# Group 47

Team Member Names:

1. Jason Roberts
2. James Weber
3. Garrett Goodwin
4. Ryan Radtke
5. Johnny Duong

## Use Case Diagram and Use Cases:

| **Login** | |
| --- | --- |
| Actors | User, System Database |
| Description | User navigates to homepage and inputs login data (user name/password). The information is then sent to the user database and user status is verified. Then a login object with user ID and session is returned. |
| Data | User username and password |
| Stimulus | User input on login page |
| Response | Success or failed login attempt followed by appropriate redirect if necessary. |
| Comments | The user must have correct credentials to access their account and be able to navigate to the relevant pages for their use case. |

| **Chef Order Page** | |
| --- | --- |
| Actors | Chef, System Database |
| Description | Chef navigates to order page to review current orders that need to be prepared and processed. From here the chef can manipulate orders, complete them, or cancel them as required. |
| Data | List of current unfulfilled orders. |
| Stimulus | Chef login is automatically redirected to this page. |
| Response | Database is queried for orders which can be altered from this page. |
| Comments | Chef will have the ability to lookup orders from the database, any canceled order is deleted from the database. |

| **Order Processing** | |
| --- | --- |
| Actors | Order Processor, Database |
| Description | Order Processor verifies the order and payment from the customer and inputs it into the database to be prepared by the chef. |
| Data | Database order information and payment. |
| Stimulus | Order processor receives orders online from the customer and then inputs them into the system. |
| Response | System database is appended with a new order and updates the relevant users. |
| Comments | The order processor will have to parse orders from the customer to verify that they are made correctly (including payment) before adding them to the database for the chef to process. |

| **Pizza Order** | |
| --- | --- |
| Actors | Customer, Database, Order processor |
| Description | Customer inputs order in the app and then the order is verified by the order processor, which is then input into the database. |
| Data | Pizza type, pizza options, |
| Stimulus | Customer inputs order into system |
| Response | Order is approved/finalized and sent to the order processor to be input into database |
| Comments | The order processor acts as a quality assurance step here to make sure the client does not input any erroneous information into the database by accident. |

| **Checkout** | |
| --- | --- |
| Actors | Customer, Order Processor, Database |
| Description | Customer inputs payment information which is then verified by the order processor and the order is finalized and sent to the database. |
| Data | Customer payment information |
| Stimulus | Customer confirms order and inputs payment information |
| Response | System either accepts or denies payment and then redirects customer to order status page or to retry inputting order |
| Comments | If customer payment information is wrong the order should be preserved so the customer can retry payment information. |

| **Order Tracking** | |
| --- | --- |
| Actors | Order Tracking Page, Database, Chef |
| Description | As the chef prepares the order and updates the database this information is relayed in real time to the order tracking page for the customer to preview. |
| Data | Pizza status |
| Stimulus | Customer viewing their order in the order status page and chef updating the order from the chef order page |
| Response | Order Tracking Page is updates in real time as the pizza goes from being prepared to cooking to ready for pickup |
| Comments | Updating this page for the customer will be crucial for the chef to complete so that this page is accurate. |

| **Account Creation** | |
| --- | --- |
| Actors | User, Database |
| Description | User navigates to home page and clicks on account creation to create account. |
| Data | Account name, password, payment information, phone number, email |
| Stimulus | User clicks create account and inputs credentials. |
| Response | Database verifies credentials are correct and adds user into system. |
| Comments | Admin has access to accounts and the ability to change account types (chef, order processor). |

| **Administration** | |
| --- | --- |
| Actors | Admin, Database |
| Description | Admin has access to accounts and can change accounts and view orders both current and previously fulfilled. |
| Data | Account information, order information |
| Stimulus | Admin queries database for relevant information |
| Response | Database provides relevant information and ability to modify information |
| Comments | Admin should have the ability to flag certain accounts as chef or order processor as well review current and previous orders and change them if needed. |

