Use Case Description

The use cases for this system can be divided into three distinct situations based on the user interacting with the system.

1. The user is a customer and will be ordering food.
2. The user is a driver and will be picking up the food and delivering the customer's order.
3. The user is restaurant personnel and will be handling the preparation of the order.

Situation 1 - Customer:

The user opens the application and is met with a login screen. If the user has an existing account, they may enter their details and progress to the next screen; otherwise, the user will be taken to a page to create an account to log in with.

Once logged in, the user progresses to a screen that will display a list of restaurants that they may order from. The user then selects a restaurant and is forwarded to a screen that lists the menu items they may order from the restaurant. The user may select some number of menu items from this list, which will be added to a cart. They may access their cart to review what they have ordered and remove items they no longer wish to receive.

When satisfied with their order, the user presses a button to progress to a payment screen where they enter their delivery and billing information. Upon confirmation, the user is forwarded to a screen that will display a live map of the driver who will pick up the food and a messenger to allow for direct messaging with the driver.

Upon the driver's arrival, the user is sent to a page that alerts them that the order has been delivered and contains a feedback form. The user is sent back to the restaurant list screen upon completion of the form, from which they may repeat the process if they wish.

Situation 2 - Delivery Driver:

The user opens the application and is met with a login screen. If the user has an existing account, they may enter their details and progress to the next screen; otherwise, the user will be taken to a page to create an account to log in with.

Once logged in, the user progresses to a delivery dashboard that displays a list of potential orders that they can take, in addition to account and payment information. Upon selecting an order, the user is forwarded to a screen that displays a map and multiple status buttons. The map displays the delivery address of the customer and the address of the restaurant they will be picking up the order from.

The status buttons are manually toggled by the driver to inform the customer of the current status of their order. When the driver picks up the order, they will select the "Picked-Up" status button. After the driver delivers the order to the address, they will select the "Delivered"

status button. When the order is delivered, the user is sent back to the main dashboard and may select another delivery if they wish.

Situation 3 - Restaurant Personnel:

The user opens the application and is met with a login screen. If the user has an existing account, they may enter their details and progress to the next screen; otherwise, the user will be taken to a page that will display contact information for the restaurant owner to contact to register an account.

Once logged in, the user is sent to a dashboard page that displays the restaurant's statistics, such as order history and orders currently being processed. This page will also allow the user to view account information and edit menu items.

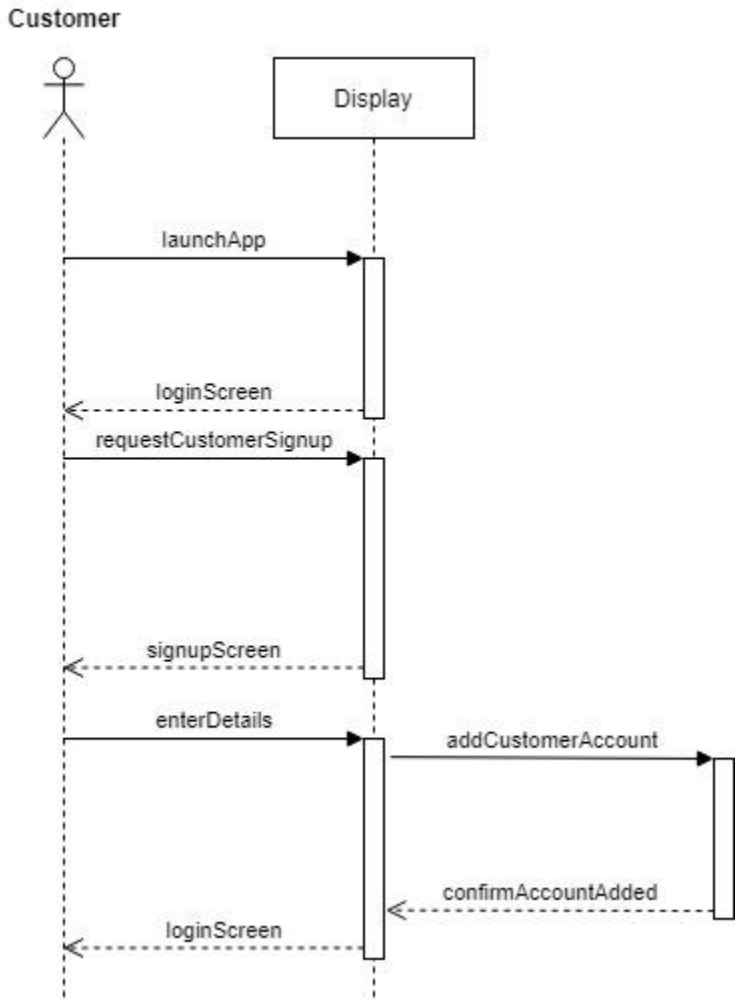
The user may select the cashier screen page to see orders that have been sent to their restaurant and update their status. They can confirm the order, indicating that the restaurant has received the order and will begin preparing it. This informs the customer and driver of the estimated time that the order will be available for pickup. When the order is ready, the user can then set the order status to "Ready for Pickup" indicating to the driver that the order is ready for them to pick up. Once the order has been picked up by the driver the order will be removed from the cashier screen and will be shown in the order history section of the restaurant dashboard.

