08854",
    "numEmployees": 2
  },
  {
    "address": "84 Joyce Kilmer Ave, Piscataway, NJ 08854",
    "numEmployees": 0
  }
]
```

Developed and tested by Ethan Sie and Brandon

Analytics2b: View the Employee count by their store/location (Not Logged In)
Resource URI: GET /getEmployeeCountByStore

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Ethan Sie and Brandon

Analytics3: View Total Customers

SQL Queries Used:

```
const query = 'SELECT COUNT(*) AS numCustomers FROM customer_account';
```

Analytics3a: View the total amount of current customers

Resource URI: GET /getTotalCustomers

Request Headers: Valid Auth Token, Signed in as Employee/Admin

Database Status: None

Request Body:

```
None
```

Response Code: 200

Response Body:

```
{
    "numCustomers": 106
}
```

Developed and tested by Ethan Sie and Brandon

Analytics3b: Not logged in.

Resource URI: GET /getTotalCustomers

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

```
None
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Ethan Sie and Brandon

Analytics4: View Total Employees

SQL Queries Used:

```
const query = 'SELECT COUNT(*) AS numEmployees FROM employee_account';
```

Analytics4a: Views the total amount of employees.

Resource URI: GET /getTotalEmployees

Request Headers: Valid Auth Token, Signed in as Employee

Database Status: None

Request Body:

```
None
```

Response Code: 200

Response Body:

```
{
    "numEmployees": 3
}
```

Developed and tested by Ethan Sie and Brandon

Analytics4b: Not logged in.

Resource URI: GET /getTotalEmployees

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

```
None
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Ethan Sie and Brandon

Analytics5: View Menu Items Sorted By Popularity

SQL Queries Used:

```
const query = `SELECT m.description, COUNT(*) AS timesOrdered
FROM menu_item m
JOIN order_item oi ON m.mid = oi.mid
GROUP BY m.description
ORDER BY timesOrdered DESC`;
```

Analytics5a: Sorts menu items by the total amount of orders made for that specific item.

Resource URI: GET /getMenuItemsSortedByPopularity

Request Headers: Valid Auth Token, Signed in as Admin

Database Status: None

Request Body:

None

Response Code: 200

Response Body:

```
[
  {
    "description": "Handmade from fresh buttery-tasting dough and
baked to a golden brown. Crusty on the outside and soft on the inside.
Drizzled with a perfect blend of cinnamon and sugar, and best served with
a side of sweet icing for dipping or drizzling.",
    "timesOrdered": 19
  },
  {
    "description": "Oven-baked bread bites handmade from fresh
buttery-tasting dough and seasoned with garlic and Parmesan. Available in
16-piece or 32-piece orders. Add marinara or your favorite dipping cup for
an additional charge.",
    "timesOrdered": 19
  },
  {
    "description": "Mr. Pizza's own spicy buffalo sauce.",
```

```
        "timesOrdered": 18
    },
    {
        "description": "Made with the goodness of real lemons, Minute
Maid® Lemonade is the quintessential refreshing beverage with the great
taste of a simpler time.",
        "timesOrdered": 17
    },
    {
        "description": "Grilled chicken breast, ranch, smoked bacon, diced
tomatoes, provolone and cheese made with 100% real mozzarella.",
        "timesOrdered": 17
    },
    {
        "description": "Two layers of pepperoni sandwiched between
provolone, Parmesan-Asiago and cheese made with 100% real mozzarella then
sprinkled with oregano.",
        "timesOrdered": 16
    },
...
}
```

Developed and tested by Ethan Sie and Brandon

Analytics5b: Sorts menu items by the total amount of orders made for that specific item. (Not Logged In)

Resource URI: GET /getMenuItemsSortedByPopularity

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Ethan Sie and Brandon

Analytics6: View Revenue Total Per Menu Item

SQL Queries Used:

```
const query = `SELECT m.description, SUM(m.price) AS totalRevenue
FROM menu_item m
JOIN order_item oi ON m.mid = oi.mid
GROUP BY m.description, wc.option_name
ORDER BY totalRevenue DESC`;
```

Analytics6a: Tells the total revenue generated by a specific menu item.

Resource URI: GET /getRevenueTotalByMenuItem

Request Headers: Valid Auth Token, Signed in as Admin

Database Status: None

Request Body:

None

Response Code: 200

Response Body:

```
[
  {
    "item": "Cali Chicken Bacon Ranch",
    "description": "Grilled chicken breast, ranch, smoked bacon, diced tomatoes, provolone and cheese made with 100% real mozzarella.",
    "totalRevenue": 373.8299961090088
  },
  {
    "item": "Ultimate Pepperoni",
    "description": "Two layers of pepperoni sandwiched between provolone, Parmesan-Asiago and cheese made with 100% real mozzarella then sprinkled with oregano.",
    "totalRevenue": 351.8399963378906
  },
  {
    "item": "Pacific Veggie",
```

```
        "description": "Fresh baby spinach, fresh onions, fresh mushrooms, tomatoes, black olives, feta, provolone, cheese made with 100% real mozzarella and sprinkled with a garlic herb seasoning.",  
        "totalRevenue": 307.8599967956543  
    },  
    {  
        "item": "Wisconsin 6 Cheese",  
        "description": "Feta, provolone, cheddar, Parmesan-Asiago, cheese made with 100% real mozzarella and sprinkled with oregano.",  
        "totalRevenue": 285.86999702453613  
    },  
    {  
        "item": "MeatZZa",  
        "description": "Pepperoni, ham, Italian sausage and beef, all sandwiched between two layers of provolone and cheese made with 100% real mozzarella.",  
        "totalRevenue": 285.86999702453613  
    },  
    {  
        "item": "Buffalo Chicken",  
        "description": "Grilled chicken breast, fresh onions, provolone, American cheese, cheddar, cheese made with 100% real mozzarella and drizzled with a hot buffalo sauce.",  
        "totalRevenue": 285.86999702453613  
    },  
]
```

Developed and tested by Ethan Sie and Brandon

Analytics6b: Not logged in

Resource URI: GET /getRevenueTotalByMenuItem

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Ethan Sie and Brandon

Analytics7: View Toppings Sorted By Popularity

SQL Queries Used:

```
const query = `SELECT m.description, wc.option_name, COUNT(*) AS timesOrdered
FROM menu_item m
JOIN with_custom wc ON wc.mid = m.mid
WHERE wc.custom_name = 'toppings'
GROUP BY m.description, wc.option_name
ORDER BY timesOrdered DESC`;
```

Analytics7a: Views the most popular toppings by amount ordered.

Resource URI: GET /getMostPopularToppings

Request Headers: Valid Auth Token, Signed in as Admin

Database Status: None

Request Body:

None

Response Code: 200

Response Body:

```
[
  {
    "item": "Pasta Primavera",
    "description": "Fresh baby spinach, diced tomatoes, fresh mushrooms and fresh onions, mixed with penne pasta and baked with a creamy Alfredo sauce.",
    "option_name": "No Diced Tomatoes",
    "timesOrdered": 11
  },
  {
    "item": "Chicken Bacon Ranch",
    "description": "Enjoy our flavorful grilled chicken breast topped with smoked bacon, creamy ranch and provolone cheese on artisan bread baked to golden brown perfection.",
    "option_name": "No Premium Chicken",
    "timesOrdered": 11
  }
]
```

```
},
{
    "item": "Specialty Chicken - Sweet BBQ Bacon",
    "description": "Tender bites of breaded chicken made with 100% whole breast white meat topped with sweet and smoky honey BBQ sauce, a blend of cheese made with mozzarella and cheddar, and crispy bacon.",
    "option_name": "No Bacon",
    "timesOrdered": 10
},
{
    "item": "Pasta Primavera",
    "description": "Fresh baby spinach, diced tomatoes, fresh mushrooms and fresh onions, mixed with penne pasta and baked with a creamy Alfredo sauce.",
    "option_name": "No Mushrooms",
    "timesOrdered": 10
},
{
    "item": "Specialty Chicken - Crispy Bacon & Tomato",
    "description": "Tender bites of breaded chicken made with 100% whole breast white meat topped with garlic Parmesan sauce, a blend of cheese made with mozzarella and cheddar, crispy bacon and tomato.",
    "option_name": "No Bacon",
    "timesOrdered": 9
},
...
]
```

Developed and tested by Ethan Sie and Brandon

Analytics7b: Not logged in.

Resource URI: GET /getMostPopularToppings

Request Headers: None (Missing Auth)

Database Status: None

Request Body:

None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Ethan Sie and Brandon

Map/Driver Use Cases

Map1: View Store Locations

SQL Queries Used:

```
let stores = await runQuery(visitorPool, 'SELECT store_id, latlng FROM store');
```

Resource URI: GET /stores

Request Headers: None

Request Body: None

Response Code: 200

Response Body: (Example Stores)

```
[  
  {  
    "store_id": 1,  
    "latlng": {  
      "x": 40.523421858838276,  
      "y": -74.45823918823967  
    }  
  },  
  {  
    "store_id": 2,  
    "latlng": {  
      "x": 40.52362753497832,  
      "y": -74.43692635431962  
    }  
  }  
]
```

Developed and tested by Ji Wu

Map2: View Detailed Store Information

SQL Queries Used:

```
let storeInfo = await runQuery(visitorPool, `SELECT * FROM store s WHERE s.store_id = "${storeId}"`);
```

Map2a: Store ID exists

Resource URI: GET/stores/1

Request Header: None

Request Body: None

Response Code: 200

Response Body:

```
{
  "store_id": 1,
  "name": "Busch Student Center",
  "address": "604 Bartholomew Rd, Piscataway, NJ 08854",
  "latlng": {
    "x": 40.523421858838276,
    "y": -74.45823918823967
  },
  "image_url": "https://scheduling.rutgers.edu/sites/default/files/
                study-space-photos/IMG_5929.jpg"
}
```

Developed and tested by Ji Wu

Map2b: Store ID does not exist

Resource URI: GET/store/20

Request Header: None

Request Body: None

Response Code: 404

Response Body:

```
Store with storeId 20 not found.
```

Developed and tested by Ji Wu

Map3: Add Store Locations

SQL Queries Used:

```
let decodedData = await getJSONBody(req, res, ['address', 'latitude', 'longitude']);
let queryRes = await runQuery(adminPool,
    `insert into store(address, latlng) values ("${decodedData.address}", point(${decodedData.latitude}, ${decodedData.longitude}))`);
```

Map3a: Successful Store Add

Resource URI: POST/stores/add

Request Header: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
  "address": "TEST STORE",
  "Latitude": "40",
  "Longitude": "-74"
}
```

Response Code: 201

Response Body:

```
Successfully added store
```

Developed and tested by Ji Wu

Map3b: Not logged in as Admin

Resource URI: POST/stores/add

Request Header: None

Request Body:

```
{
  "address": "TEST STORE",
  "Latitude": "40",
  "Longitude": "-74"
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Ji Wu

Map3c: Missing parameters from input body.

Resource URI: POST/stores/add

Request Header: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{  
}  
}
```

Response Code: 400

Response Body:

```
{  
    "error": "Missing required properties",  
    "missingProperties": [  
        "address",  
        "latitude",  
        "longitude"  
    ]  
}
```

Developed and tested by Ji Wu

Map4: Edit Store Locations

SQL Queries Used:

```
let decodedData = await getJSONBody(req, res, ['address', 'latitude', 'longitude']);

let queryRes = await runQuery(adminPool,
    `UPDATE store SET address = "${decodedData.address}", latlng =
    point(${decodedData.latitude}, ${decodedData.longitude}) WHERE store_id =
    ${storeId}`);
```

Map4a: Successful store edit

Resource URI: POST /store/1/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
    "address": "busch student center!",
    "latitude": "40.523421858838276",
    "longitude": "-74.45823918823967"
}
```

Response Code: 200

Response Body:

```
Successfully Edited Store
```

Developed and tested by Ji Wu and Damon Lin

Map4b: Store ID does not exist
Resource URI: POST /store/10/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{  
    "address": "busch student center!",  
    "latitude": "40.523421858838276",  
    "longitude": "-74.45823918823967"  
}
```

Response Code: 404

Response Body:

```
Store not Found.
```

Developed and tested by Ji Wu and Damon Lin

Map4c: Missing required properties
Resource URI: POST /store/1/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{  
    "address": "busch student center!",  
    "latitude": "40.523421858838276",  
}
```

Response Code: 400

Response Body:

```
{  
    "error": "Missing required properties",  
    "missingProperties": [  
        "longitude"  
    ]  
}
```

Developed and tested by Ji Wu and Damon Lin

Map3d: Not logged in as Admin
Resource URI: /stores/1/edit

Request Header: None

Request Body:

```
{  
    "address": "TEST STORE",  
    "Latitude": "40",  
    "Longitude": "-74"  
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Ji Wu and Damon Lin

Map5: Get List of Store Names

SQL Queries Used:

```
let stores = await runQuery(visitorPool, 'SELECT store_id, name FROM store');
```

Map5a: Success

Resource URI: GET /storeNames

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body: None

Response Code: 200

Response Body:

```
[  
  {  
    "store_id": 1,  
    "name": "Busch Student Center"  
  },  
  {  
    "store_id": 2,  
    "name": "Livingston Student Center"  
  }]
```

Developed and tested by Brandon Cheng

Map6: Validate Input Address

SQL Queries Used:

None

Map6a: Accepted address

Resource URI: POST /validateAddress

Request Headers: None

Request Body:

```
{  
    "regionCode": "US",  
    "locality": "Piscataway",  
    "administrativeArea": "NJ",  
    "postalCode": "08854",  
    "addressLines": [  
        "220 Marvin Lane",  
        "  
    ]  
}
```

Response Code: 200

Response Body:

```
{  
    "validationResult": "ACCEPTED",  
    "formattedAddress": "220 Marvin Lane, Piscataway, NJ 08854, USA",  
    "location": {  
        "latitude": 40.5187174,  
        "longitude": -74.4553767  
    }  
}
```

Developed and tested by Ji Wu, Brandon Cheng and Damon Lin

Map6b: Inferred address - verification required
Resource URI: POST /validateAddress

Request Headers: None

Request Body:

```
{  
    "regionCode": "US",  
    "locality": "Piscataway",  
    "administrativeArea": "NJ",  
    "postalCode": "",  
    "addressLines": [  
        "220 Marvin Lane",  
        ""  
    ]  
}
```

Response Code: 200

Response Body:

```
{  
    "validationResult": "VERIFY_REQUIRED",  
    "formattedAddress": "220 Marvin Lane, Piscataway, NJ 08854, USA",  
    "location": {  
        "latitude": 40.5187174,  
        "longitude": -74.4553767  
    }  
}
```

Developed and tested by Ji Wu, Brandon Cheng and Damon Lin

Map6c: Invalid address - fix required
Resource URI: POST /validateAddress

Request Headers: None

Request Body:

```
{  
    "regionCode": "US",  
    "locality": "Piscataway",  
    "administrativeArea": "NJ",  
    "postalCode": "",  
    "addressLines": [  
        "hello",  
        ""  
    ]  
}
```

Response Code: 200

Response Body:

```
{  
    "validationResult": "FIX_REQUIRED",  
    "warningFields": [  
        "addressLine1"  
    ]  
}
```

Developed and tested by Ji Wu, Brandon Cheng and Damon Lin

Map6d: Missing required properties
Resource URI: POST /validateAddress

Request Headers: None

Request Body:

```
{  
}
```

Response Code: 400

Response Body:

```
{  
    "error": "Missing required properties",  
    "missingProperties": [  
        "regionCode",  
        "locality",  
        "administrativeArea",  
        "postalCode",  
        "addressLines"  
    ]  
}
```