## 

## Object Identification and CRC

**Use Cases:**

**Login:**

**Objects:**

User, Database, Session, UI

**Chef Order Page:**

**Objects:**

User (Chef), Database, Session, UI

**Order Processing:**

**Objects:**

User (Order Processor), Database, Session, UI

**Administration:**

**Objects:**

User (Admin), Database, Session, UI

**Pizza Order:**

**Objects:**

User (Customer, Order Processor), Database, Session, UI

**Checkout:**

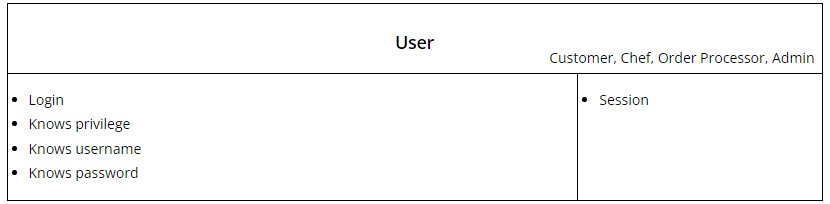
**Objects:**

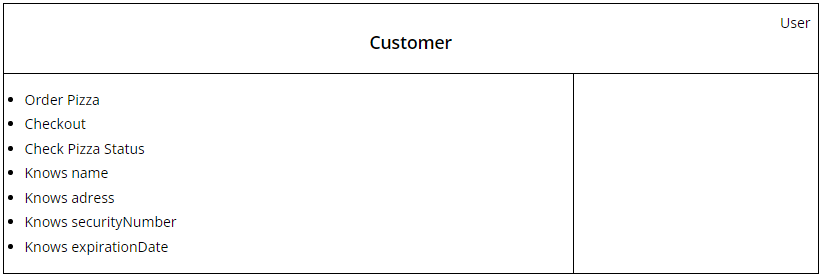
User (Customer, Order Processor), Database, Session, UI

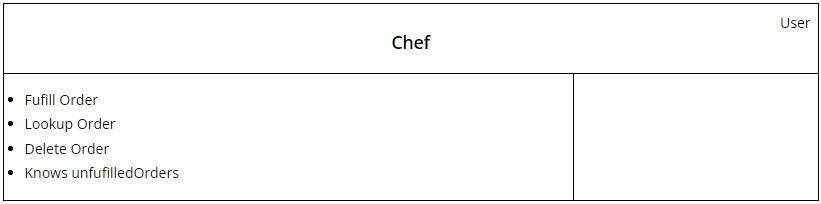
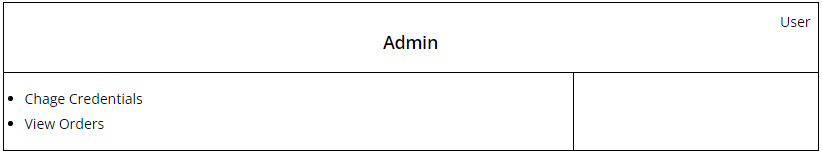
**Order Tracking:**