Sequence Diagrams

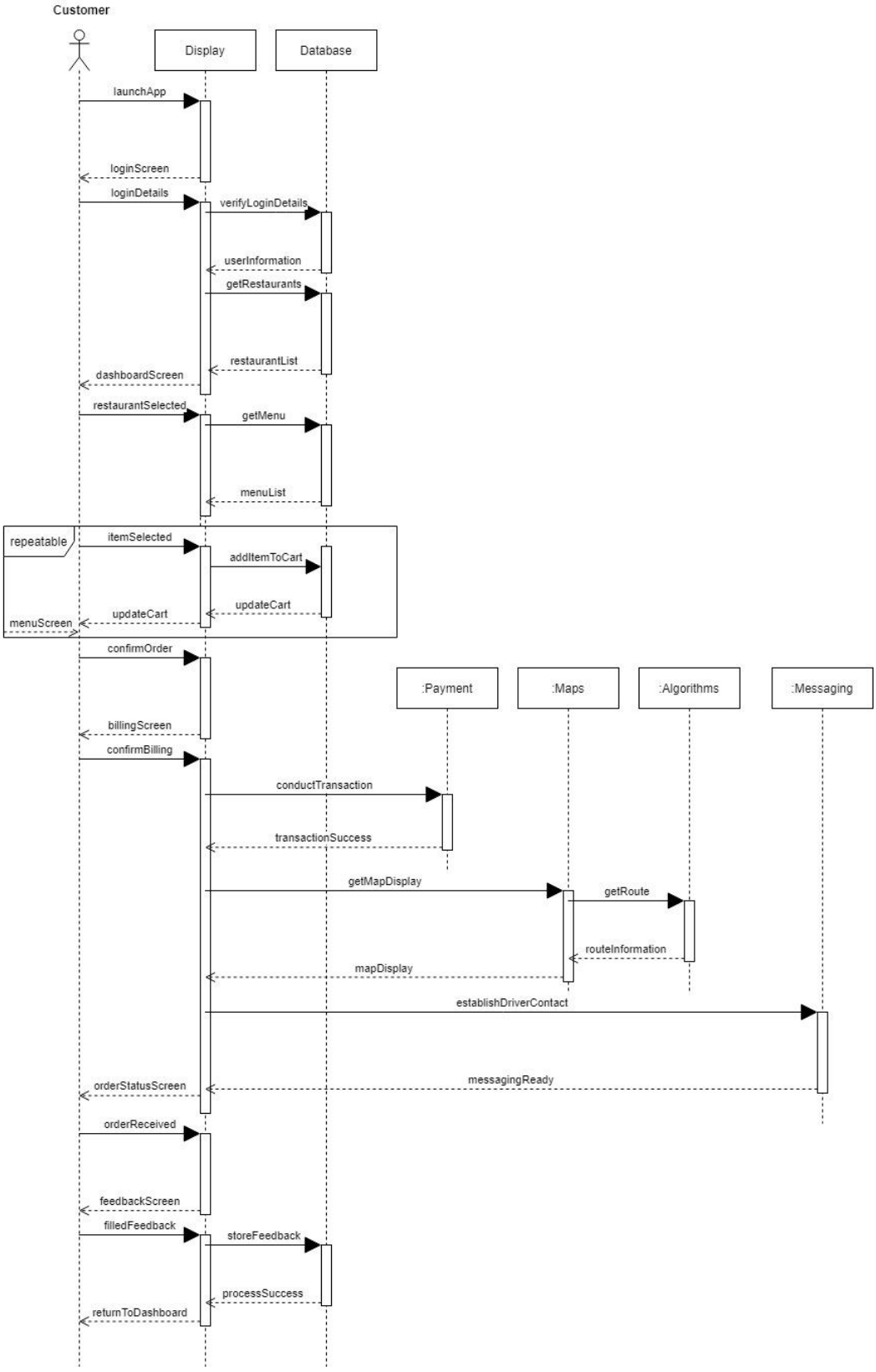
Sequence diagrams show the flow of events for the user of an application. As the YUMI application will have three different types of users, this section is broken down based on whether the user is the customer, driver, or restaurant employee.