Developed and tested by Ji Wu, Brandon Cheng and Damon Lin

Map6e: Location not in NJ

Resource URI: POST /validateAddress

Request Headers: None

Request Body:

```
{  
    "regionCode": "US",  
    "locality": "Mountain View",  
    "administrativeArea": "CA",  
    "postalCode": "",  
    "addressLines": [  
        "1600 Amphitheatre Parkway",  
        ""  
    ]  
}
```

Response Code: 200

Response Body:

```
{  
    "validationResult": "FIX_REQUIRED",  
    "warningFields": [  
        "regionCode",  
        "administrativeArea"  
    ]  
}
```

Developed and tested by Ji Wu, Brandon Cheng and Damon Lin

Map6f: Failed to fetch from Google Address Validation API
Resource URI: POST /validateAddress

Request Headers: None

Request Body:

```
{  
    "regionCode": "US",  
    "locality": "Piscataway",  
    "administrativeArea": "NJ",  
    "postalCode": "08854",  
    "addressLines": [  
        "220 Marvin Lane",  
        ""  
    ]  
}
```

Response Code: 500

Response Body:

```
{  
    error: "Failed to fetch from Address Validation API."  
}
```

Developed and tested by Ji Wu, Brandon Cheng and Damon Lin

Delivery1: View Unassigned Orders

SQL Queries Used:

```
let query = `select o.order_id, o.delivery_latlng, o.DT_created,
    c.username, timestampdiff(second, o.DT_created, utc_timestamp())
timeSinceCreation
    from customer_order o
    join customer_account c on o.ordered_by = c.cid
    where o.made_at = ${storeId}
    and o.status = "in transit"
    and o.order_id not in (
        select order_id
        from in_batch
    )`;
```

```

Delivery1a: Successful

Resource URI: GET /unassigned/{store\_id}

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body: None

Response Code: 200

Response Body:

```
[
 {
 "order_id": 1,
 "delivery_latlng": {
 "x": 40.41829236146375,
 "y": -74.70639548636213
 },
 "DT_created": "2024-04-02T00:03:34.000Z",
 "username": "deliveryCust1",
 "timeSinceCreation": 15,
 "timeEstimate": "2394s"
 },
 {
 "order_id": 2,
 "delivery_latlng": {
 "x": 40.42837937050338,
 "y": -74.67260209649424
 },
 "DT_created": "2024-04-02T00:03:34.000Z",
 "username": "deliveryCust2",
 "timeSinceCreation": 15,
 "timeEstimate": "2310s"
 }
]
```

```
[{"order_id": 3, "delivery_latlng": {"x": 40.42837937050338, "y": -74.63260209649424}, "DT_created": "2024-04-02T00:03:34.000Z", "username": "deliveryCust2", "timeSinceCreation": 15, "timeEstimate": "1963s"}, {"order_id": 4, "delivery_latlng": {"x": 40.6573258481948, "y": -74.3632707679412}, "DT_created": "2024-04-02T00:03:34.000Z", "username": "deliveryCust2", "timeSinceCreation": 15, "timeEstimate": "2222s"}, {"order_id": 5, "delivery_latlng": {"x": 40.646906977241095, "y": -74.34146977403721}, "DT_created": "2024-04-02T00:03:34.000Z", "username": "deliveryCust2", "timeSinceCreation": 15, "timeEstimate": "2138s"}, {"order_id": 6, "delivery_latlng": {"x": 40.6622742419765, "y": -74.31554890726946}, "DT_created": "2024-04-02T00:03:34.000Z", "username": "deliveryCust2", "timeSinceCreation": 15, "timeEstimate": "2281s"}]
```

Delivery1b: Not signed in as employee  
Resource URI: GET /unassigned/{store\_id}

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Brandon Cheng

Delivery1c: Signed in as an Employee but in wrong store ID.  
Resource URI: GET /unassigned/2

Request Headers: Valid Auth Token Cookie, signed in as employee 2

Request Body: None

Response Code: 401

Response Body:

Employee does not work at the store with id 2.

Developed and tested by Brandon Cheng

Delivery1d: No unassigned orders.  
Resource URI: GET /unassigned/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body: None

Response Code: 200

Response Body:

[  
]

Developed and tested by Brandon Cheng



## Delivery2: See Available Drivers

SQL Queries Used:

```
return runQuery(employeePool,
 `select e.eid, e.email, e.status
 from employee_account e
 where e.employee_type = "driver"
 and e.status = "idle"`)
```

Delivery2a: Successful fetch

Resource URI: GET /availableDrivers/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body: None

Response Code: 200

Response Body:

```
{
 "eid": 2,
 "email": "d1@mrpizza.com",
 "status": "idle"
},
{
 "eid": 3,
 "email": "d2@mrpizza.com",
 "status": "idle"
}
```

Developed and tested by Brandon Cheng

Delivery2b: Not signed in as Employee

Resource URI: GET /availableDrivers/1

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Brandon Cheng

Delivery2c: Signed in as Employee but with wrong store ID

Resource URI: GET /availableDrivers/2

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with different store\_id

Request Body: None

Response Code: 401

Response Body:

Employee does not work at the store with id 2.

Developed and tested by Brandon Cheng

Delivery2d: No available drivers

Resource URI: GET /availableDrivers/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body: None

Response Code: 200

Response Body:

[]

Developed and tested by Brandon Cheng

## Delivery3: Calculate Optimal Assignment

SQL Queries Used:

```
// to verify that the orders are unassigned
let query = `select o.order_id, o.delivery_latlng, o.DT_created,
 c.username, timestampdiff(second, o.DT_created, utc_timestamp())
timeSinceCreation
 from customer_order o
 join customer_account c on o.ordered_by = c.cid
 where o.made_at = ${storeId}
 and o.status = "in transit"
 and o.order_id not in (
 select order_id
 from in_batch
)`;
// to verify that the drivers are available
return runQuery(employeePool,
 `select e.eid, e.email, e.status
 from employee_account e
 where e.employee_type = "driver"
 and e.status = "idle"`)
```

### Delivery3a: Successful Calculation

Resource URI: POST /optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]

Drivers with eids 1, 2 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": 3
}
```

Response Code: 200

Response Body:

```
[
 {
 "eid": 2,
 "orders": [
 3,
 2,
 1
]
 }
]
```

```
]
 },
{
 "eid": 3,
 "orders": [
 5,
 4,
 6
]
}]
```

Developed and tested by Brandon Cheng

Delivery3b: Not logged in as an employee  
Resource URI: POST /optimalAssignment/1

Request Headers: None

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]  
Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": 3
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Brandon Cheng

Delivery3c: Signed in as Employee but with wrong store ID

Resource URI: POST /optimalAssignment/2

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with different store\_id

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": 3
}
```

Response Code: 401

Response Body:

```
Employee does not work at the store with id 2.
```

Developed and tested by Brandon Cheng

Delivery3d: Missing required properties

Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing required properties",
 "missingProperties": [
 "drivers",
 "maxPerDriver"
]
}
```

Developed and tested by Brandon Cheng

Delivery3e: Invalid properties - maxPerDriver is not an integer  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": 3.2
}
```

Response Code: 400

Response Body:

```
Positive integer required for maxPerDriver
```

Developed and tested by Brandon Cheng

Delivery3f: Invalid properties - maxPerDriver is not an integer  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": "hello"
}
```

Response Code: 400

Response Body:

```
Positive integer required for maxPerDriver
```

Developed and tested by Brandon Cheng

Delivery3g: Invalid properties - drivers is not an array  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "drivers": "hello",
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

```
non-empty drivers array required
```

Developed and tested by Brandon Cheng

Delivery3h: Invalid properties - drivers array is empty  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "drivers": [],
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

```
non-empty drivers array required
```

Developed and tested by Brandon Cheng

Delivery3i: Driver ID does not exist  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]  
Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [28],
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

```
Driver with id 28 is not available.
```

Developed and tested by Brandon Cheng

Delivery3j: Driver with chosen ID is not available  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]  
Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [2, 10],
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

```
Driver with ID 10 is not available
```

Developed and tested by Brandon Cheng

Delivery3k: Not enough drivers for orders.  
Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]  
Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [2],
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

The server picks the oldest unassigned orders, then finds the best permutation.

```
[
 {
 "eid": 2,
 "orders": [
 3,
 2,
 1
]
 }
]
```

Developed and tested by Brandon Cheng

Delivery3l: More drivers than orders.

Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1]

Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

The server assigns one order to one driver, and nothing to the other.

```
[
 {
 "eid": 2,
 "orders": [
 1
]
 }
 {
 "eid": 3,
 "orders": [
]
 }
]
```

Developed and tested by Brandon Cheng

Delivery3m: More driver capacity than orders.

Resource URI: POST/optimalAssignment/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3]

Drivers with eids 2, 3 work for store\_id 1,

unassigned orders are for store\_id 1

Request Body:

```
{
 "drivers": [2, 3],
 "maxPerDriver": 3
}
```

Response Code: 400

Response Body:

The server picks best assignment prioritizing customer delay.

```
[
 {
 "eid": 2,
 "orders": [
 1
]
 },
 {
 "eid": 3,
 "orders": [
 3,
 2
]
 }
]
```

Developed and tested by Brandon Cheng

## Delivery4: Assign Orders To Drivers

SQL Queries Used:

```
// to verify that the orders are unassigned
let query = `select o.order_id, o.delivery_latlng, o.DT_created,
 c.username, timestampdiff(second, o.DT_created, utc_timestamp())
timeSinceCreation
 from customer_order o
 join customer_account c on o.ordered_by = c.cid
 where o.made_at = ${storeId}
 and o.status = "in transit"
 and o.order_id not in (
 select order_id
 from in_batch
)`;
// to verify that the driver exists
let existRes = await runQuery(employeePool,
 `select * from employee_account where eid = ${eid}`);
// to verify that the driver is not assigned an order
let checkRes = await runQuery(employeePool,
 `select * from delivery_batch where assignedToEmp = ${eid}`);
// add the delivery batch
let batchRes = await runQuery(employeePool,
 `insert into delivery_batch(driver_status, assignedToEmp) values
 ("idle", ${eid})`);
let batchId = (await runQuery(employeePool,
 `select batch_id from delivery_batch where assignedToEmp =
 ${eid}`))[0].batch_id;
// add the orders to the batch
let inbatchQuery = `insert into in_batch(order_id, batch_id, order_index)
 values `;
inbatchQuery += orders.map((o, i) => `(${o}, ${batchId}, ${i})`).join(', ')
let inbatchRes = await runQuery(employeePool, inbatchQuery);
// set status of the driver
let updateDriverRes = await runQuery(employeePool,
 `update employee_account set status = "assigned" where eid = ${eid}`)
```

Delivery4a: Successful assign

Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id

Request Body:

```
{}
```

```
{
 "eid": 2,
 "orders": [3,2,1]
}
```

Response Code: 200

Response Body:

**Orders assigned successfully**

Developed and tested by Brandon Cheng

Delivery4b: Driver is already assigned a delivery batch

Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Driver 1 is not available.

Request Body:

```
{
 "eid": 2,
 "orders": [3,2,1]
}
```

Response Code: 400

Response Body:

**Driver is not available**

Developed and tested by Brandon Cheng and Damon Lin

Delivery4c: Not logged in as an employee

Resource URI: POST/assignOrders/1

Request Headers: None

Database Status: Available drivers ids: [2, 3], Unassigned order ids: [1, 2, 3, 4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id

Request Body:

```
{
```

```
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4d: Signed in as Employee but with wrong store ID

Resource URI: POST/assignOrders/2

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with different store\_id

Request Body:

```
{
 "eid": 2,
 "orders": [3,2,1]
}
```

Response Code: 401

Response Body:

```
Employee does not work at the store with id 2.
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4e: Missing required properties  
Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
}
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing required properties",
 "missingProperties": [
 "orders",
 "eid"
]
}
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4f: Invalid properties - eid is not an integer  
Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "eid": "hello",
 "orders": [3,2,1]
}
```

Response Code: 400

Response Body:

```
integer eid values are required
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4g: Invalid properties - orders is not an array  
Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "eid": 2,
 "orders": "hello"
}
```

Response Code: 400

Response Body:

```
non-empty orders array required
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4h: Invalid properties - orders array is empty  
Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "eid": 2,
 "orders": []
}
```

Response Code: 400

Response Body:

```
non-empty orders array required
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4i: Driver does not exist  
Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Request Body:

```
{
 "eid": 10,
 "orders": [3,2,1]
}
```

Response Code: 400

Response Body:

```
Driver does not exist
```

Developed and tested by Brandon Cheng and Damon Lin

Delivery4j: Order is already assigned  
Resource URI: POST/assignOrder/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]  
Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1

Request Body:

```
{
 "eid": 2,
 "orders": [3]
}
```

Response Code: 400

Response Body:

```
Order with id 3 is not available.
```

Developed and tested by Brandon Cheng and Damon Lin

## Delivery5: Get Assigned Order Information

SQL Queries Used:

```
// this query checks whether or not the user is a driver
let roleRes = await runQuery(employeePool,
 `select * from employee_account where employee_type = "driver"`);

// this query gets the assigned orders for the signed in driver
let queryRes = await runQuery(employeePool,
 `SELECT o.*
 FROM customer_order o
 JOIN in_batch ib on ib.order_id = o.order_id
 JOIN delivery_batch db on db.batch_id = ib.batch_id
 WHERE db.assignedToEmp = ${jwtBody.id}
 ORDER BY ib.order_index ASC`);
```

Delivery5a: Successful Fetch.

Resource URI: GET/directions/assignedOrder

Request Headers: Valid Auth Token Cookie, signed in as an driver for the store with the same store\_id.

Request Body: None

Response Code: 200

Response Body:

```
[
 {
 "order_id": 3,
 "credit_card": null,
 "status": "in transit",
 "total_price": null,
 "delivery_address": null,
 "delivery_latlng": {
 "x": 40.42837937050338,
 "y": -74.63260209649424
 },
 "DT_created": "2024-04-02T08:08:10.000Z",
 "DT_delivered": null,
 "ordered_by": 4,
 "made_at": 1
 },
 {
 "order_id": 2,
 "credit_card": null,
 "status": "in transit",
 "total_price": null,
 "delivery_address": null,
 "delivery_latlng": {
```

```
 "x": 40.42837937050338,
 "y": -74.67260209649424
 },
 "DT_created": "2024-04-02T08:08:10.000Z",
 "DT_delivered": null,
 "ordered_by": 4,
 "made_at": 1
},
{
 "order_id": 1,
 "credit_card": null,
 "status": "in transit",
 "total_price": null,
 "delivery_address": null,
 "delivery_latlng": {
 "x": 40.41829236146375,
 "y": -74.70639548636213
 },
 "DT_created": "2024-04-02T08:08:10.000Z",
 "DT_delivered": null,
 "ordered_by": 3,
 "made_at": 1
}
]
```

Developed and tested by Damon Lin

Delivery5b: Is not signed in as a driver.

Resource URI: POST/assignOrders/1

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Damon Lin

Delivery5c:No orders have been assigned

Resource URI: POST/assignOrders/1

Request Headers: Valid Auth Token Cookie, signed in as an driver for the store with the same store\_id.