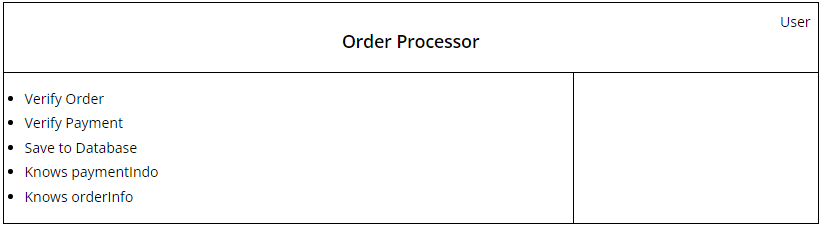
**Objects:**

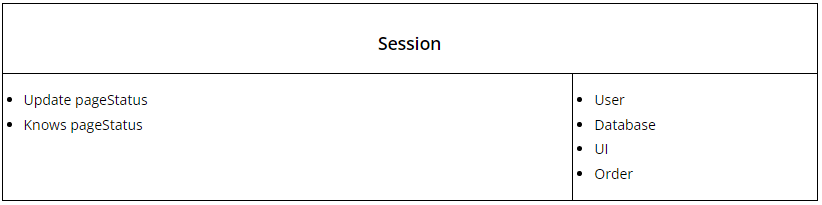
User (Chef), Database, Session, UI

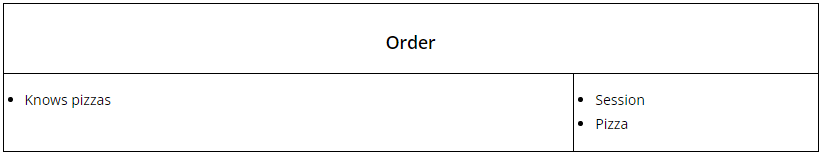


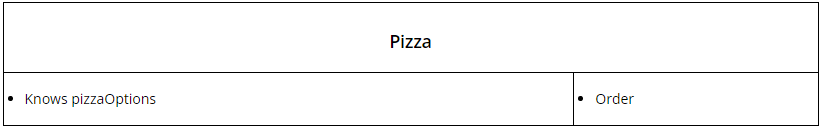


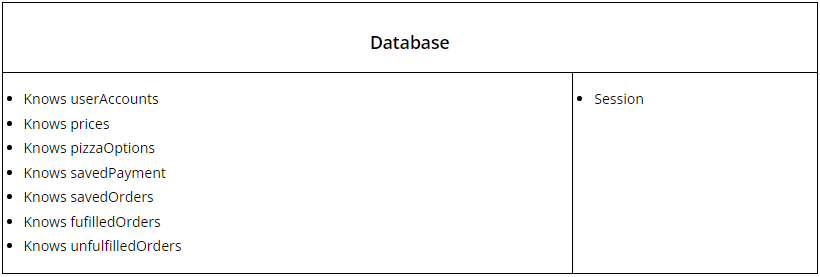


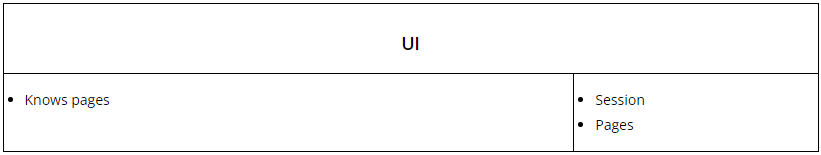


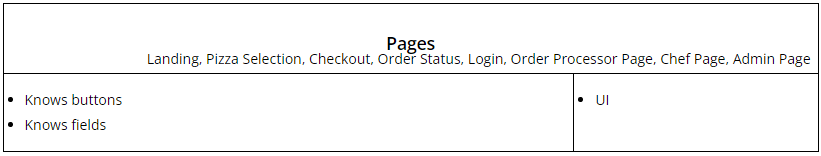






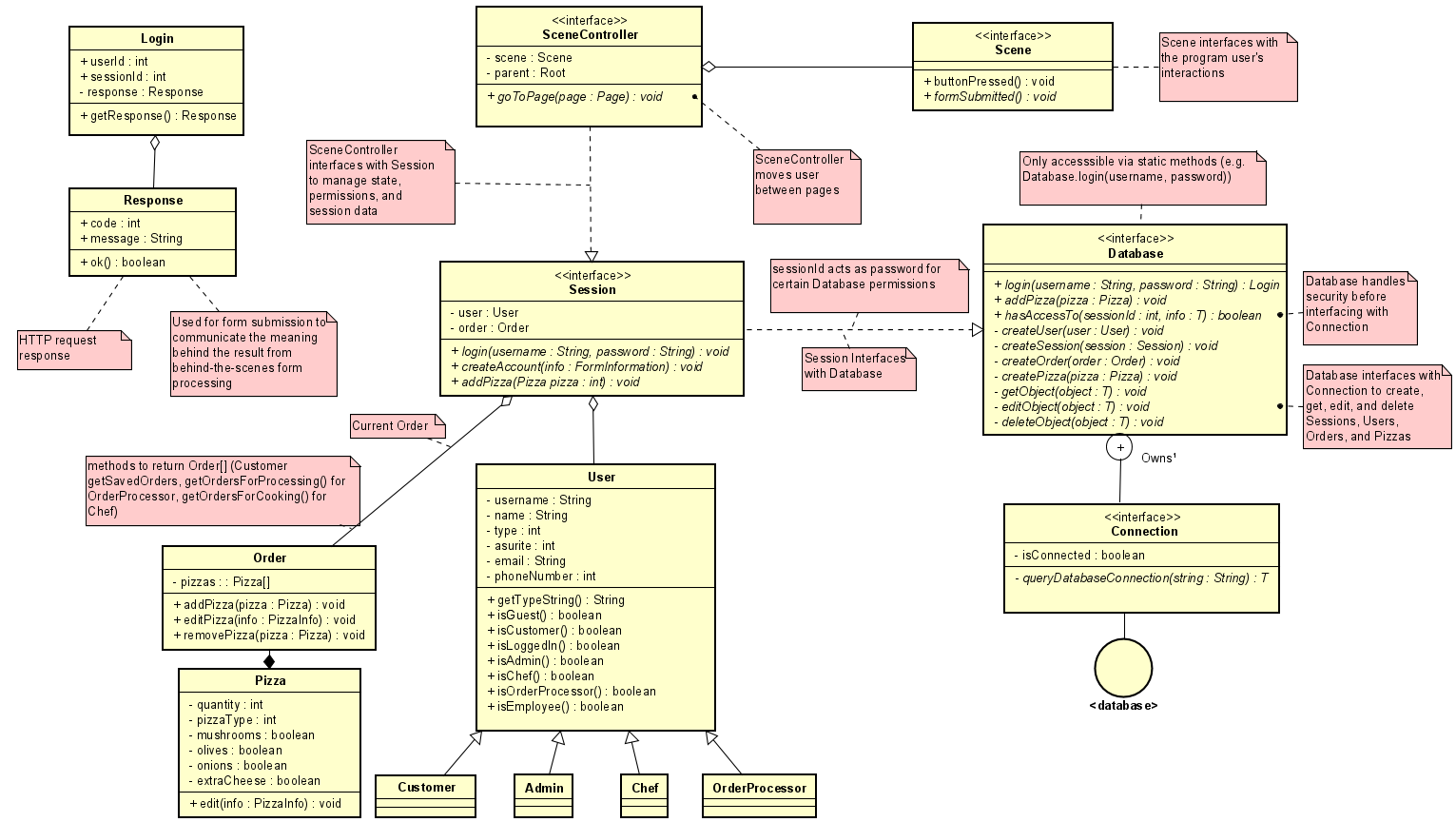




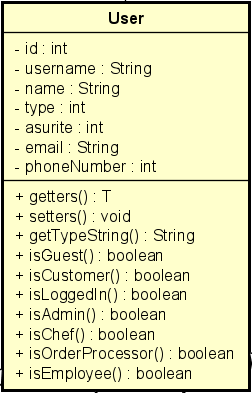


## 

## Initial Class Diagram and class descriptions



1 “Owns” relationship describes a nested class, as defined in IBM Rational Software Architect Standard Edition, Version 7.5.5.5



Users are the class that would basically show the people and their roles. The other roles will inherit the user class with different “type” values. To find out whether or not the user logging in is a customer, admin, chef, or order processor. The User class includes the id, username, password, name, type, asurite, email, and phoneNumber as attributes. For the operations, there are getters(), setters(), getTypeString(), isGuest(), isCustomer(), isAdmin(), isChef(), isOrderProcessor(), isEmployee(), and isLoggedIn(). These attributes and operations will be inherited by the subclasses, which the operations in the User class could show what user the user may be when logging in. This would be part of the system to show the inheritance of the User class to the other classes.