Customer as User

The following sequence diagram shows the flow of events for a customer launching the application and creating an account.

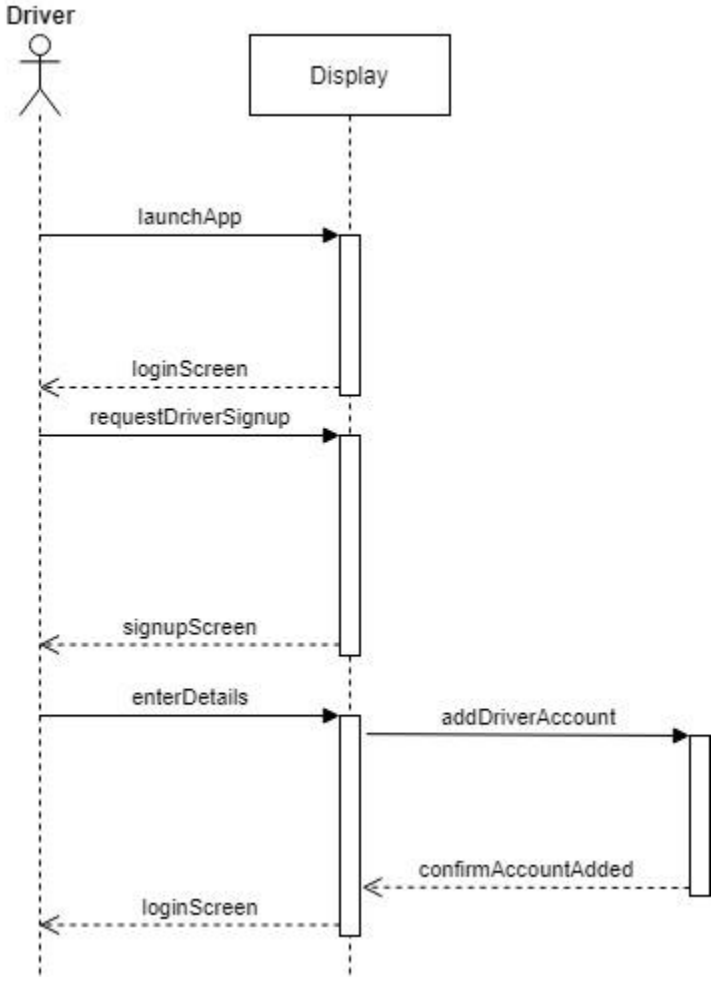


The following sequence diagram shows the flow of events for a customer launching the application and ordering food.