Request Body: None

Response Code: 200

Response Body:

```
[
]
```

Developed and tested by Damon Lin

## Delivery6: Get Waypoints for Driving

SQL Queries Used:

```
// this query checks whether or not the user is a driver
let roleRes = await runQuery(employeePool,
 `select * from employee_account where eid = ${jwtBody.id} and
employee_type = "driver"`);

// this query gets the waypoints for each order
let queryRes = await runQuery(employeePool,
 `select delivery_latlng from customer_order where order_id =
${order_id}`);
```

Delivery6a: Success

Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,

unassigned orders are for store\_id 1,

Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": [3,2,1]
}
```

Response Code: 200

Response Body:

The first and last coordinates correspond with the Mr. Pizza store location

```
[
 {
 "x": 40.523421858838276,
 "y": -74.45823918823967
 },
 {
 "x": 40.42837937050338,
 "y": -74.63260209649424
 },
 {
 "x": 40.42837937050338,
 "y": -74.67260209649424
 },
 {
 "x": 40.41829236146375,
```

```
 "y": -74.70639548636213
 },
{
 "x": 40.523421858838276,
 "y": -74.45823918823967
}
]
```

Developed and tested by Damon Lin

Delivery6b: Not logged in as an employee  
Resource URI: GET/directions/waypoints/1

Request Headers: None

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]  
Drivers with eids 2, 3 work for store\_id 1,  
unassigned orders are for store\_id 1,  
Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": [3,2,1]
}
```

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Damon Lin

Delivery6c: Not logged in as an employee that is a driver  
 Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id but with employee\_type that is not driver

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]  
 Drivers with eids 2, 3 work for store\_id 1,  
 unassigned orders are for store\_id 1,  
 Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": [3,2,1]
}
```

Response Code: 401

Response Body:

```
User is not a driver
```

Developed and tested by Damon Lin

Delivery6d: Signed in as Employee but with wrong store ID  
 Resource URI: GET/directions/waypoints/2

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with different store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]  
 Drivers with eids 2, 3 work for store\_id 1,  
 unassigned orders are for store\_id 1,  
 Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": [3,2,1]
}
```

Response Code: 401

Response Body:

```
Employee does not work at the store with id 2.
```

Developed and tested by Damon Lin

Delivery6e: Missing required properties  
Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,

unassigned orders are for store\_id 1,

Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
}
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing required properties",
 "missingProperties": [
 "orders"
]
}
```

Developed and tested by Damon Lin

Delivery6f: Invalid properties - orders is not an array  
 Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,  
 unassigned orders are for store\_id 1,  
 Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": "hello"
}
```

Response Code: 400

Response Body:

```
non-empty orders array required
```

Developed and tested by Damon Lin

Delivery6g: Invalid properties - orders array is empty  
 Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,  
 unassigned orders are for store\_id 1,  
 Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": []
}
```

Response Code: 400

Response Body:

```
non-empty orders array required
```

Developed and tested by Damon Lin

Delivery6h: Invalid properties - order ID is not an integer  
 Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,  
 unassigned orders are for store\_id 1,  
 Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": ["hello"]
}
```

Response Code: 400

Response Body:

```
Integer required for order_id
```

Developed and tested by Damon Lin

Delivery6i: Invalid properties - order ID does not exist/invalid  
 Resource URI: GET/directions/waypoints/1

Request Headers: Valid Auth Token Cookie, signed in as employee that works at store with same store\_id

Database Status: Available drivers ids: [3], Unassigned order ids: [4, 5, 6]

Drivers with eids 2, 3 work for store\_id 1,  
 unassigned orders are for store\_id 1,  
 Driver with eid 2 has orders [3, 2, 1]

Request Body:

```
{
 "orders": [10]
}
```

Response Code: 400

Response Body:

```
Invalid order id.
```

Developed and tested by Damon Lin



## Delivery7: Update Driver Location

SQL Queries Used:

```
// Updates the position of the driver's current delivery batch
let updateRes = await runQuery(employeePool,
 `update delivery_batch set current_latlng = POINT(?, ?) where
assignedToEmp = ${jwtBody.id}`,
 [decodedData.latlng.lat, decodedData.latlng.lng]);
```

Delivery7a: Success

Resource URI: POST /geolocation/updatePos

Request Headers: Valid Auth Token Cookie, signed in as driver with an assigned delivery batch

Database Status: Currently signed-in driver is assigned a delivery batch

Request Body:

```
{
 "latlng": {
 "lat": 40.522770621227224,
 "lng": -74.45704341744475
 }
}
```

Response Code: 200

Response Body:

**Successfully updated driver location.**

Developed and tested by Damon Lin

Delivery7b: Not signed in

Resource URI: POST /geolocation/updatePos

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

**You are not signed in.**

Developed and tested by Damon Lin

Delivery7c: Missing required properties  
Resource URI: POST /geolocation/updatePos

Request Headers: Valid Auth Token Cookie, signed in as driver with an assigned delivery batch

Database Status: Currently signed-in driver is assigned a delivery batch

Request Body:

```
{
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing required properties",
 "missingProperties": [
 "latlng"
]
}
```

Developed and tested by Damon Lin

Delivery7d: Invalid latlng  
Resource URI: POST /geolocation/updatePos

Request Headers: Valid Auth Token Cookie, signed in as driver with an assigned delivery batch

Database Status: Currently signed-in driver is assigned a delivery batch

Request Body:

```
{
 "latlng": 40.522770621227224
}
```

Response Code: 400

Response Body:

```
{
 "error": "Invalid latlng."
}
```

Developed and tested by Damon Lin

Delivery7e: Invalid latlng

Resource URI: POST /geolocation/updatePos

Request Headers: Valid Auth Token Cookie, signed in as driver with an assigned delivery batch

Database Status: Currently signed-in driver is assigned a delivery batch

Request Body:

```
{
 "latlng": 40.522770621227224
}
```

Response Code: 400

Response Body:

```
{
 "error": "Invalid latlng."
}
```

Developed and tested by Damon Lin

Delivery7f: Not assigned a delivery batch  
Resource URI: POST /geolocation/updatePos

Request Headers: Valid Auth Token Cookie, signed in as driver with an assigned delivery batch

Database Status: Currently signed-in driver is **not** assigned a delivery batch

Request Body:

```
{
 "latlng": {
 "lat": 40.522770621227224,
 "lng": -74.45704341744475
 }
}
```

Response Code: 404

Response Body:

```
{
 "error": "You are not assigned any delivery batch."
}
```

Developed and tested by Damon Lin

## Delivery8: See Driver's Location

SQL Queries Used:

```
// Get the position of the driver for the most recent order that is
In-Transit
let queryRes = await runQuery(customerPool,
 `select current_latlng
 from customer_order c
 join in_batch i on i.order_id = c.order_id
 join delivery_batch d on d.batch_id = i.batch_id
 where c.ordered_by = ${jwtBody.id}
 order by DT_created desc
 limit 1`);
```

Delivery8a: Success

Resource URI: GET /geolocation/fetchOrderPosCustomer

Request Headers: Valid Auth Token Cookie, signed in as customer

Database Status: Currently signed-in customer has a delivery that is In-Transit

Request Body: None

Response Code: 200

Response Body:

```
{
 "current_latlng": {
 "x": 40.523421858838276,
 "y": -74.45823918823967
 }
}
```

Developed and tested by Damon Lin

Delivery8b: Not signed in

Resource URI: GET /geolocation/fetchOrderPosCustomer

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Damon Lin

Delivery8c: No In-Transit Order

Resource URI: GET /geolocation/fetchOrderPosCustomer

Request Headers: Valid Auth Token Cookie, signed in as customer

Database Status: Currently signed-in customer does not have a delivery in transit

Request Body: None

Response Code: 404

Response Body:

```
{
 "error": "You do not have any In-Transit orders."
}
```

Developed and tested by Damon Lin

## Delivery9: See My Delivery Destination (Customer)

SQL Queries Used:

```
// Get the position of the most recent order that is In-Transit
let queryRes = await runQuery(customerPool,
 `select c.delivery_latlng
 from customer_order c
 where c.ordered_by = ${jwtBody.id}
 and c.status = 'In-Transit'
 order by DT_created desc
 limit 1
`);
```

Delivery9a: Success

Resource URI: GET /geolocation/fetchCustomerPos

Request Headers: Valid Auth Token Cookie, signed in as customer

Database Status: Currently signed-in customer has a delivery that is In-Transit

Request Body: None

Response Code: 200

Response Body:

```
{
 "delivery_latlng": {
 "x": 40.41829236146375,
 "y": -74.70639548636213
 }
}
```

Developed and tested by Damon Lin

Delivery9b: Not signed in

Resource URI: GET /geolocation/fetchCustomerPos

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

```
You are not signed in.
```

Developed and tested by Damon Lin

Delivery9c: No In-Transit Order

Resource URI: GET /geolocation/fetchCustomerPos

Request Headers: Valid Auth Token Cookie, signed in as customer

Database Status: Currently signed-in customer does not have a delivery in transit

Request Body: None

Response Code: 404

Response Body:

```
{
 "error": "You do not have any In-Transit orders."
}
```

Developed and tested by Damon Lin

## Delivery10: Get Delivery Time Estimate

SQL Queries Used:

```
// query to select my order
let queryRes = await runQuery(customerPool,
 `select c.order_id
 from customer_order c
 where c.ordered_by = ${jwtBody.id}
 order by c.DT_created desc
 limit 1;`)

// gets the order ids of all orders in the same delivery batch
let queryRes2 = await runQuery(customerPool,
 `select c.order_id, c.delivery_latlng, i.order_index
 from customer_order c
 join in_batch i on c.order_id = i.order_id
 where c.status = 'In-Transit'
 and batch_id = (select i.batch_id from in_batch i where i.order_id =
?)
 ORDER BY i.order_index asc; `, myOrder);

// gets order of the current customer
let queryRes3 = await runQuery(customerPool,
 `select current_latlng
 from customer_order c
 join in_batch i on i.order_id = c.order_id
 join delivery_batch d on d.batch_id = i.batch_id
 where c.ordered_by = ${jwtBody.id} `);
```

Delivery10a: Success

Resource URI: GET /timeEst/getOrderTimeEst

Request Headers: Valid Auth Token Cookie, signed in as customer

Database Status: Currently signed-in customer has a delivery that is In-Transit

Request Body: None

Response Code: 200

Response Body:

```
{
 "timeEstimate": 4709
}
```

Developed and tested by Damon Lin

Delivery10b: Not signed in  
Resource URI: GET /timeEst/getOrderTimeEst

Request Headers: None

Request Body: None

Response Code: 401

Response Body:

You are not signed in.

Developed and tested by Damon Lin

Delivery10c: No In-Transit Order  
Resource URI: GET /timeEst/getOrderTimeEst

Request Headers: Valid Auth Token Cookie, signed in as customer

Database Status: Currently signed-in customer does not have a delivery in transit

Request Body: None

Response Code: 404

Response Body:

```
{
 "error": "You do not have any In-Transit orders."
}
```

Developed and tested by Damon Lin

## Chatbot1: Ask the Chatbot a Question

SQL Queries Used:

```
None
```

Chatbot1a: Success

Resource URI: POST /chatbot

Request Headers: None

Request Body:

```
{
 "question": "Where are the stores?"
}
```

Response Code: 200

Response Body:

```
{
 "success": true,
 "answer": "Visitors can view a map of store locations on the website
to find the nearest store and get detailed store information, including
store hours and location. While the availability of menu items may vary by
location, the document does not specify restrictions on item
availability."
}
```

Developed and tested by Brandon Cheng

Chatbot1b: Missing required properties  
Resource URI: POST /chatbot

Request Headers: None

Request Body:

```
{
}
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing required properties",
 "missingProperties": [
 "question"
]
}
```

Developed and tested by Brandon Cheng

Chatbot1c: Unclassified Question  
Resource URI: POST /chatbot

Request Headers: None

Request Body:

```
{
 "question": "eturdytfiuygoiuh;ilknlu"
}
```

Response Code: 200

Response Body:

```
{
 "success": false,
 "answer": "Sorry, I can't help you with that. \nFor further
assistance, please submit a help ticket."
}
```

Developed and tested by Brandon Cheng

Chatbot1d: Question about the menu categories

Resource URI: POST /chatbot

Request Headers: None

Request Body:

```
{
 "question": "What are the menu categories?"
}
```

Response Code: 200

Response Body:

```
{
 "success": true,
 "answer": "These are the categories on our menu: \n - Breads\n - Oven-baked Dips & Twists combos\n - Oven-Baked Dips\n - Chicken\n - Desserts\n - Drinks\n - Dipping Cups\n - Salad Dressing\n - Loaded Tots\n - Pasta\n - Salads\n - Sandwiches\n - Specialty Pizzas"
}
```

Developed and tested by Brandon Cheng

Chatbot1e: Question about a specific category

Resource URI: POST /chatbot

Request Headers: None

Request Body:

```
{
 "question": "Tell me about the specialty pizzas"
}
```

Response Code: 200

Response Body:

```
{
 "success": true,
 "answer": "These are the Specialty Pizzas we offer: \n - ExtravaganZZa\n - MeatZZa\n - Philly Cheese Steak\n - Pacific Veggie\n - Honolulu Hawaiian\n - Deluxe\n - Cali Chicken Bacon Ranch\n - Buffalo Chicken\n - Ultimate Pepperoni\n - Memphis BBQ Chicken\n - Wisconsin 6 Cheese\n - Spinach & Feta"
}
```

Developed and tested by Brandon Cheng

Chatbot1f: Question about a specific item

Resource URI: POST /chatbot

Request Headers: None

Request Body:

```
{
 "question": "What are the toppings for the MeatZZa"
}
```

Response Code: 200

Response Body:

```
{
 "success": true,
 "answer": "Here are the customization options for MeatZZa:
\n\ncheese\n - Extra\n - Light\n - None\n - Normal\n\nrust\n - Crunchy
Thin Crust\n - Gluten Free Crust\n - Hand Tossed\n - Handmade Pan\n - New
York Style\n\nmeats\n - Bacon\n - No Beef\n - No Ham\n - No Italian
Sausage\n - No Pepperoni\n - Philly Steak\n - Premium Chicken\n -
Salami\n\nnonMeats\n - Banana Peppers\n - Black Olives\n - Cheddar Cheese
Blend\n - Diced Tomatoes\n - Feta Cheese\n - Green Peppers\n - Hot Buffalo
Sauce\n - Jalapeno Peppers\n - Mushrooms\n - No Shredded Provolone
Cheese\n - Onions\n - Pineapple\n - Shredded Parmesan Asiago\n -
Spinach\n\nsauce\n - Alfredo Sauce\n - Extra\n - Garlic Parmesan Sauce\n -
Hearty Marinara Sauce\n - Honey BBQ Sauce\n - Light\n - No Sauce\n -
Normal\n - Ranch\n - Robust Inspired Tomato Sauce\n\nsize\n - 10\" Small\n - 12\" Medium\n - 14\" Large\n - 16\" X-Large"
}
```

Developed and tested by Brandon Cheng

## Pizza Pipeline

### O1: Order Payment

O4c: Successfully added order to database  
Resource URI: <https://127.0.0.1:8080/order/postOrder>  
Request Body:

```
{
 "orderData": [
 {
 "made_at": 1,
 "credit_card": "1234567890123456",
 "status": "Processing",
 "total_price": 9.99,
 "delivery_address": "123 Main St, City, Country",
 "DT_created": "2024-03-25 10:00:00",
 "DT_delivered": null,
 "ordered_by": 1
 }
],
 "menuItemData": [
 {
 "price": 9.99,
 "image_url": "https://example.com/image1.jpg",
 "description": "Pizza Margherita"
 },
 {
 "price": 9.99,
 "image_url": "https://example.com/image2.jpg",
 "description": "Cheese Pizza"
 }
],
 "orderItemData": [
 {"order_id": 0, "mid": 1},
 {"order_id": 0, "mid": 2},
]}
```

```
{
 "order_id": 0, "mid": 2}
]
}
```