Customers are the ones that will be making the orders of the pizzas. The information of the customers would be their id, username, password, name, type, asurite, email, and phoneNumber, which is inherited from the User class. Guests would also be considered customers. Additionally, for payment of the order, the Customer class would be able to check the orders before finalizing their orders and payment through the checkout page. Customers are a part of the system to show the Customer class that inherited things from the User class with the addition of attributes and operations.



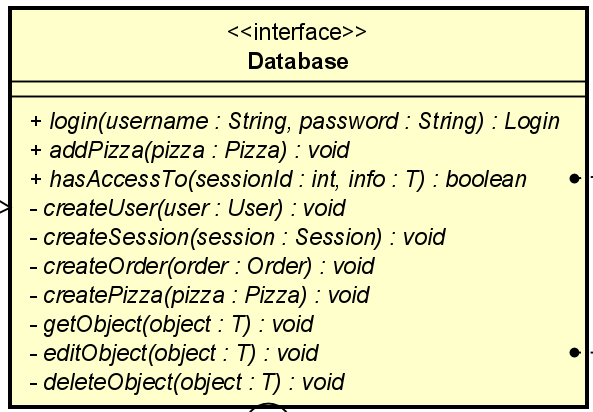
The OrderProcessor class is the class that will confirm the orders of the pizza and the price of it. This information will then be placed in the database. The order processors also inherit from the User class with a different “type” integer to show that the user is the order processor, which allows the order processor to access the Order Processor Page in the UI interface. Order processors are a part of the system since they inherit from the User class to have access as an order processor and would input the information of the orders into the database.