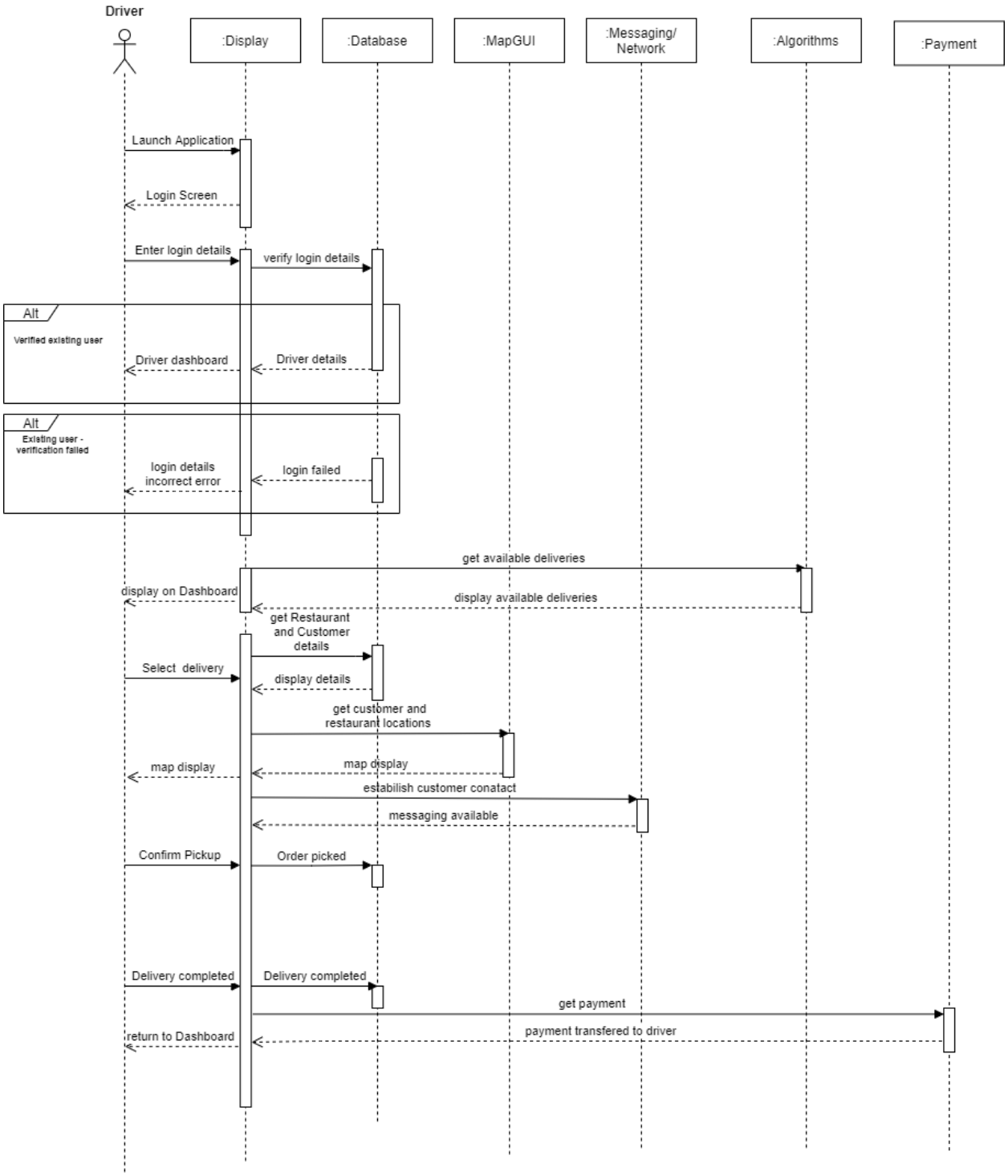


Driver as User

The following sequence diagram shows the flow of events for a driver launching the application and creating an account.