Response Code: 200

Response Body:

```
Order confirmed, inserted into database
```

O4d: Error parsing the request body

Resource URI: <https://127.0.0.1:8080/order/postOrder>

Request Body:

```
{
 "iuwfheefiuuh" : 3
}
```

Response Code: 400

Response Body:

```
Error reading JSON
```

Created and tested by Arya Shetty and Vineal Sunkara

## O2: Apply Rewards Points

Resource URI: <https://127.0.0.1:8080/order/redeemRewards> POST

SQL Queries Used:

```
let query = "UPDATE customer_account SET rewards_points = rewards_points - 5
WHERE cid = " + cusID;
```

a: Customer is successfully able to redeem their rewards points made from purchases

Resource URI: /order/redeemRewards

Request Headers:

```
POST,
(customer) Cookie:
```

```
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULApD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have an orders placed in the database to have available rewards points to use

Request Body:

```
{
 "redeem": true
}
```

Response Code: **200**

Response Body:

"Successfully Redeemed!"

b: Customer is unable to redeem reward points

Resource URI: /order/redeemRewards

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULApD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer does not have orders in the database which results in no points

Request Body:

```
{
 "redeem": true
}
```

Response Code: **404**

Response Body:

```
"Internal Server Error: Error redeeming points for customer john_doe ."
```

c: Customer is unable to redeem reward points due to invalid credentials

Resource URI: /order/redeemRewards

Request Headers:

```
NONE
```

Database Status:

```
Customer does not have orders in the database since they are not signed in
```

Request Body:

```
{
 "redeem": true
}
```

Response Code: **404**

Response Body:

```
"Invalid Credentials"
```

Resource URI: <https://127.0.0.1:8080/order/getRewards> GET

SQL Queries Used:

```
let rewardQuery = "SELECT rewards_points FROM customer_account WHERE cid = "
+ cusID;
```



a: Customer is able to get rewards points from purchases they make

Resource URI: /order/getRewards

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must place an order in the database in order to receive the rewards points

Request Body:

NONE

Response Code: **200**

Response Body:

```
[
 {
 "rewards_points": -8
 }
]
```

b: Customer is unable to get rewards points from purchases since they didnt make them

Resource URI: /order/getRewards

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer doesnt have anything in the database

Request Body:

NONE

Request Body:

NONE

Response Code: **404**

Response Body:

"Not Found: No rewards found for this customer."

d: Customer is unable to get rewards points from purchases they make due to internal server error

Resource URI: /order/getRewards

Request Headers:

GET,  
(customer) Cookie:  
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV  
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1  
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure

Database Status:

Customer must place an order in the database in order to receive the rewards points

Request Body:

NONE

Request Body:

NONE

Response Code: **500**

Response Body:

"Internal Server Error: Failed to retrieve reward points for customer jone\_doe."

### O3: Input Valid Address

Resource URI: <https://127.0.0.1:8080/order/checkOption> GET

SQL Queries Used:

```
let query = "SELECT delivery_address FROM customer_order WHERE ordered_by =
" + cusID + " ORDER BY order_id DESC LIMIT 1";
```

a: Customer is able to check the address that their order is getting delivered to

Resource URI: /order/checkOption

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULApD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer must have an order placed that is in the database to check the
address
```

Request Body:

```
NONE
```

Response Code: **201**

Response Body:

```
[
 {
 "delivery_address": "9 Pension Hill Road, Manalapan Township, NJ
07726-3895, USA"
 }
```

]

b: Customer is unable to check the address that their order is getting delivered due to order not existing

Resource URI: /order/checkOption

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer does not have any order in the database

Request Body:

NONE

Response Code: **404**

Response Body:

"Not Found: No delivery details found for this customer."

C: Customer is unable to check the address that their order is getting delivered due to internal server error

Resource URI: /order/checkOption

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer has an order in the database

Request Body:

NONE

Response Code: **500**

Response Body:

```
"Internal Server Error: Failed to check delivery status for customer
john_doe.
```

## O4: Stripe Payment

Resource URI: <https://127.0.0.1:8080/order/createCheckoutSession>

See [Interaction with Stripe API for O4](#)

## O5: Refund Order

/order/handleRefund

SQL Queries Used:

```
const result = await runQuery(customerPool, 'SELECT stripe_checkout_id,
stripe_payment_intent_id FROM customer_order WHERE order_id = ?',
[orderId]);
const updateQuery = 'UPDATE customer_order SET status = "Refunded" WHERE
order_id = ?';
let query = "UPDATE customer_account SET rewards_points = rewards_points - 1
WHERE cid = " + customerID;
```

O5a: Successful refund to customer with Stripe (valid input)

Resource URI: /order/handleRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have an order in the database in order to refund, they must also comply to the business policies to refund.

Request Body:

```
{
 "orderId": "459"
}
```

Response Code: **200**

Response Body:

```
{
 "status": "success",
 "refundId": "re_3PCqLCP5gIWmEZ1P1jIlfWJH"
```

```
}
```

O5b: Unsuccessful refund due to order not being found

Resource URI: /order/handleRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULApD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer does not have an order in the database
```

Request Body:

```
{
 "orderId": "500"
}
```

Response Code: **404**

Response Body:

```
{
 "status": "error",
 "message": "Order not found."
}
```

O5c: Unsuccessful refund due to invalid JSON

Resource URI: /order/handleRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJOVwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer has order in database
```

Request Body:

```
{}
```

Response Code: **400**

Response Body:

```
{
 "status": "error",
 "message": "Invalid JSON format in request body."
}
```

O5d: Unsuccessful refund due to invalid credentials

Resource URI: /order/handleRefund

Request Headers:

```
NONE
```

Database Status:

```
Customer is not signed in and has nothing in their cart
```

Request Body:

```
{
 "orderID": 51
}
```

Response Code: **500**

Response Body:

```
{
 Invalid Credentials
}
```

/order/emailOrderRefund

SQL Queries Used:

```
let recentOrderQuery = "SELECT order_id FROM customer_order WHERE
ordered_by = " + cusID + " ORDER BY order_id DESC LIMIT 1";
let emailQuery = "SELECT email FROM customer_account WHERE cid = " + cusID;
const query = `
 SELECT co.order_id,
 co.status,
 co.DT_created AS date_created,
 co.total_price,
 co.delivery_address,
 co.stripe_checkout_id,
 oi.item_num,
 mi.price AS item_price,
 mi.item_name AS item_description
 FROM customer_order AS co
 LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
 LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
 WHERE co.order_id = ?
`;
```

a: Customer successfully receives email confirmation that order is refunded

Resource URI: /order/emailOrderRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have placed an order in the database and must comply with business policies in order to refund their order

Request Body:

```
{
 "orderId": 2
}
```

Response Code: **200**

Response Body:

```
Email sent successfully!
```

b: Customer fails to receive email confirmation that order is refunded

Resource URI: /order/emailOrderRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer did not place an order in the database

Request Body:

```
{
 "orderId": 500
}
```

```
}
```

Response Code: **404**

Response Body:

```
"Order not found"
```

c: Customer fails to receive email confirmation that order is refunded due to invalid JSON

Resource URI: /order/emailOrderRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJOVwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer must have placed an order in the database and must comply with
business policies in order to refund their order
```

Request Body:

```
{
 "orderId": 2
}
```

Response Code: **400**

Response Body:

```
"Invalid JSON"
```

d: Customer fails to receive email confirmation that order is refunded due to internal server error

Resource URI: /order/emailOrderRefund

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJOVwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have placed an order in the database and must comply with business policies in order to refund their order

Request Body:

```
{
 "orderId": 2
}
```

Response Code: **500**

Response Body:

```
"Internal server error when fetching email"
```

Request Body:

```
{
 "orderId": 2
}
```

Response Code: **500**

Response Body:

```
"Internal server error when fetching order"
```

Response Body:

```
"Order not found"
```

Request Body:

```
{
 "orderId": 2
}
```

Response Code: **500**

Response Body:

```
"Failed to retrieve order details"
```

## O6: View All Customer Orders

/order/getCustomerOrder

SQL Queries Used:

```
let storeQuery = "SELECT works_at FROM employee_account WHERE eid = " +
employeeID;
query = `SELECT co.order_id, co.status, co.DT_created AS date_created,
co.total_price,
oi.item_num, mi.price AS item_price, mi.item_name AS
item_description, co.made_at
FROM customer_order AS co
LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
WHERE co.made_at = ${storeID} AND co.status != 'Processing'`;
query = `SELECT co.order_id, co.status, co.DT_created AS date_created,
co.total_price,
oi.item_num, mi.price AS item_price, mi.item_name AS
item_description, co.made_at
FROM customer_order AS co
LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
WHERE co.ordered_by = ${cusID}`;
```

O6a: Customer can successfully view their order

Resource URI: <https://127.0.0.1:8080/order/getCustomerOrder>

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJOVwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer needs to have their order in the database in order to view their

```
custom order history
```

Request Body:

```
None
```

Response Code: **200**

Response Body:

```
[
 {
 "order_id": 457,
 "status": "Refunded",
 "date_created": "2024-05-04T21:35:18.000Z",
 "total_price": 25.97,
 "item_num": 921,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
 },
 {
 "order_id": 457,
 "status": "Refunded",
 "date_created": "2024-05-04T21:35:18.000Z",
 "total_price": 25.97,
 "item_num": 922,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
 },
 {
 "order_id": 457,
 "status": "Refunded",
 "date_created": "2024-05-04T21:35:18.000Z",
 "total_price": 25.97,
```

```
 "item_num": 923,
 "item_price": 7.99,
 "item_description": "Garlic Bread Twists",
 "made_at": 1
},
{
 "order_id": 458,
 "status": "Refunded",
 "date_created": "2024-05-04T21:38:34.000Z",
 "total_price": 25.97,
 "item_num": 924,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
},
{
 "order_id": 458,
 "status": "Refunded",
 "date_created": "2024-05-04T21:38:34.000Z",
 "total_price": 25.97,
 "item_num": 925,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
},
{
 "order_id": 458,
 "status": "Refunded",
 "date_created": "2024-05-04T21:38:34.000Z",
 "total_price": 25.97,
 "item_num": 926,
 "item_price": 7.99,
 "item_description": "Garlic Bread Twists",
 "made_at": 1
}
```

```
 },
{
 "order_id": 459,
 "status": "Refunded",
 "date_created": "2024-05-04T21:39:49.000Z",
 "total_price": 25.97,
 "item_num": 927,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
},
{
 "order_id": 459,
 "status": "Refunded",
 "date_created": "2024-05-04T21:39:49.000Z",
 "total_price": 25.97,
 "item_num": 928,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
},
{
 "order_id": 459,
 "status": "Refunded",
 "date_created": "2024-05-04T21:39:49.000Z",
 "total_price": 25.97,
 "item_num": 929,
 "item_price": 7.99,
 "item_description": "Garlic Bread Twists",
 "made_at": 1
}
]
```

O6b: Customer is unsuccessful at viewing their order

Resource URI: https://127.0.0.1:8080/order/getCustomerOrder

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer does not have an order in the database

Request Body:

NONE

Response Code: **404**

Response Body:

"Not Found: No order found."

O6c: Customer is unsuccessful at viewing their order

Resource URI: https://127.0.0.1:8080/order/getCustomerOrder

Request Headers:

```
GET,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer has order in the database

Request Body:

NONE

Response Code: **500**

Response Body:

"Internal Server Error: Failed to retrieve past orders."

O6d: Customer is unsuccessful at viewing their order since they are not signed in  
Resource URI: <https://127.0.0.1:8080/order/getCustomerOrder>  
Request Headers:

NONE

Database Status:

Customer does not have anything in the database since they're not signed in

Request Body:

NONE

Response Code: **401**

Response Body:

"Invalid Credentials"

Created and tested by Nikash Rajeshbabu

Resource URI: <https://127.0.0.1:8080/order/createReview> POST

SQL Queries Used:

```
const insertQuery = `UPDATE order_item SET reviewDescription = `` +
description + `` WHERE item_num = `` + item_num;
```



a: Customer is successfully able to create a review for an item that they have bought

Resource URI: /order/createReview

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have purchased that item that they are writing a review for,  
it must be in the database

Request Body:

```
{
 "item_num": "1",
 "description": "Great pizza!"
}
```

Response Code: **201**

Response Body:

Review created successfully.

b: Customer is unable to create a review for an item that they have bought due to JSON format

Resource URI: /order/createReview

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have purchased that item that they are writing a review for,  
the database must reflect this purchase

Request Body:

```
{
 "item_num": "1",
```

Response Code: **400**

Response Body:

“Bad Request: JSON data is missing”

c: Customer is unable to create a review for an item due to invalid credentials

Resource URI: /order/createReview

Request Headers:

NONE

Database Status:

The customer is not signed in, nothing is in the database to write a review for

Request Body:

```
{
 "item_num": "1"
}
```

Response Code: **401**

Response Body:

```
{
 "Invalid Credentials"
}
```

Resource URI: <https://127.0.0.1:8080/order/getReview> POST

SQL Queries Used:

```
const insertQuery = `SELECT reviewDescription FROM order_item WHERE item_num
= ` + item_num;
```

a: Customer is successfully able to retrieve a review they have written for an item

Resource URI: /order/getReview

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have purchased the item and written a review for the item they are getting their review for

Request Body:

```
{
 "item_num": 921
}
```

Response Code: **201**

Response Body:

“I really liked it!”

b: Customer is not able to retrieve a review they have written for an item due to invalid json

Resource URI: /order/getReview

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have purchased the item and written a review for the item they are getting their review for

Request Body:

```
{}
```

```
"item_num": 921
```

Response Code: **400**

Response Body:

```
"Bad Request: JSON data is missing"
```

c: Customer is unable to retrieve a review they have written for an item due to an internal server error

Resource URI: /order/getReview

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer must have purchased the item and written a review for the item
they are getting their review for
```

Request Body:

```
{
 "item_num": 1234
}
```

Response Code: **500**

Response Body:

```
"Internal Server error."
```

d: Customer is unable to retrieve a review they have written for an item due to invalid credentials

Resource URI: /order/getReview

Request Headers:

```
NONE
```

Database Status:

Customer is not signed in so there is nothing in the database

Request Body:

```
{
 "item_num": 1234
}
```

Response Code: **401**

Response Body:

"Invalid Credentials"

## O7: View All Store Orders

/order/setStatus

SQL Queries Used:

```
const query = 'UPDATE customer_order SET status = ? WHERE order_id = ?';
const result = await runQuery(adminPool, query, [newStatus, order_id]);
```

O3a: Request body is empty

Resource URI: <https://127.0.0.1:8080/order/setStatus>

Request Body:

```
{ }
```

Response Code: 400

Response Body:

```
{
 "error": "Missing order_id or newStatus in request"
}
```

O3b: order\_id and status sent

Resource URI: <https://127.0.0.1:8080/order/setStatus>

Request Body:

```
{
 "order_id" : 9,
 "newStatus" : "Processing"
}
```

Response Code: 200

Response Body:

```
{
 "success": true,
 "message": "Order status updated successfully"
}
```

O3c: Only sent order\_id

Resource URI: <https://127.0.0.1:8080/order/setStatus>

Request Body:

```
{
 "order_id" : 9
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing order_id or newStatus in request"
}
```

O3d: Only sent newStatus

Resource URI: <https://127.0.0.1:8080/order/setStatus>

Request Body:

```
{
 "newStatus" : "Processing"
}
```

Response Code: 400

Response Body:

```
{
 "error": "Missing order_id or newStatus in request"
}
```

Developed and tested by Arya Shetty

/order/getPastOrder

SQL Queries Used:

```
let storeQuery = "SELECT works_at FROM employee_account WHERE eid = " +
employeeID;
query = `SELECT co.order_id, co.status, co.DT_created AS date_created,
co.total_price,
oi.item_num, mi.price AS item_price, mi.item_name AS
item_description, co.made_at
FROM customer_order AS co
LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
WHERE co.made_at = ${storeID} AND co.status != 'Processing'`;
query = `SELECT co.order_id, co.status, co.DT_created AS date_created,
co.total_price,
oi.item_num, mi.price AS item_price, mi.item_name AS
item_description, co.made_at
FROM customer_order AS co
LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
WHERE co.ordered_by = ${cusID}`;
```

a: Employee successfully retrieved past order

Resource URI: /order/getPastOrder

Request Headers:

```
GET
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImVtcGxveWV
lIiwiZWQiOjIsImlhCI6MTcxNDk2NDY1N30.FP7UtLmJpPUug0hsAGclev9DYN3eJcc0X7AVq
JbnodU; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
The orders must be in the database for the employee to retrieve them
```

Request Body:

```
NONE
```

Response Code: **201**

Response Body:

```
[
 {
 "order_id": 457,
 "status": "Refunded",
 "date_created": "2024-05-04T21:35:18.000Z",
 "total_price": 25.97,
 "item_num": 921,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
 },
 {
 "order_id": 457,
 "status": "Refunded",
 "date_created": "2024-05-04T21:35:18.000Z",
 "total_price": 25.97,
 "item_num": 922,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 }
]
```

```
 "made_at": 1
 },
 {
 "order_id": 457,
 "status": "Refunded",
 "date_created": "2024-05-04T21:35:18.000Z",
 "total_price": 25.97,
 "item_num": 923,
 "item_price": 7.99,
 "item_description": "Garlic Bread Twists",
 "made_at": 1
 },
 {
 "order_id": 458,
 "status": "Refunded",
 "date_created": "2024-05-04T21:38:34.000Z",
 "total_price": 25.97,
 "item_num": 924,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
 },
 {
 "order_id": 458,
 "status": "Refunded",
 "date_created": "2024-05-04T21:38:34.000Z",
 "total_price": 25.97,
 "item_num": 925,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
 },
 {
 "order_id": 458,
```

```
 "status": "Refunded",
 "date_created": "2024-05-04T21:38:34.000Z",
 "total_price": 25.97,
 "item_num": 926,
 "item_price": 7.99,
 "item_description": "Garlic Bread Twists",
 "made_at": 1
},
{
 "order_id": 459,
 "status": "Refunded",
 "date_created": "2024-05-04T21:39:49.000Z",
 "total_price": 25.97,
 "item_num": 927,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
},
{
 "order_id": 459,
 "status": "Refunded",
 "date_created": "2024-05-04T21:39:49.000Z",
 "total_price": 25.97,
 "item_num": 928,
 "item_price": 8.99,
 "item_description": "Stuffed Cheesy Bread",
 "made_at": 1
},
{
 "order_id": 459,
 "status": "Refunded",
 "date_created": "2024-05-04T21:39:49.000Z",
 "total_price": 25.97,
 "item_num": 929,
```

```

 "item_price": 7.99,
 "item_description": "Garlic Bread Twists",
 "made_at": 1
}
]
```

b: Employee failed to retrieve past order due to not being assigned to a store

Resource URI: /order/getPastOrder

Request Headers:

```

GET
NONE
```

Database Status:

```
The orders are in the database
```

Request Body:

```
NONE
```

Response Code: **404**

Response Body:

```
"Not Found: Employee does not have an assigned store."
```

c: Employee failed to retrieve past order due to item not existing

Resource URI: /order/getPastOrder

Request Headers:

```

GET
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImVtcGxveWV
lIiwiaWQiOjIsImlhCI6MTcxNDY1N30.FP7UtLmJpPUug0hsAGclev9DYN3eJcc0X7AVq
JbnodU; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
The orders are not in the database
```

Request Body:

NONE

Response Code: **404**

Response Body:

"Not Found: No orders found."

d: Employee failed to retrieve past order due internal server error

Resource URI: /order/getPastOrder

Request Headers:

GET  
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImVtcGxveWV  
lIiwiaWQiOjIsImlhCI6MTcxNDk2NDY1N30.FP7UtLmJpPUug0hsAGclev9DYN3eJcc0X7AVq  
JbnodU; Max-Age=86400; Path=/; HttpOnly; Secure

Database Status:

The orders are in the database

Request Body:

NONE

Response Code: **500**

Response Body:

"Internal Server Error: Failed to retrieve past orders."

## O8: Cancel Order

/order/cancelOrder

SQL Queries Used:

```
For cancelorder:

const updateQuery = 'UPDATE customer_order SET status = "Canceled" WHERE

order_id = ?';
```

```
results = await runQuery(adminPool, updateQuery, [order_id]);
```

O7a: Given order id in database

Resource URI: <https://127.0.0.1:8080/order/cancelOrder>

Request Body:

```
{
 "order_id":2
}
```

Response Code: **200**

Response Body:

```
Canceled order number 2
```

Created and tested by Arya Shetty

Resource URI: <https://127.0.0.1:8080/order/emailOrderCancel> POST

SQL Queries Used:

```
let recentOrderQuery = "SELECT order_id FROM customer_order WHERE ordered_by = " + cusID + " ORDER BY order_id DESC LIMIT 1";
let emailQuery = "SELECT email FROM customer_account WHERE cid = " + cusID;
const query = `
 SELECT co.order_id,
 co.status,
 co.DT_created AS date_created,
 co.total_price,
 co.delivery_address,
 oi.item_num,
 mi.price AS item_price,
 mi.item_name AS item_description
 FROM customer_order AS co
 LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
 LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
 WHERE co.order_id = ?`
```

a: Customer successfully receives email confirmation that order is canceled

Resource URI: /order/emailOrderCancel

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have placed an order in the database and must comply with business policies in order to cancel their order

Request Body:

```
{
 "orderId": 1
}
```

Response Code: **200**

Response Body:

```
Email sent successfully!
```

b: Customer fails to receive email confirmation that order is canceled due to order not existing

Resource URI: /order/emailOrderCancel

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer did not place an order in the database

Request Body:

```
{
 "orderId": 500
}
```

```
}
```

Response Code: **404**

Response Body:

```
"Order not found"
```

c: Customer fails to receive email cancel confirmation due to invalid JSON

Resource URI: /order/emailOrderCancel

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJOVwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer must have placed an order and must comply with business policies
in order to cancel their order
```

Request Body:

```
{
 "orderId": 1
```

Response Code: **400**

Response Body:

```
"Invalid JSON"
```

d: Customer fails receives email confirmation that order is canceled due to internal server error

Resource URI: /order/emailOrderCancel

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJOVwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

Customer must have placed an order in database and must comply with business policies in order to cancel their order

Request Body:

```
{
 "orderId": 1
}
```

Response Code: **500**

Response Body:

```
"Internal server error when fetching email"
```

Request Body:

```
{
 "orderId": 1
}
```

Response Code: **500**

Response Body:

```
"Internal server error when fetching order"
```

Response Body:

```
"Order not found"
```

Request Body:

```
{
 "orderId": 1
}
```

Response Code: **500**

Response Body:

```
"Failed to retrieve order details"
```

## O9: View Order Status

/order/checkStatus

SQL Queries Used:

```

For checkstatus:
const query = `SELECT status FROM customer_order WHERE order_id = ?`;

results = await runQuery(customerPool, query, [order_id]);

For getoid:
const query = 'SELECT order_id FROM customer_order WHERE ordered_by = ?';

results = await runQuery(customerPool, query, [customer_id]);

For getorder:
const query = `
 SELECT co.order_id,
 co.status,
 co.DT_created AS date_created,
 co.total_price,
 oi.item_num,
 mi.price AS item_price,
 mi.description AS item_description
 FROM customer_order AS co
 LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
 LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
 WHERE co.order_id = ?
 `;

results = await runQuery(customerPool, query, [order_id]);

For pastorder:
let query = "SELECT co.order_id, " +
 " co.status, " +
 " co.DT_created AS date_created, " +
 " co.total_price, " +
 " oi.item_num, " +
 " mi.price AS item_price, " +
 " mi.description AS item_description " +
 "FROM customer_order AS co " +
 "LEFT JOIN order_item AS oi ON co.order_id = oi.order_id " +
 "LEFT JOIN menu_item AS mi ON oi.mid = mi.mid";

let results = await runQuery(adminPool, query);

```



O9a: No order id submitted

Resource URI: <https://127.0.0.1:8080/order/checkStatus>

Request Body:

```
{ }
```

Response Code: 200

Response Body:

```
{
 "No order_id provided or invalid".
}
```

O9b: Order status of id 9 requested

Resource URI: <https://127.0.0.1:8080/order/checkStatus>

Request Body:

```
{
 "order_id" : 9
}
```

Response Code: 200

Response Body:

```
[
 {
 "status": "Processing"
 }
]
```

O9c: Order status of id 10 requested

Resource URI: <https://127.0.0.1:8080/order/checkStatus>

Request Body:

```
{
 "order_id" : 10
}
```

Response Code: 200

Response Body:

```
[
 {
 "status": "Processing"
 }
]
```

Created and tested by Arya Shetty

O9d: Checked the wrong customer id not in the database  
Resource URI: <https://127.0.0.1:8080/order/getOID>

Request Body:

```
{
 "customer_id" : 3
}
```

Response Code: 200

Response Body:

```
[]
```

O8e: Successfully retrieved orders from customer id in database  
Resource URI: <https://127.0.0.1:8080/order/getOID>

Request Body:

```
{
 "customer_id" : 1
}
```

Response Code: 200

Response Body:

```
[
 {
 "order_id": 9
 },
 {
 "order_id": 10
 },
 {
 "order_id": 11
 },
 {
 "order_id": 12
 },
 {
 "order_id": 13
 },
 {
 "order_id": 14
 }]
```

/order/getOrder

O9f: Successfully requested order id 9  
 Resource URI: <https://127.0.0.1:8080/order/getOrder>

Request Body:

```
{
 "order_id": 9
}
```

Response Code: **200**

Response Body:

```
[
 {
 "order_id": 9,
 "status": "Processing",
 "date_created": "2024-03-25T14:00:00.000Z",
 "total_price": 9.99,
 "item_num": 1,
 "item_price": 9.99,
 "item_description": "Pizza Margherita"
 },
 {
 "order_id": 9,
 "status": "Processing",
 "date_created": "2024-03-25T14:00:00.000Z",
 "total_price": 9.99,
 "item_num": 2,
 "item_price": 12.99,
 "item_description": "Pepperoni Pizza"
 },
 {
 "order_id": 9,
 "status": "Processing",
 "date_created": "2024-03-25T14:00:00.000Z",
 "total_price": 9.99,
 "item_num": 3,
 "item_price": 12.99,
 "item_description": "Pepperoni Pizza"
 }
]
```

O9g: Successfully requested order id 10  
 Resource URI: <https://127.0.0.1:8080/order/getOrder>

Request Body:

```
{
 "order_id": 10
}
```

Response Code: **200**

Response Body:

```
[
 {
 "order_id": 10,
 "status": "Processing",
 "date_created": "2024-03-25T14:00:00.000Z",
 "total_price": 9.99,
 "item_num": 4,
 "item_price": 9.99,
 "item_description": "Pizza Margherita"
 },
 {
 "order_id": 10,
 "status": "Processing",
 "date_created": "2024-03-25T14:00:00.000Z",
 "total_price": 9.99,
 "item_num": 5,
 "item_price": 12.99,
 "item_description": "Pepperoni Pizza"
 },
 {
 "order_id": 10,
 "status": "Processing",
 "date_created": "2024-03-25T14:00:00.000Z",
 "total_price": 9.99,
 "item_num": 6,
 "item_price": 12.99,
 "item_description": "Pepperoni Pizza"
 }
]
```

Developed and tested by Vineal Sunkara

## Menu

### Menu-Load

SQL Queries Used:

```

const queryItems = await runQuery(customerPool, `SELECT * FROM
mrpizza.menu_item JOIN mrpizza.item_availability USING (mid) WHERE store_id =
${storeId} AND available = '1';`);

const queryToppings = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'toppings' AND available = '1';`);

const querySize = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'size' AND available = '1';`);

const queryServingOptions = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'servingOptions' AND available = '1';`);

const querySauces = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'sauces' AND available = '1';`);

const queryCrust = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'crust' AND available = '1';`);

const queryMeats = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'meats' AND available = '1';`);

const queryNonMeats = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'nonMeats' AND available = '1';`);
```

```
const queryCheese = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'cheese' AND available = '1';`);
```

Load-1: Successful acquisition of menu

Resource URI: /menu/1

Request Headers: None

Request Body:

None

Response Code: 200

Response Body:

```
{
 "mid": 1,
 "item_name": "Stuffed Cheesy Bread",
 "price": 8.99,
 "image":
"https://cache.dominos.com/olo/6_130_2/assets/build/market/US/_en/images/i
mg/products/larges/F_SCBRD.jpg",
 "description": "Our oven-baked breadsticks are stuffed and covered
in a blend of cheese made with 100% real mozzarella and cheddar. Seasoned
with a touch of garlic and Parmesan. Add marinara or your favorite dipping
cup for an additional charge.",
 "category": "Breads",
 "available": 1,
 "toppings": [],
 "size": [],
 "servingOptions": [],
 "sauces": [],
 "crust": [],
 "meats": [],
 "nonMeats": [],
 "cheese": []
},
{
 "mid": 2,
 "item_name": "Pepperoni Stuffed Cheesy Bread",
 "price": 8.99,
 "image":
"https://cache.dominos.com/olo/6_130_2/assets/build/market/US/_en/images/i
mg/products/larges/F_SPBRD.jpg",
 "description": "Our oven-baked breadsticks are stuffed with cheese
and pepperoni - covered in a blend of cheese made with 100% real
```

mozzarella and cheddar. Seasoned with a touch of garlic and Parmesan. Add marinara or your favorite dipping cup for an additional charge.",  
"category": "Breads", . . .

\*this body is 5820 lines long, thus only this portion will be given

Developed and tested by Keanu Melo Rojas

Load-2: Acquisition of a menu with no items available

Resource URI: /menu/2

Request Headers: None

Request Body:

None

Response Code: 200

Response Body:

{}

Developed and tested by Keanu Melo Rojas

Load-3: Error when store does not exist

Resource URI: /menu/3

Request Headers: None

Request Body:

None

Response Code: 200

Response Body:

```
{
 "error": "Invalid store id."
}
```

Developed and tested by Keanu Melo Rojas

## Menu-Search

SQL Queries Used:

```

const queryItems = await runQuery(customerPool, `SELECT * FROM
mrpizza.menu_item JOIN mrpizza.item_availability USING (mid) WHERE store_id =
${storeId} AND available = '1';`);

const queryToppings = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'toppings' AND available = '1';`);

const querySize = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'size' AND available = '1';`);

const queryServingOptions = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'servingOptions' AND available = '1';`);

const querySauces = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'sauces' AND available = '1';`);

const queryCrust = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'crust' AND available = '1';`);

const queryMeats = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'meats' AND available = '1';`);

const queryNonMeats = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'nonMeats' AND available = '1';`);

const queryCheese = await runQuery(customerPool, `SELECT * FROM
(mrpizza.custom_option JOIN mrpizza.custom_availability USING (custom_name, mid,
option_name)) JOIN mrpizza.custom USING (custom_name, mid) WHERE store_id =
${storeId} AND custom_name = 'cheese' AND available = '1';`);

```

Search-1: Successful reset of menu

Resource URI: /menu/1

Request Headers: None

Request Body:

None

Response Code: 200

Response Body:

```
{
 "mid": 1,
 "item_name": "Stuffed Cheesy Bread",
 "price": 8.99,
 "image":
"https://cache.dominos.com/olo/6_130_2/assets/build/market/US/_en/images/i
mg/products/larges/F_SCBRD.jpg",
 "description": "Our oven-baked breadsticks are stuffed and covered
in a blend of cheese made with 100% real mozzarella and cheddar. Seasoned
with a touch of garlic and Parmesan. Add marinara or your favorite dipping
cup for an additional charge.",
 "category": "Breads",
 "available": 1,
 "toppings": [],
 "size": [],
 "servingOptions": [],
 "sauces": [],
 "crust": [],
 "meats": [],
 "nonMeats": [],
 "cheese": []
},
{
 "mid": 2,
 "item_name": "Pepperoni Stuffed Cheesy Bread",
 "price": 8.99,
 "image":
"https://cache.dominos.com/olo/6_130_2/assets/build/market/US/_en/images/i
mg/products/larges/F_SPBRD.jpg",
 "description": "Our oven-baked breadsticks are stuffed with cheese
and pepperoni - covered in a blend of cheese made with 100% real
mozzarella and cheddar. Seasoned with a touch of garlic and Parmesan. Add
marinara or your favorite dipping cup for an additional charge.",
 "category": "Breads", . . .
}
```

\*this body is 5820 lines long, thus only this portion will be given

Developed and tested by Keanu Melo Rojas

## Check-Availability

CA1: Block user for using unavailable items.