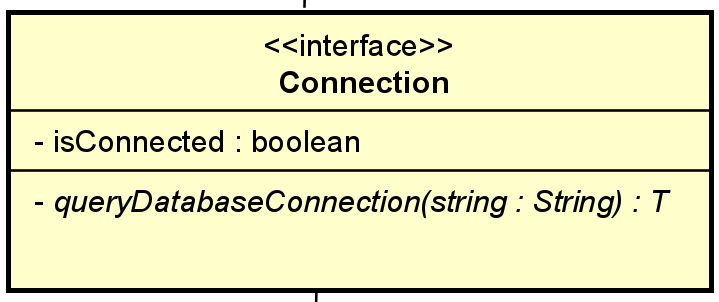
The Chef class is the one that will take the orders from the pizzas and make the pizzas. The chefs are able to change the orders, complete them, or cancel the orders if required to. Like the other subclasses, the chefs also inherit from the User class but can access the Chef Page in the UI interface to look at the orders that customers have made. The page also allows the chefs to change, cancel, or complete the order. This is a part of the system, for the chef inherits from the User class with additional access because of the different “type” integer from the User class.



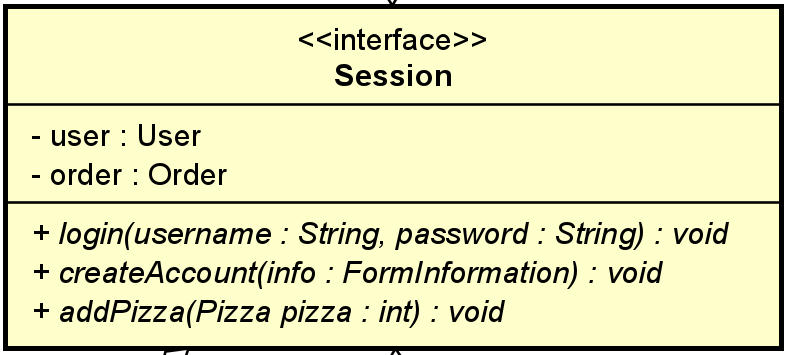
The admins can change the accounts and look at the orders that were fulfilled previously and currently. They have the ability to modify the information of the orders and flag the accounts like chefs and order processors. Since the Admin class also inherits from the User class, their “type” would show that they are part of the Admin class and have additional options as well, which would be the access to the Check Credentials Page to modify the information of orders and flag accounts. This is a part of the system, for it inherits from the User class with a different “type” integer to show they are a part of the Admin class.



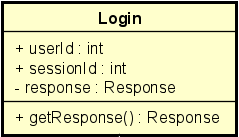
The Database class interfaces between the Session class and its nested class - Connection - to save, delete, read, and change data about user accounts, sessions, orders, and pizzas. It handles security and logic before interfacing with the Connection class to manipulate data.



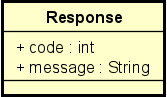
The Connection class is nested inside the Database class, and performs the actual interfacing between the program and the actual database. It saves, deletes, reads, and changes data when it is asked to do so by the Database class.



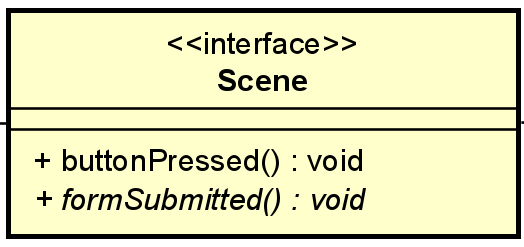
Session provides information that is relevant to the User to the SceneController class. It retrieves this information by interfacing with the Database class. The Session receives its sessionId from the database upon a successful login, and the sessionId is used as a key to retrieve permitted information from the Database.



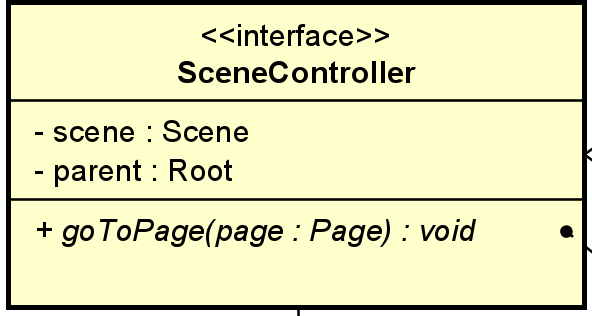
The Login class is where the users will use their usernames and passwords to log in. The usernames and passwords need to be valid, and valid username and password combinations are found in the database. This class includes the userId, sessionId, and response as attributes, and getResponse() as an operation. There would be a prompt from the Response class to show whether or not the user has successfully logged in the application. This would be a part of the system since it allows the users to get certain access to the app.