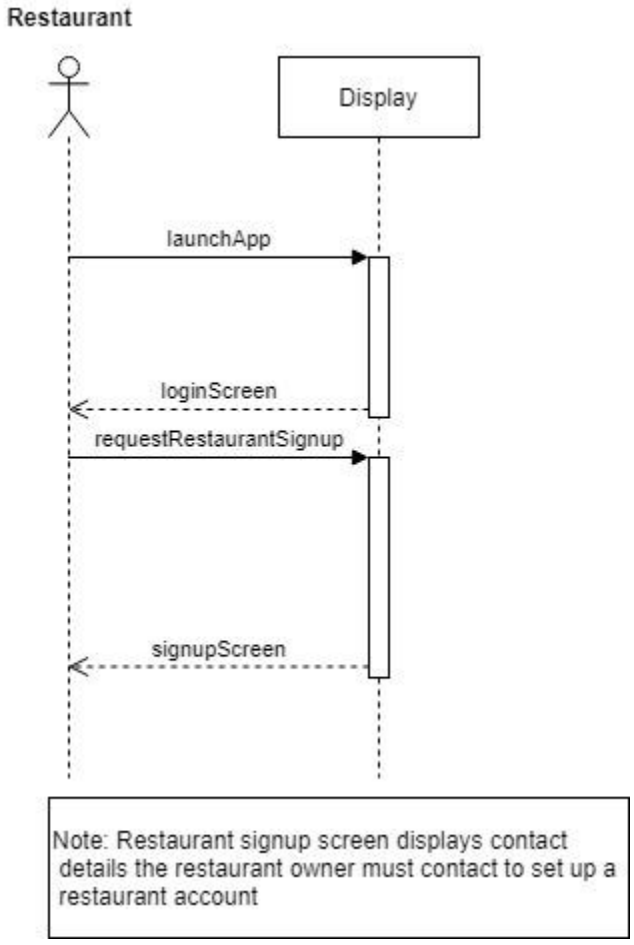


The following sequence diagram shows the flow of events for a driver launching the application, accepting, picking up and delivering an order.