Resource URI: /menu

Response Code: 200

Response Body:

**Item with MID '2' is not available.**

Developed and tested by Keanu Melo Rojas and Joshua Menezes

M-1: Menu not found

Resource URI:/menu

Response Code: 404

Response Body:

**404 on route /menu/menu.html**

Developed and tested by Keanu Melo Rojas and Joshua Menezes

## Check-Options-1

CO1: Alert user of item's size options.

Resource URI: /menu

Response Code: 200

Response Body:

**Item 'Antipasta' with MID '3' has no sizes.**

Developed and tested by Keanu Melo Rojas and Joshua Menezes

## Check-Options-2

CO2: Alert user of item's toppings options.

Resource URI: /menu

Response Code: 200

Response Body:

**Item 'Soda' with MID '4' has no toppings.**

Developed and tested by Keanu Melo Rojas and Joshua Menezes

## Admin-Edit-1

SQL Queries Used:

```
await runQuery(adminPool, `UPDATE mrpizza.menu_item SET ${property} = ${newValue} WHERE mid = '${mid}' AND item_name = '${name}'`);
```

AdminEdit1a: Edit item name for a menu item

Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "test",
 "property": "item_name",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit1b: Edit price for a menu item  
Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "4.00",
 "property": "price",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit1c: Edit image for a menu item  
Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue":
 "https://static.vecteezy.com/system/resources/previews/024/277/079/original/hamburger-fast-food-clipart-illustration-vector.jpg",
 "property": "image",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit1d: Edit description for a menu item  
Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "It tastes good, buy it twice",
 "property": "description",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit1e: Edit category for a menu item  
Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "Chicken",
 "property": "category",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit1f: Not signed in as admin  
Resource URI: POST /menu/edit

Request Headers: None

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "Chicken",
 "property": "category",
 "action": "edit"
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Admin-Edit-2

SQL Queries Used:

```
await runQuery(adminPool, `UPDATE mrpizza.custom_option SET ${property} = ${newValue} WHERE mid = '${mid}' AND option_name = '${name}' AND custom_name = '${custom_name}'`);
```

AdminEdit2a: Edit price for a customization option

Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "custom",
 "custom_name": "toppings",
 "name": "No Bacon",
 "mid": "11",
 "newValue": "4",
 "property": "price",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit2b: Edit whether a customization option is default  
Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "custom",
 "custom_name": "toppings",
 "name": "No Bacon",
 "mid": "11",
 "newValue": "false",
 "property": "isDefault",
 "action": "edit"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminEdit2c: Not signed in as admin  
Resource URI: POST /menu/edit

Request Headers: None

Request Body:

```
{
 "object": "custom",
 "custom_name": "toppings",
 "name": "No Bacon",
 "mid": "11",
 "newValue": "false",
 "property": "isDefault",
 "action": "edit"
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Admin-Delete-1

SQL Queries Used:

```
await runQuery(adminPool, 'DELETE FROM mrpizza.menu_item WHERE mid = ${mid} AND item_name = ${name}');
```

AdminDelete1a: Delete menu item

Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "delete",
 "property": "item_name",
 "action": "delete"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminDelete1b: Not signed in as admin  
Resource URI: POST /menu/edit

Request Headers: None

Request Body:

```
{
 "object": "item",
 "mid": "1",
 "name": "Stuffed Cheesy Bread",
 "newValue": "delete",
 "property": "item_name",
 "action": "delete"
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Admin-Delete-2

SQL Queries Used:

```
await runQuery(adminPool, 'DELETE FROM mrpizza.custom_option WHERE mid = '${mid}' AND option_name = '${name}' AND custom_name = '${custom_name}');
```

AdminDelete2a: Delete Customization Option

Resource URI: POST /menu/edit

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "custom",
 "custom_name": "toppings",
 "name": "No Bacon",
 "mid": "11",
 "newValue": "delete",
 "property": "isDefault",
 "action": "delete"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Menu item edited successfully"
}
```

Developed and tested by Joshua Menezes

AdminDelete2b: Not signed in as admin  
 Resource URI: POST /menu/edit

Request Headers: None

Request Body:

```
{
 "object": "custom",
 "custom_name": "toppings",
 "name": "No Bacon",
 "mid": "11",
 "newValue": "delete",
 "property": "isDefault",
 "action": "delete"
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Admin-Add-1

SQL Queries Used:

```

await runQuery(adminPool, `INSERT INTO mrpizza.menu_item (mid, price, image_url,
description, item_name, category) VALUES ('${mid}', '${price}', '${image}', '${description}',
='${item_name}', '${category}')`);

await runQuery(adminPool, `INSERT INTO mrpizza.item_availability (mid, store_id,
available) VALUES ('${mid}', '${storeId}', '${real_available}')`);
```

AdminAdd1a: Add menu item  
Resource URI: POST /menu/add

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "item",
 "description": "For people who like burger",
 "mid": "100",
 "price": "10",
 "image":
 "https://static.vecteezy.com/system/resources/previews/024/277/079/original/hamburger-fast-food-clipart-illustration-vector.jpg",
 "item_name": "Burger",
 "category": "Sandwiches",
 "available": "true"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Item added to menu successfully"
}
```

Developed and tested by Joshua Menezes

AdminAdd1b: Not signed in as admin  
Resource URI: POST /menu/add

Request Headers: None

Request Body:

```
{
 "object": "item",
 "description": "For people who like burger",
 "mid": "100",
 "price": "10",
 "image":
 "https://static.vecteezy.com/system/resources/previews/024/277/079/original/hamburger-fast-food-clipart-illustration-vector.jpg",
 "item_name": "Burger",
 "category": "Sandwiches",
 "available": "true"
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Admin-Add-2

SQL Queries Used:

```
await runQuery(adminPool, `INSERT INTO mrpizza.custom_option (custom_name, mid, option_name, price, isDefault) VALUES ('${custom_name}', '${mid}', '${option_name}', '${price}', '${Default}')`);

await runQuery(adminPool, `INSERT INTO mrpizza.custom_availability (mid, custom_name, option_name, store_id, available) VALUES ('${mid}', '${custom_name}', '${option_name}', '${storeId}', '${real_available}')`);
```

AdminAdd2a: Add Customization Options

Resource URI: POST /menu/add

Request Headers: Valid Auth Token Cookie, signed in as admin

Request Body:

```
{
 "object": "custom",
 "custom_name": "toppings",
 "mid": "11",
 "option_name": "maple syrup",
 "price": "10",
 "isDefault": "false",
 "available": "1",
 "mutually_exclusive": "0"
}
```

Response Code: 200

Response Body:

```
{
 "message": "Item added to menu successfully"
}
```

Developed and tested by Joshua Menezes

AdminAdd2b: Not signed in as admin  
Resource URI: POST /menu/add

Request Headers: None

Request Body:

```
{
 "object": "item",
 "description": "For people who like burger",
 "mid": "100",
 "price": "10",
 "image":
 "https://static.vecteezy.com/system/resources/previews/024/277/079/original/hamburger-fast-food-clipart-illustration-vector.jpg",
 "item_name": "Burger",
 "category": "Sandwiches",
 "available": "true"
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Employee-Availability-1

SQL Queries Used:

```
await runQuery(employeePool, `UPDATE mrpizza.item_availability SET available = ${real_available} WHERE store_id = ${storeId} AND mid = ${mid}`);
```

EmployeeAvailability1a: Update item availability

Resource URI: POST /menu/availability

Request Headers: Valid Auth Token Cookie, signed in as Employee

Request Body:

```
{
 "object": "item",
 "storeId": "1",
 "mid": "1",
 "available": "false",
}
```

Response Code: 200

Response Body:

```
{
 "message": "Availability edited successfully"
}
```

Developed and tested by Joshua Menezes

EmployeeAvailability1b: Not signed in as employee  
Resource URI: POST /menu/availability

Request Headers: None

Request Body:

```
{
 "object": "item",
 "storeId": "1",
 "mid": "1",
 "available": "false",
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Employee-Availability-2

SQL Queries Used:

```
await runQuery(employeePool, `UPDATE mrpizza.custom_availability
SET available = '${real_available}' WHERE store_id = '${storeId}' AND mid =
'${mid}' AND option_name = '${option_name}' AND custom_name =
'${custom_name}'`);
```

EmployeeAvailability2a: Update customization options availability

Resource URI: POST /menu/availability

Request Headers: Valid Auth Token Cookie, signed in as Employee

Request Body:

```
{
 "object": "custom",
 "storeId": "1",
 "mid": "11",
 "option_name": "No Bacon",
 "custom_name": "toppings",
 "available": "false",
}
```

Response Code: 200

Response Body:

```
{
 "message": "Availability edited successfully"
}
```

Developed and tested by Joshua Menezes

EmployeeAvailability2b: Not signed in as employee  
Resource URI: POST /menu/availability

Request Headers: None

Request Body:

```
{
 "object": "custom",
 "storeId": "1",
 "mid": "11",
 "option_name": "No Bacon",
 "custom_name": "toppings",
 "available": "false",
}
```

Response Code: 401

Response Body:

```
{
 You are not signed in.
}
```

Developed and tested by Joshua Menezes

## Report 3: Third-Party API Documentation

### Map/Driver Subgroup

#### Calculate Route Matrix for Delivery1: Unassigned Order Time Estimation

Resource URI: POST <https://routes.googleapis.com/distanceMatrix/v2:computeRouteMatrix>

Database Status: finding distance between Mr. Pizza Store, 6 delivery locations, and each other.

Request Headers:

```
'X-Goog-Api-Key': 'AIzaSyD0Y7j1X8Ug2psarQtGmCnwuRYdz00-ENk'
'X-Goog-FieldMask': 'originIndex,destinationIndex,duration,distanceMeters,
condition'
```

Request Body:

```
{
 "origins": [
 {
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.523421858838276,
 "longitude": -74.45823918823967
 }
 }
 },
 "routeModifiers": {
 "avoid_ferries": true
 }
 }
],
 "destinations": [
 {
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.41829236146375,
 "longitude": -74.70639548636213
 }
 }
 }
 },
 {
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.42837937050338,
 "longitude": -74.689193548636213
 }
 }
 }
 }
]
}
```

```
 "longitude": -74.67260209649424
 }
}
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.42837937050338,
 "longitude": -74.63260209649424
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.6573258481948,
 "longitude": -74.3632707679412
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.646906977241095,
 "longitude": -74.34146977403721
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.6622742419765,
 "longitude": -74.31554890726946
 }
 }
 }
}
],
"travelMode": "DRIVE",
"routingPreference": "TRAFFIC_AWARE"
}
```

Response Code: 200

Response Body:

```
[
 {
 "originIndex": 0,
 "destinationIndex": 3,
 "status": {},
 "distanceMeters": 24719,
 "duration": "1717s",
 "condition": "ROUTE_EXISTS"
 },
 {
 "originIndex": 0,
 "destinationIndex": 0,
 "status": {},
 "distanceMeters": 32112,
 "duration": "1990s",
 "condition": "ROUTE_EXISTS"
 },
 {
 "originIndex": 0,
 "destinationIndex": 2,
 "status": {},
 "distanceMeters": 26633,
 "duration": "1709s",
 "condition": "ROUTE_EXISTS"
 },
 {
 "originIndex": 0,
 "destinationIndex": 4,
 "status": {},
 "distanceMeters": 21669,
 "duration": "1743s",
 "condition": "ROUTE_EXISTS"
 },
 {
 "originIndex": 0,
 "destinationIndex": 1,
 "status": {},
 "distanceMeters": 29422,
 "duration": "1860s",
 "condition": "ROUTE_EXISTS"
 },
 {
 "originIndex": 0,
 "destinationIndex": 5,
 "status": {},
 "distanceMeters": 28572,
 "duration": "1802s",
 "condition": "ROUTE_EXISTS"
}
```

]

## Calculate Route Matrix for Delivery3: Optimal Assignment Calculation

Resource URI: POST <https://routes.googleapis.com/distanceMatrix/v2:computeRouteMatrix>

Database Status: finding approximate time for delivery of 6 unassigned orders

Request Headers:

```
'X-Goog-Api-Key': 'AIzaSyD0Y7j1X8Ug2psarQtGmCnwuRYdz00-ENk'
'X-Goog-FieldMask': 'originIndex,destinationIndex,duration,distanceMeters,
status,condition'
```

Request Body:

```
{
 "origins": [
 {
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.41829236146375,
 "longitude": -74.70639548636213
 }
 }
 },
 "routeModifiers": {
 "avoid_ferries": true
 }
 },
 {
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.42837937050338,
 "longitude": -74.67260209649424
 }
 }
 },
 "routeModifiers": {
 "avoid_ferries": true
 }
 },
 {
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.42837937050338,
 "longitude": -74.63260209649424
 }
 }
 },
 "routeModifiers": {
```

```
 "avoid_ferries": true
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.6573258481948,
 "longitude": -74.3632707679412
 }
 }
 },
 "routeModifiers": {
 "avoid_ferries": true
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.646906977241095,
 "longitude": -74.34146977403721
 }
 }
 },
 "routeModifiers": {
 "avoid_ferries": true
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.6622742419765,
 "longitude": -74.31554890726946
 }
 }
 },
 "routeModifiers": {
 "avoid_ferries": true
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.523421858838276,
 "longitude": -74.45823918823967
 }
 }
 },
 "routeModifiers": {
```

```
 "avoid_ferries": true
 }
},
],
"destinations": [
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.41829236146375,
 "longitude": -74.70639548636213
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.42837937050338,
 "longitude": -74.67260209649424
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.42837937050338,
 "longitude": -74.63260209649424
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.6573258481948,
 "longitude": -74.3632707679412
 }
 }
 }
},
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.646906977241095,
 "longitude": -74.34146977403721
 }
 }
 }
}
```

```

 }
 },
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.6622742419765,
 "longitude": -74.31554890726946
 }
 }
 },
{
 "waypoint": {
 "location": {
 "latLng": {
 "latitude": 40.523421858838276,
 "longitude": -74.45823918823967
 }
 }
 }
},
"travelMode": "DRIVE",
"routingPreference": "TRAFFIC_AWARE"
}