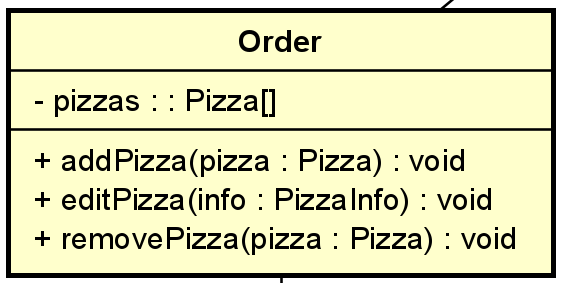
The Response class is where there is a prompt from the attempt of the user’s login. It is similar to the HTTP request response, which it would show if it was a success or not found. This class would include the code and message attributes, and the response class is a part of the Login class (it is an aggregation). This is in the system, for it generates the responses when the user attempts to log in.



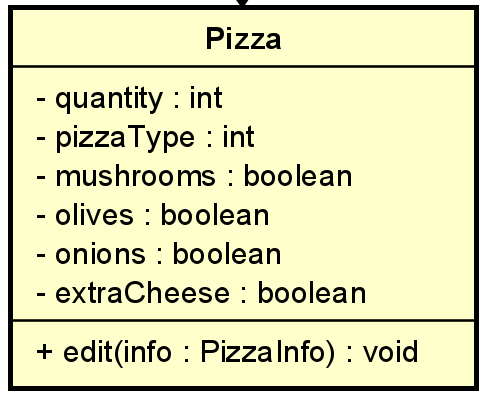
The Scene is where users will see the application and interact with it for their session. The scenes include many pages such as the Landing Page, Login Page, New Account Page, Pizza Selection Page, Checkout Page, Order Status Page, Chef Page, Order Processor Page, and Check Credentials Page (Admin Page). This will have operations of buttonPressed() and formSubmitted(), which these would be from the pages that are found in the UI and scenes for the interactions of the user. This would be in the system as it is a part of the sceneController of the interface.



The SceneController part of the interface would be for the use of being able to navigate the pages in the application. This would have attributes of scene and parent and the operation of goToPage(page : Page). The SceneController part would interface with the Session to manage the state, permission, and session data. This would be a part of the system, for it is to navigate the scenes in the interface.



The Order class is the class where it has the list of pizzas that the customer has ordered during the session. The class consists of the attribute pizza, which is a list of the pizzas that the customer ordered. If there are no pizzas in the Pizza class, then the Order class will not exist, for the Order class has a relationship of composition with the Pizza class; the Order class cannot stand by itself. The Order class also includes operations of addPizza(pizza : Pizza), editPizza(info : PizzaInfo), and removePizza(pizza : Pizza). These operations allow the customers to change the order as they please. This is in the system because the Order class is a part of the Session interface but cannot exist without the Pizza class.



The Pizza class is the class that consists of the pizza that the customer has ordered. The pizza would have attributes of quantity, pizzaTupe, mushrooms, olives, onions, and extraCheese. Customers can include as much pizza toppings as they want, which would be the mushrooms, olives, onions, and extraCheese. The Pizza class is what would create the Order class, for the Order class needs the pizzas from the Pizza class to create the order for the session. The operation of the Pizza class is edit(info : PizzaInfo), which allows the customers to edit the pizza they create. This would be a part of the system, for the Pizza class has a relationship with the Order class, which would be for the sessions of customers to create their orders.

## Test Plan for Functional Testing

## -Test account creation, make sure all accounts created are validated and any errors are appropriately handled (make input format agnostic and test defensive coding)

-Test correct account creation, ensure accounts are properly added to the database

-Test account information security, functions have appropriate privacy/accessor settings

-Test order handling from multiple accounts, warning message should pop up alerting to any changes that have occurred within the last minute from a different account

-Test order cancellation, ensure order is properly removed from the database

-Test payment authentication

-Test Order completion, customer should be alerted, and order should go into archive

-Test account login, incorrect and correct information should deliver the user to the respective page

-Test order creation, adding pizza options should update price properly and reflect in the final order

-Test administrator account authority, administrator should be able to adjust other accounts and review orders and query the database

-Test database limitations, make sure the database can handle high amounts of users

## 

## Evidence of Working Prototype Code

Our Working Prototype Video:

<https://youtu.be/AzetynfUwTY>