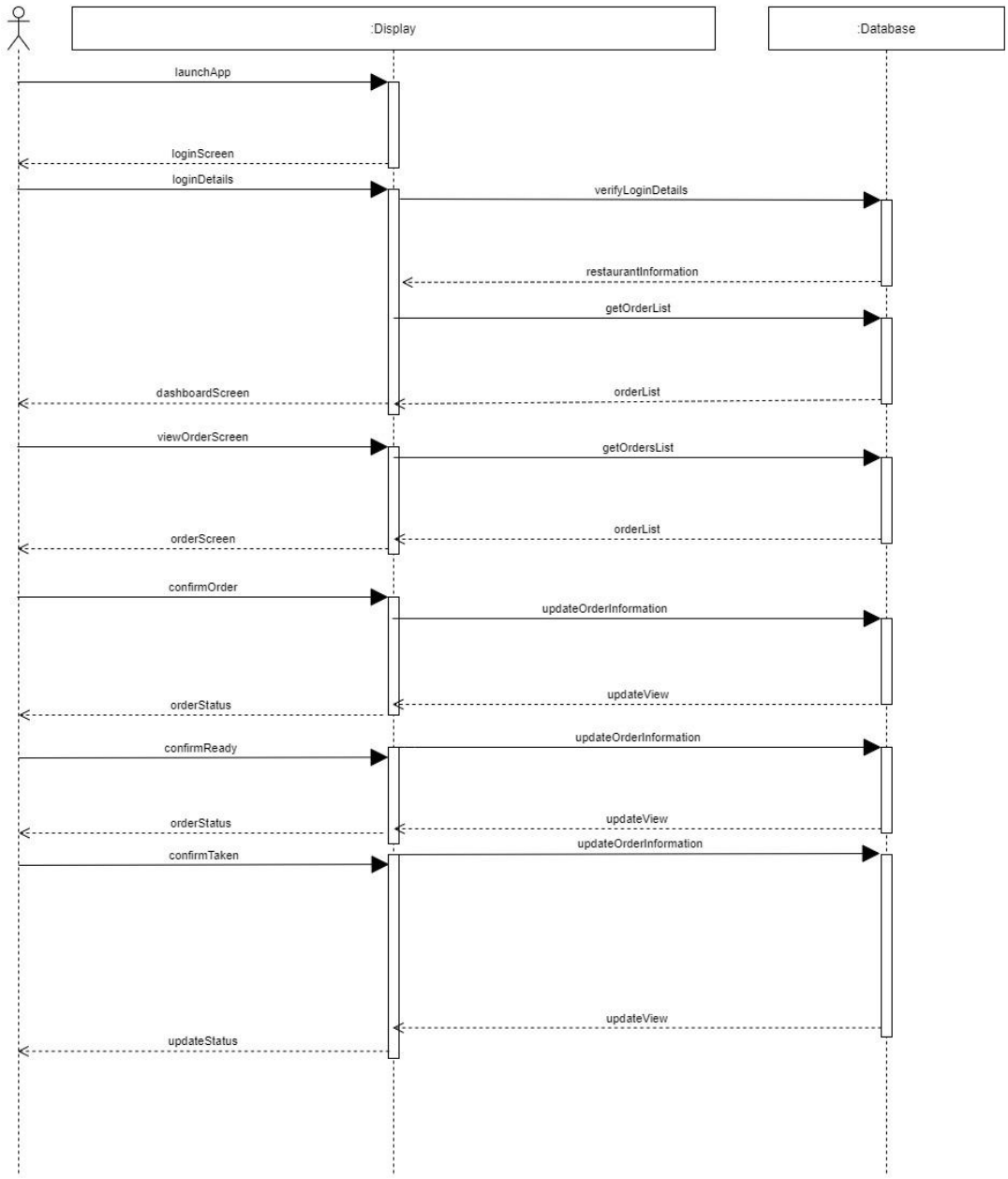
Restaurant as User

The following sequence diagram shows the flow of events for a driver launching the application and creating an account. Note that restaurant users must contact YUMI customer service to set up a restaurant account for the first iteration of the application.