```

Response Code: 200

Response Body:

```
[
{
 "originIndex": 4,
 "destinationIndex": 5,
 "distanceMeters": 3497,
 "duration": "356s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 4,
 "destinationIndex": 3,
 "distanceMeters": 2701,
 "duration": "294s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 5,
 "destinationIndex": 4,
 "distanceMeters": 3805,
 "duration": "386s",
}
```

```
"condition": "ROUTE_EXISTS"
},
{
 "originIndex": 5,
 "destinationIndex": 3,
 "distanceMeters": 5174,
 "duration": "509s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 6,
 "destinationIndex": 3,
 "distanceMeters": 24719,
 "duration": "1733s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 6,
 "destinationIndex": 1,
 "distanceMeters": 29422,
 "duration": "1849s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 5,
 "destinationIndex": 6,
 "distanceMeters": 34777,
 "duration": "1770s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 6,
 "destinationIndex": 5,
 "distanceMeters": 28572,
 "duration": "1801s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 3,
 "destinationIndex": 3,
 "duration": "0s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 3,
 "destinationIndex": 4,
 "distanceMeters": 2723,
 "duration": "288s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 5,
```

```
"destinationIndex": 1,
"distanceMeters": 51402,
"duration": "2692s",
"condition": "ROUTE_EXISTS"
},
{
 "originIndex": 4,
 "destinationIndex": 4,
 "duration": "0s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 4,
 "destinationIndex": 1,
 "distanceMeters": 53490,
 "duration": "2730s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
 "destinationIndex": 2,
 "distanceMeters": 8282,
 "duration": "635s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 6,
 "destinationIndex": 6,
 "duration": "0s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
 "destinationIndex": 1,
 "distanceMeters": 3865,
 "duration": "350s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 6,
 "destinationIndex": 0,
 "distanceMeters": 32112,
 "duration": "1969s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 2,
 "destinationIndex": 6,
 "distanceMeters": 27080,
 "duration": "1764s",
 "condition": "ROUTE_EXISTS"
},
```

```
{
 "originIndex": 3,
 "destinationIndex": 1,
 "distanceMeters": 47557,
 "duration": "2451s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 3,
 "destinationIndex": 5,
 "distanceMeters": 5184,
 "duration": "517s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 3,
 "destinationIndex": 0,
 "distanceMeters": 50248,
 "duration": "2581s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 2,
 "destinationIndex": 1,
 "distanceMeters": 6015,
 "duration": "434s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 1,
 "destinationIndex": 2,
 "distanceMeters": 4844,
 "duration": "415s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 1,
 "destinationIndex": 1,
 "duration": "0s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 2,
 "destinationIndex": 3,
 "distanceMeters": 42492,
 "duration": "2483s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 2,
 "destinationIndex": 0,
 "distanceMeters": 8222,
```

```
"duration": "615s",
"condition": "ROUTE_EXISTS"
},
{
 "originIndex": 1,
 "destinationIndex": 4,
 "distanceMeters": 45470,
 "duration": "2650s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
 "destinationIndex": 5,
 "distanceMeters": 51678,
 "duration": "2943s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 1,
 "destinationIndex": 5,
 "distanceMeters": 48929,
 "duration": "2775s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
 "destinationIndex": 6,
 "distanceMeters": 32594,
 "duration": "2091s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
 "destinationIndex": 3,
 "distanceMeters": 48023,
 "duration": "2646s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 5,
 "destinationIndex": 5,
 "duration": "0s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
 "destinationIndex": 0,
 "duration": "0s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 2,
```

```
"destinationIndex": 2,
"duration": "0s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 1,
"destinationIndex": 0,
"distanceMeters": 3898,
"duration": "334s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 3,
"destinationIndex": 6,
"distanceMeters": 20323,
"duration": "1719s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 6,
"destinationIndex": 2,
"distanceMeters": 26633,
"duration": "1685s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 6,
"destinationIndex": 4,
"distanceMeters": 21669,
"duration": "1743s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 5,
"destinationIndex": 2,
"distanceMeters": 47143,
"duration": "2712s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 4,
"destinationIndex": 2,
"distanceMeters": 53047,
"duration": "2562s",
"condition": "ROUTE_EXISTS"
},
{
"originIndex": 2,
"destinationIndex": 5,
"distanceMeters": 46146,
"duration": "2773s",
"condition": "ROUTE_EXISTS"
```

```
},
{
 "originIndex": 1,
 "destinationIndex": 6,
 "distanceMeters": 29845,
 "duration": "1948s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 4,
 "destinationIndex": 6,
 "distanceMeters": 31399,
 "duration": "1583s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 3,
 "destinationIndex": 2,
 "distanceMeters": 43299,
 "duration": "2458s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 4,
 "destinationIndex": 0,
 "distanceMeters": 54471,
 "duration": "2766s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 1,
 "destinationIndex": 3,
 "distanceMeters": 45275,
 "duration": "2487s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 5,
 "destinationIndex": 0,
 "distanceMeters": 54092,
 "duration": "2838s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 2,
 "destinationIndex": 4,
 "distanceMeters": 42687,
 "duration": "2631s",
 "condition": "ROUTE_EXISTS"
},
{
 "originIndex": 0,
```

```
"destinationIndex": 4,
"distanceMeters": 48218,
"duration": "2785s",
"condition": "ROUTE_EXISTS"
}
]
```

## Address Validation Request for Map6a: Validate Input Address, Accepted Address

Resource URI: POST

[https://addressvalidation.googleapis.com/v1/validateAddress?key=\\${googleMapsApiKey}](https://addressvalidation.googleapis.com/v1/validateAddress?key=${googleMapsApiKey})

googleMapsApiKey = 'AIzaSyD0Y7j1X8Ug2psarQtGmCnwuRYdz00-ENk'

Request Headers: None

Request Body:

```
{
 "address": {
 "regionCode": "US",
 "locality": "Piscataway",
 "administrativeArea": "NJ",
 "postalCode": "08854",
 "addressLines": [
 "220 Marvin Lane",
 ""
]
 },
 "enableUspsCass": true
}
```

Response Code: 200

Response Body:

```
{
 "result": {
 "verdict": {
 "inputGranularity": "PREMISE",
 "validationGranularity": "PREMISE",
 "geocodeGranularity": "PREMISE",
 "addressComplete": true
 },
 "address": {
 "formattedAddress": "220 Marvin Lane, Piscataway, NJ 08854,
USA",
 "postalAddress": {
 "regionCode": "US",
 "languageCode": "en",
 "postalCode": "08854",
 "administrativeArea": "NJ",
 "locality": "Piscataway",
 "addressLines": [
 "220 Marvin Ln"
]
 },
 "addressComponents": [
 {
 "name": "220 Marvin Ln",
 "type": "street_address"
 },
 {
 "name": "Piscataway",
 "type": "locality"
 },
 {
 "name": "New Jersey",
 "type": "administrative_area_level_1"
 },
 {
 "name": "United States",
 "type": "country"
 }
]
 }
 }
}
```

```
 "componentName": {
 "text": "220"
 },
 "componentType": "street_number",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "Marvin Lane",
 "languageCode": "en"
 },
 "componentType": "route",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "Piscataway",
 "languageCode": "en"
 },
 "componentType": "locality",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "NJ",
 "languageCode": "en"
 },
 "componentType": "administrative_area_level_1",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "08854"
 },
 "componentType": "postal_code",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "USA",
 "languageCode": "en"
 },
 "componentType": "country",
 "confirmationLevel": "CONFIRMED"
 }
],
},
"geocode": {
 "location": {
 "latitude": 40.5187174,
 "longitude": -74.4553767
 },
}
```

```
 "plusCode": {
 "globalCode": "87G7GG9V+FR"
 },
 "bounds": {
 "low": {
 "latitude": 40.5187174,
 "longitude": -74.4553767
 },
 "high": {
 "latitude": 40.5187174,
 "longitude": -74.4553767
 }
 },
 "placeId": "ChIJ_Wf8MQrHw4kRTdGJ4vwJL9o",
 "placeTypes": [
 "street_address"
]
 },
 "metadata": {
 "business": false,
 "residential": true
 },
 "uspsData": {
 "standardizedAddress": {
 "firstAddressLine": "220 MARVIN LANE",
 "cityStateZipAddressLine": "PISCATAWAY NJ 08854",
 "city": "PISCATAWAY",
 "state": "NJ",
 "zipCode": "08854"
 },
 "dpvFootnote": "A1",
 "postOfficeCity": "PISCATAWAY",
 "postOfficeState": "NJ",
 "cassProcessed": true
 }
},
"responseId": "4132e097-3d76-4ef2-83d3-8aebb14adffb"
}
```

## Address Validation Request for Map6b: Validate Input Address, Invalid Address – Verification Required

Resource URI: POST

[https://addressvalidation.googleapis.com/v1/validateAddress?key=\\${googleMapsApiKey}](https://addressvalidation.googleapis.com/v1/validateAddress?key=${googleMapsApiKey})

googleMapsApiKey = 'AIzaSyD0Y7j1X8Ug2psarQtGmCnwuRYdz0O-ENk'

Request Headers: None

Request Body:

```
{
 "address": {
 "regionCode": "US",
 "locality": "Piscataway",
 "administrativeArea": "NJ",
 "postalCode": "",
 "addressLines": [
 "220 Marvin Lane",
 ""
]
 },
 "enableUspsCass": true
}
```

Response Code: 200

Response Body:

```
{
 "result": {
 "verdict": {
 "inputGranularity": "PREMISE",
 "validationGranularity": "PREMISE",
 "geocodeGranularity": "PREMISE",
 "addressComplete": true,
 "hasInferredComponents": true
 },
 "address": {
 "formattedAddress": "220 Marvin Lane, Piscataway, NJ 08854,
 USA",
 "postalAddress": {
 "regionCode": "US",
 "languageCode": "en",
 "postalCode": "08854",
 "administrativeArea": "NJ",
 "locality": "Piscataway",
 "addressLines": [
 "220 Marvin Ln"
]
 }
 }
 }
}
```

```
"addressComponents": [
 {
 "componentName": {
 "text": "220"
 },
 "componentType": "street_number",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "Marvin Lane",
 "languageCode": "en"
 },
 "componentType": "route",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "Piscataway",
 "languageCode": "en"
 },
 "componentType": "locality",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "NJ",
 "languageCode": "en"
 },
 "componentType": "administrative_area_level_1",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "USA",
 "languageCode": "en"
 },
 "componentType": "country",
 "confirmationLevel": "CONFIRMED"
 },
 {
 "componentName": {
 "text": "08854"
 },
 "componentType": "postal_code",
 "confirmationLevel": "CONFIRMED",
 "inferred": true
 }
],
"geocode": {
 "location": {
```

```
 "latitude": 40.5187174,
 "longitude": -74.4553767
 },
 "plusCode": {
 "globalCode": "87G7GG9V+FR"
 },
 "bounds": {
 "low": {
 "latitude": 40.5187174,
 "longitude": -74.4553767
 },
 "high": {
 "latitude": 40.5187174,
 "longitude": -74.4553767
 }
 },
 "placeId": "ChIJ_Wf8MQrHw4kRTdGJ4vwJL9o",
 "placeTypes": [
 "street_address"
]
},
"metadata": {
 "business": false,
 "residential": true
},
"uspsData": {
 "standardizedAddress": {
 "firstAddressLine": "220 MARVIN LANE",
 "cityStateZipAddressLine": "PISCATAWAY NJ",
 "city": "PISCATAWAY",
 "state": "NJ"
 },
 "dpvFootnote": "A1",
 "postOfficeCity": "PISCATAWAY",
 "postOfficeState": "NJ",
 "cassProcessed": true
}
},
"responseId": "bca7b508-38b6-42e3-8c93-e5972a0a2889"
}
```

## Address Validation Request for Map6c: Validate Input Address, Invalid Address – Fix Required

Resource URI: POST

[https://addressvalidation.googleapis.com/v1:validateAddress?key=\\${googleMapsApiKey}](https://addressvalidation.googleapis.com/v1:validateAddress?key=${googleMapsApiKey})

googleMapsApiKey = 'AIzaSyD0Y7j1X8Ug2psarQtGmCnwuRYdz0O-ENk'

Request Headers: None

Request Body:

```
{
 "address": {
 "regionCode": "US",
 "locality": "Piscataway",
 "administrativeArea": "NJ",
 "postalCode": "",
 "addressLines": [
 "hello",
 ""
]
 },
 "enableUspsCass": true
}
```

Response Code: 200

Response Body:

```
{
 "result": {
 "verdict": {
 "inputGranularity": "PREMISE",
 "validationGranularity": "OTHER",
 "geocodeGranularity": "OTHER",
 "hasUnconfirmedComponents": true,
 "hasInferredComponents": true
 },
 "address": {
 "formattedAddress": "hello, Piscataway, NJ 08854, USA",
 "postalAddress": {
 "regionCode": "US",
 "languageCode": "en",
 "postalCode": "08854",
 "administrativeArea": "NJ",
 "locality": "Piscataway",
 "addressLines": [
 "hello"
]
 }
 },
 "addressComponents": [
 ...
]
 }
}
```

```
{
 "componentName": {
 "text": "hello",
 "languageCode": "en"
 },
 "componentType": "point_of_interest",
 "confirmationLevel": "UNCONFIRMED_BUT_PLAUSIBLE"
},
{
 "componentName": {
 "text": "Piscataway",
 "languageCode": "en"
 },
 "componentType": "locality",
 "confirmationLevel": "CONFIRMED"
},
{
 "componentName": {
 "text": "NJ",
 "languageCode": "en"
 },
 "componentType": "administrative_area_level_1",
 "confirmationLevel": "CONFIRMED"
},
{
 "componentName": {
 "text": "USA",
 "languageCode": "en"
 },
 "componentType": "country",
 "confirmationLevel": "CONFIRMED"
},
{
 "componentName": {
 "text": "08854"
 },
 "componentType": "postal_code",
 "confirmationLevel": "CONFIRMED",
 "inferred": true
}
],
"missingComponentTypes": [
 "route",
 "street_number"
],
"unconfirmedComponentTypes": [
 "point_of_interest"
]
},
"geocode": {
 "location": {
 "latitude": 40.554887,
 "longitude": -74.40625
 }
}
```

```
 "longitude": -74.4642861
 },
 "plusCode": {
 "globalCode": "87G7HG3P+X7"
 },
 "bounds": {
 "low": {
 "latitude": 40.506009,
 "longitude": -74.519503
 },
 "high": {
 "latitude": 40.5971879,
 "longitude": -74.4086819
 }
 },
 "featureSizeMeters": 7184.945,
 "placeId": "ChIJ1UF7VI04w4kReB1HYVDPmrY",
 "placeTypes": [
 "locality",
 "political"
]
},
"uspsData": {
 "standardizedAddress": {
 "firstAddressLine": "HELLO",
 "cityStateZipAddressLine": "PISCATAWAY NJ",
 "city": "PISCATAWAY",
 "state": "NJ"
 },
 "dpvFootnote": "A1M1",
 "postOfficeCity": "PISCATAWAY",
 "postOfficeState": "NJ",
 "cassProcessed": true
},
{
 "responseId": "250ab604-a491-41be-9b73-5eb14e67c754"
}
```

## Front-end API Usage

To display the maps and routes on the front end, we use the following mjs module imports:

```
const { Map } = await google.maps.importLibrary("maps");
const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");
const { DirectionsService } = google.maps;
const directionsService = new DirectionsService();
```

## Pizza Pipeline Subgroup

### Interaction with Stripe API for O4

SQL Queries Used:

```
await runQuery(customerPool, 'UPDATE customer_order SET stripe_checkout_id = ?, stripe_payment_intent_id = ? WHERE order_id = ?', [session.id, session.payment_intent, decodedOrderId]);
```

O4-1a: Successful payment with Stripe (valid input)

Resource URI: /order/createCheckoutSession

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULApD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

```
Customer must have items in cart in order to be able to create a checkout
session and pay with stripe
```