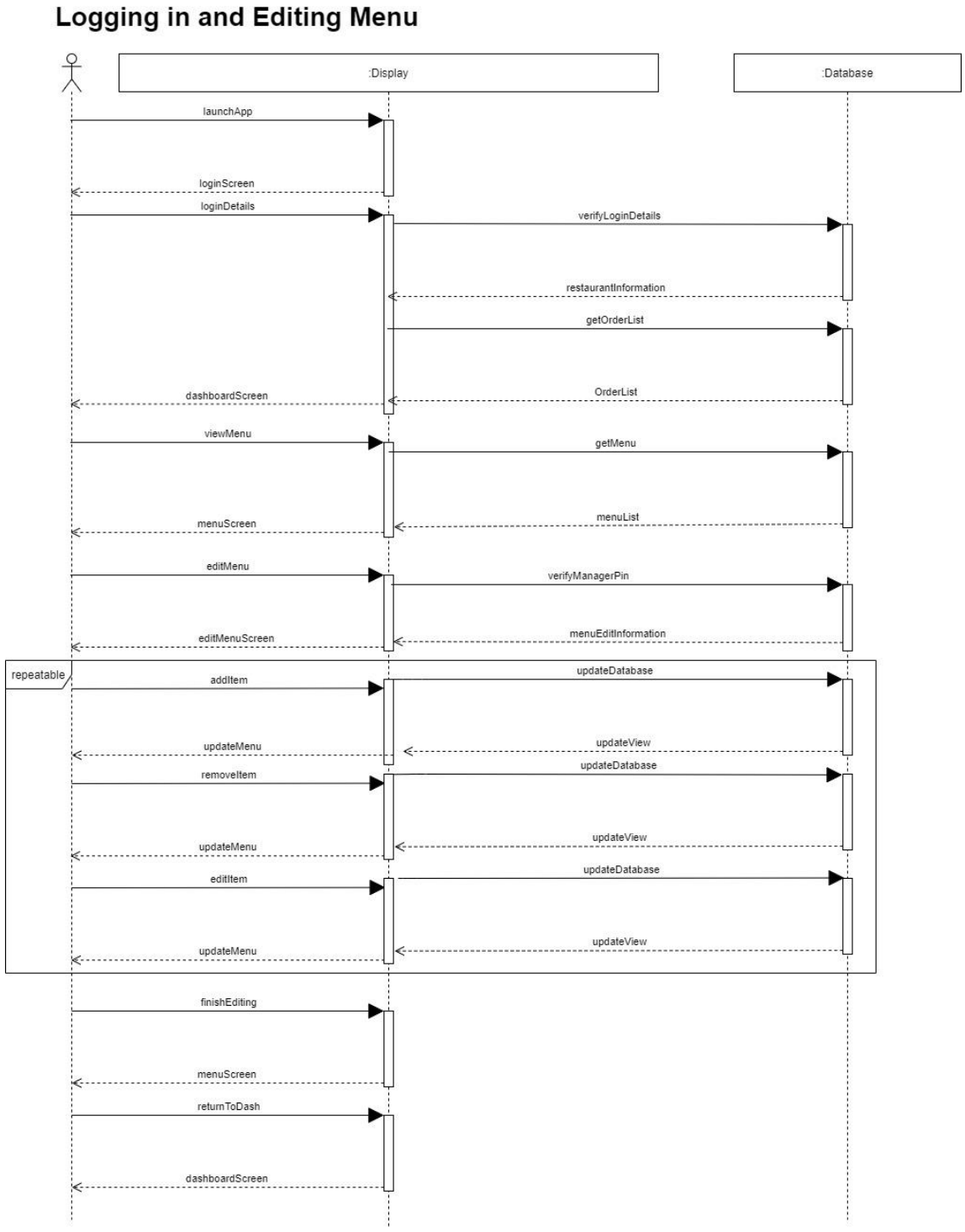


The following sequence diagram shows the flow of events for a restaurant worker launching the application and receiving an order.

Logging in and Confirming Orders



The following sequence diagram gives an idea of the flow of events for a restaurant worker launching the application and editing the restaurant menu.