Request Body:

```
{
 "total":3
}
```

Response Code: 200

Response Body:

```
{
 "id": "cs_test_a1gGWQ2gqICSHqXriJD3zRQVCmfvxuTklnrZFaPldPM9bQtTXwC0ftSNRO",
 "object": "checkout.session",
 "after_expiration": null,
 "allow_promotion_codes": null,
 "amount_subtotal": 300,
 "amount_total": 300,
 "automatic_tax": {
```

```
 "enabled": false,
 "liability": null,
 "status": null
 },
 "billing_address_collection": null,
 "cancel_url": null,
 "client_reference_id": null,
 "client_secret":
"cs_test_a1gGWQ2gqICSHqXriJD3zRQVCmfvxuTklnrZFaPldPM9bQtTXwC0ftSNR0_secret
_fidwbEhqYWAnPydgaGdgYWFgYScpJ2lkfGpwcvF8dWAnPyd2bGtiawBabHFgaCcpJ3dgYlx3Y
GZxSmtGamh1aWBxbGprJz8nZGlyZHx2J3gl",
 "consent": null,
 "consent_collection": null,
 "created": 1712018532,
 "currency": "usd",
 "currency_conversion": null,
 "custom_fields": [],
 "custom_text": {
 "after_submit": null,
 "shipping_address": null,
 "submit": null,
 "terms_of_service_acceptance": null
 },
 "customer": null,
 "customer_creation": "if_required",
 "customer_details": null,
 "customer_email": null,
 "expires_at": 1712104932,
 "invoice": null,
 "invoice_creation": {
 "enabled": false,
 "invoice_data": {
 "account_tax_ids": null,
 "custom_fields": null,
 "label": null
 }
 }
}
```

```
 "description": null,
 "footer": null,
 "issuer": null,
 "metadata": {},
 "rendering_options": null
 }
},
"livemode": false,
"locale": null,
"metadata": {},
"mode": "payment",
"payment_intent": null,
"payment_link": null,
"payment_method_collection": "if_required",
"payment_method_configuration_details": null,
"payment_method_options": {
 "card": {
 "request_three_d_secure": "automatic"
 }
},
"payment_method_types": [
 "card"
],
"payment_status": "unpaid",
"phone_number_collection": {
 "enabled": false
},
"recovered_from": null,
"redirect_on_completion": "always",
"return_url": "https://127.0.0.1:8080/order/orderStatus.html",
"setup_intent": null,
"shipping_address_collection": null,
"shipping_cost": null,
"shipping_details": null,
```

```

"shipping_options": [],
"status": "open",
"submit_type": null,
"subscription": null,
"success_url": null,
"total_details": {
 "amount_discount": 0,
 "amount_shipping": 0,
 "amount_tax": 0
},
"ui_mode": "embedded",
"url": null
}

```

Developed and tested by Nikash Rajeshbabu

O41b: Unsuccessful payment with stripe due to invalid JSON

Resource URI: /order/createCheckoutSession

Request Headers:

```

POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImLhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure

```

Database Status:

```

Customer must have items in cart in order to be able to create a checkout
session and pay with stripe

```

Request Body:

```
{}
```

Response Code: **400**

Response Body:

```
{}
```

```

 "status": "error",
 "message": "Invalid JSON format in request body."
}

```

O41c: Unsuccessful payment with stripe due to internal server error

Request Headers:

```

POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure

```

Database Status:

```
Customer must have items in cart in order to be able to create a checkout session and pay with stripe
```

Request Body:

```
{
 "total":3
}
```

Response Code: **500**

Response Body:

```
{
 "status": "error",
 "message": "Internal Server Error: Failed to create Stripe checkout session."
}
```

O41d: Unsuccessful payment with stripe due to invalid credentials

Resource URI: /order/createCheckoutSession

Request Headers:

```
NONE
```

Database Status:

Customer is not signed in and has nothing in their cart

Request Body:

```
{
 "total":3
}
```

Response Code:401

Response Body:

```
{
 Invalid Credentials
}
```

## Stripe Interaction for O4-2

SQL Queries Used:

```
let result = await runQuery(adminPool,
 `UPDATE customer_order SET status = "Paid" WHERE
stripe_checkout_id = ?`,
 [checkoutId])

let recentOrderQuery = "SELECT order_id FROM customer_order WHERE
stripe_checkout_id = ?";

let getCidQuery = "SELECT ordered_by FROM customer_order WHERE order_id
= " + order_id;

let checkTotalQuery = "SELECT total_price FROM customer_order WHERE
order_id = " + order_id;

let addRewardsQuery = "UPDATE customer_account SET rewards_points =
rewards_points + 1 WHERE cid = " + customerID;

let emailQuery = "SELECT email FROM customer_account WHERE cid = " +
customerID;

const query = `
 SELECT co.order_id,
 co.status,
 co.DT_created AS date_created,
```

```
 co.total_price,
 co.delivery_address,
 oi.item_num,
 mi.price AS item_price,
 mi.item_name AS item_description
FROM customer_order AS co
LEFT JOIN order_item AS oi ON co.order_id = oi.order_id
LEFT JOIN menu_item AS mi ON oi.mid = mi.mid
WHERE co.order_id = ?
`;
```

O42a: Unsuccessful confirmation with Stripe, sends email confirmation and updates reward points (valid input)

Resource URI: /order/stripeWebhook

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

Database Status:

For successful checks there needs to be verified order id in the database, with checkout id, and connected to logged in customer

Request Body:

```
{}
```

Response Code: **400**

Response Body:

```
Bad Request: Invalid JSON data received.
```

Request Body:

```
{"hello":1}
```

Response Code: **400**

Response Body:

```
Bad Request: Invalid Stripe Event.
```

O4-2b: Successful confirmation with Stripe, sends email confirmation and updates reward points (valid input)

Resource URI: /order/stripeWebhook

Request Headers:

```
POST,
(customer) Cookie:
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXR0Um9sZSI6ImN1c3RvbWV
yIiwiaWQiOjQsImlhdCI6MTcxNDg2MTA3Mn0.iB86bDH5zzD2yS00WYlSoULAypD8iJ0VwJti1
7lCcqk; Max-Age=86400; Path=/; HttpOnly; Secure
```

## Request Body:

```
{
 "id": "evt_3PCqLCP5gIWmEZ1P1JFTvv2B",
 "object": "event",
 "api_version": "2023-10-16",
 "created": 1714858920,
 "data": {
 "object": {
 "id": "pi_3PCqLCP5gIWmEZ1P16borUDk",
 "object": "payment_intent",
 "amount": 2597,
 "amount_capturable": 0,
 "amount_details": {
 "tip": {}
 },
 "amount_received": 2597,
 "application": null,
 "application_fee_amount": null,
 "automatic_payment_methods": null,
 "canceled_at": null,
 "cancellation_reason": null,
 "capture_method": "automatic",
 "client_secret":
 "pi_3PCqLCP5gIWmEZ1P16borUDk_secret_U5gf0vLoKJk0chIAgfBMXqjm0",
 "confirmation_method": "automatic",
 "created": 1714858918,
 "currency": "usd",
 "customer": null,
 "description": null,
 "invoice": null,
 "last_payment_error": null,
 "latest_charge": "ch_3PCqLCP5gIWmEZ1P1weLtXRb",
 "livemode": false,
 "metadata": {},
 "next_action": null,
 "on_behalf_of": null,
 "payment_method": "pm_1PCqLCP5gIWmEZ1PobU06HKf",
 "payment_method_configuration_details": null,
 "payment_method_options": {
 "card": {
 "installments": null,
 "mandate_options": null,
 "network": null,
 "request_three_d_secure": "automatic"
 }
 },
 "payment_method_types": [
 "card"
]
 }
 }
}
```

```

],
 "processing": null,
 "receipt_email": null,
 "review": null,
 "setup_future_usage": null,
 "shipping": null,
 "source": null,
 "statement_descriptor": null,
 "statement_descriptor_suffix": null,
 "status": "succeeded",
 "transfer_data": null,
 "transfer_group": null
 }
},
"livemode": false,
"pending_webhooks": 1,
"request": {
 "id": "req_BKDvi7QbysQ4wq",
 "idempotency_key": "49c70eb7-c5f8-490d-a0a3-518d1ea38291"
},
"type": "payment_intent.succeeded"
}

```

Response Code: **200**

Response Body:

```
Success, but session has not been completed.
```

Request Body:

```
{
 "id": "evt_3PCwgTP5gIWmEZ1P1U7rqln4",
 "object": "event",
 "api_version": "2023-10-16",
 "created": 1714883302,
 "data": {
 "object": {
 "id": "pi_3PCwgTP5gIWmEZ1P1cYmpmGb",
 "object": "payment_intent",
 "amount": 17589,
 "amount_capturable": 0,
 "amount_details": {
 "tip": {}
 },
 "amount_received": 17589,
 "application": null,
 "application_fee_amount": null,
 "cancel_reason": null,
 "closed": false,
 "confirmation": null,
 "confirmation_method": null,
 "currency": "USD",
 "customer": null,
 "customer_email": null,
 "customer_name": null,
 "description": null,
 "dispute": null,
 "failure_code": null,
 "failure_message": null,
 "final_payment_intent": null,
 "fraud_review": null,
 "group_id": null,
 "livemode": false,
 "method": "card",
 "mode": "payment",
 "on_behalf_of": null,
 "order": null,
 "paid": false,
 "payment": null,
 "payment_method": null,
 "payment_method_types": [
 "card"
],
 "payment_status": "not_set",
 "receipt_email": null,
 "receipt_phone": null,
 "review": null,
 "setup_future_usage": null,
 "shipping": null,
 "source": null,
 "statement_descriptor": null,
 "statement_descriptor_suffix": null,
 "status": "succeeded",
 "transfer_data": null,
 "transfer_group": null
 }
 }
}
```

```
"automatic_payment_methods": null,
"canceled_at": null,
cancellation_reason": null,
capture_method": "automatic",
"client_secret": "pi_3PCwgTP5gIWmEZ1P1cYmpmGb_secret_ssEUcWYWM9nFZdgrleOs1fq3D",
"confirmation_method": "automatic",
"created": 1714883301,
"currency": "usd",
"customer": null,
"description": null,
"invoice": null,
"last_payment_error": null,
"latest_charge": "ch_3PCwgTP5gIWmEZ1P1VZrxpgR",
"livemode": false,
"metadata": {},
"next_action": null,
"on_behalf_of": null,
"payment_method": "pm_1PCwgTP5gIWmEZ1P1YH7TgNp",
"payment_method_configuration_details": null,
"payment_method_options": {
 "card": {
 "installments": null,
 "mandate_options": null,
 "network": null,
 "request_three_d_secure": "automatic"
 }
},
"payment_method_types": [
 "card"
],
"processing": null,
"receipt_email": null,
"review": null,
"setup_future_usage": null,
"shipping": null,
"source": null,
"statement_descriptor": null,
"statement_descriptor_suffix": null,
"status": "succeeded",
"transfer_data": null,
"transfer_group": null
},
"livemode": false,
"pending_webhooks": 1,
"request": {
 "id": "req_0CItlof7noR8aK",
 "idempotency_key": "58ab24bd-5671-4cb1-a124-a88b59530373"
},
"type": "payment_intent.succeeded"
```

```
}
```

Response Code: **200**

Response Body:

```
Stripe confirmation successful.
```

O4-2c: Unsuccessful payment with stripe due to invalid credentials

Resource URI: /order/createCheckoutSession

Request Headers:

```
NONE
```

Database Status:

```
Customer is not signed in
```

Request Body:

```
{
 "total":3
}
```

Response Code:401

Response Body:

```
{
 Invalid Credentials
}
```

Developed and tested by Nikash Rajeshbabu

# Report 3 Contributions Tables

## General Setup

| Item                           | People                         |
|--------------------------------|--------------------------------|
| Steps to Run Server            | Brandon Cheng(75%), Ji Wu(25%) |
| Backend Implementation Details | Brandon Cheng(100%)            |

## MAP/DRIVER GROUP

| Item                          | People                                               |
|-------------------------------|------------------------------------------------------|
| API Updates Table             | Ji Wu(100%)                                          |
| Server-Side Routing Diagram   | Damon Lin(50%), Brandon Cheng(50%)                   |
| API Documentation/Use Cases   | Ji Wu(33.3%), Brandon Cheng(33.3%), Damon Lin(33.3%) |
| Third Party API Documentation | Brandon Cheng(90%), Damon Lin(10%)                   |

## PIZZA PIPELINE GROUP

| Item                          | People                                                                 |
|-------------------------------|------------------------------------------------------------------------|
| API Updates Table             | Vineal Sunkara (50%) and Arya Shetty (50%)                             |
| Server-Side Routing Diagram   | Vineal Sunkara (100%)                                                  |
| API Documentation/Use Cases   | Vineal Sunkara (33.3%), Nikash Rajeshbabu (33.3%), Arya Shetty (33.3%) |
| Third Party API Documentation | Vineal Sunkara (33.3%), Nikash Rajeshbabu (33.3%), Arya Shetty (33.3%) |

## MENU GROUP

| Item | People |
|------|--------|
|      |        |

|                             |                                                                     |
|-----------------------------|---------------------------------------------------------------------|
| API Updates Table           | Joshua Menezes (40%) Keanu Melo Rojas (40%), Thomas O'Connell (20%) |
| Server-Side Routing Diagram | Joshua Menezes (100%)                                               |
| API Documentation/Use Cases | Joshua Menezes(40%), Keanu Melo Rojas(40%), Thomas O'Connell(20%)   |

### Customer Accounts

| Item                        | People                            |
|-----------------------------|-----------------------------------|
| API Updates Table           | Aman Patel (50%), Ethan Sie(50%)  |
| Server-Side Routing Diagram | Aman Patel (50%), Ethan Sie (50%) |
| API Documentation/Use Cases | Anna Yeakel (100%)                |

## Report 4 Contributions Table

| Report 4 Contributions                       |                                                                               |
|----------------------------------------------|-------------------------------------------------------------------------------|
| Groups                                       | Percentage/Part                                                               |
| Map/Driver (For all Updating Reports)        | Brandon Cheng (33.33%), Ji Wu (33.33%), Damon Lin (33.33%)                    |
| Pizza Pipeline (For all Updating Reports)    | Arya Shetty (33.33%), Vineal Sunkara (33.33%), Nikash Rajeshbabu (33.33%)     |
| Menu and Items (For all Updating Reports)    | Keanu Melo Rojas (33.33%), Joshua Menezes (33.33%), Thomas O'Connell (33.33%) |
| Customer/Accounts (For all Updating Reports) | Ethan Sie (33.33%), Anna Yeakel (33.33%), Aman Patel (33.33%)                 |

|                   | Weighted logarithmic points |         |                         |
|-------------------|-----------------------------|---------|-------------------------|
|                   | LOC                         | reports | Average of LOC, Reports |
| map/driver        | 125.19%                     | 128.74% | 126.97%                 |
| brandon           | 172.41%                     | 143.75% | 158.08%                 |
| ji                | 64.53%                      | 128.96% | 96.75%                  |
| damon             | 138.64%                     | 113.50% | 126.07%                 |
| pizza pipeline    | 117.95%                     | 110.83% | 114.39%                 |
| arya              | 134.75%                     | 112.16% | 123.45%                 |
| vineal            | 135.91%                     | 120.09% | 128.00%                 |
| nikash            | 83.19%                      | 100.23% | 91.71%                  |
| menu              | 86.13%                      | 87.39%  | 86.76%                  |
| josh              | 99.61%                      | 122.23% | 110.92%                 |
| keanu             | 106.11%                     | 78.77%  | 92.44%                  |
| thomas            | 52.67%                      | 61.17%  | 56.92%                  |
| customer/accounts | 70.72%                      | 73.05%  | 71.89%                  |
| anna              | 99.44%                      | 66.63%  | 83.03%                  |
| ethan             | 70.48%                      | 70.93%  | 70.71%                  |
| aman              | 42.26%                      | 81.58%  | 61.92%                  |