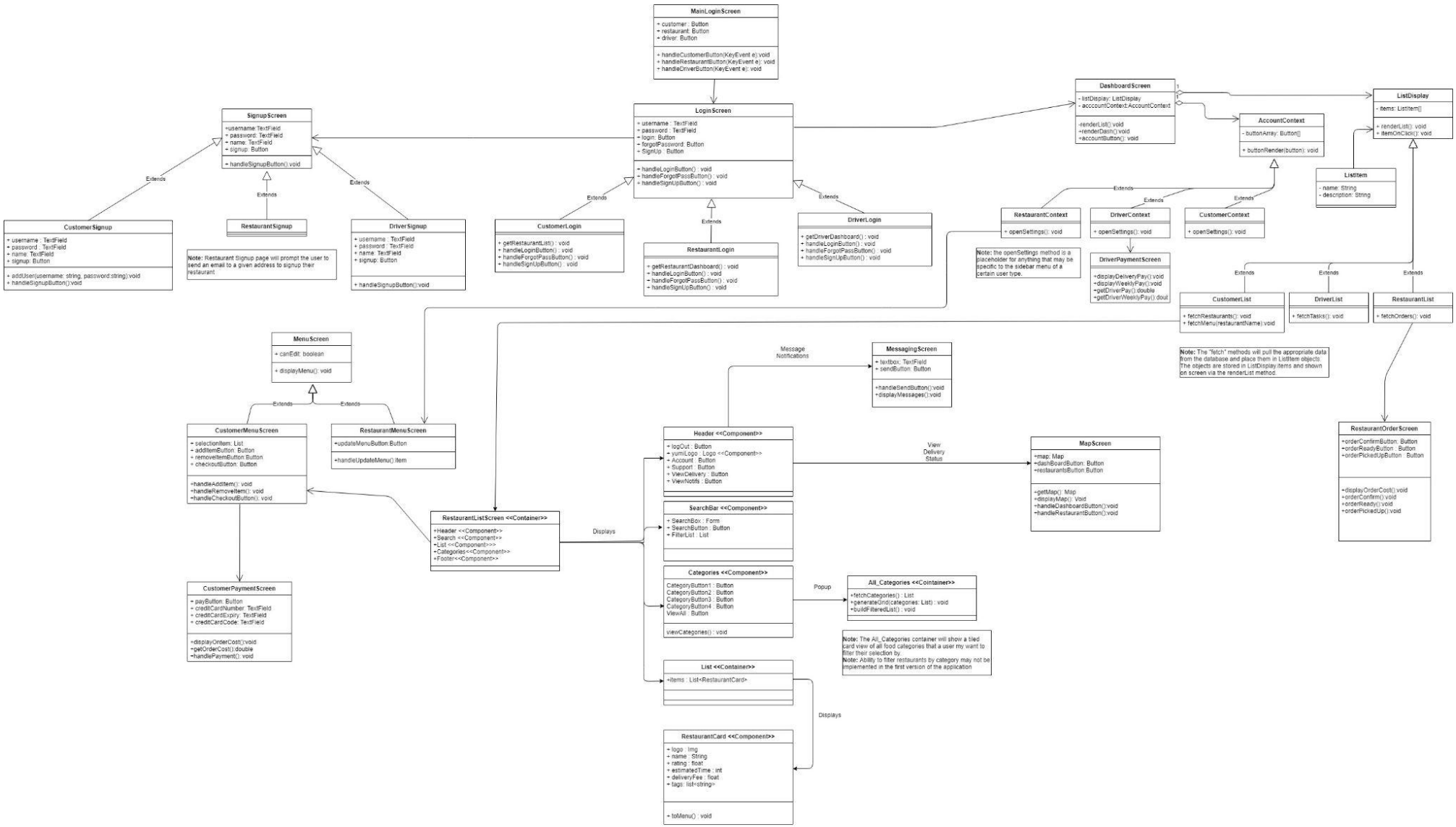


UML Diagram

GUI is a different type of programming in that it focuses on the layout and design of displays rather than the implementation of algorithms. As such, the GUI team has decided to organize the code based upon the open-close principle and decorator design pattern. There will be common types of screens that more than one type of user shares, such as the dashboard and the menu screens. To avoid rewriting the same code for each of these user screens, an abstract class will hold the template for a page and have child classes that inherit the base template but will modify what is displayed based upon the user type. For example, both the restaurant and the customer will have a menu page with a similar layout; however, the implementation for the restaurant will be slightly different than the customer as the restaurant will be able to edit their menu options. A common menu screen will be inherited by the restaurant menu screen and the customer menu screen allowing for the template of the menu screen to be reused. However, the functionality of the menu pages will be different as the customer will be able to add items to their order, and the restaurant will be able to edit their menu. In addition, this structure allows for a change in the functionality of the customer menu screen to not require changes to be made to the restaurant menu screen.

Aggregations are also used for the dashboard screen when displaying user account information. There should only be one user account associated with the dashboard screen, which is that of the user that is currently logged in. The same is true with the type of list displayed on the dashboard screen, only one list should be displayed, and it should be the one that matches the user type. For example, the driver should be able to view a list of orders available for them to accept and not the list of orders submitted to a restaurant.

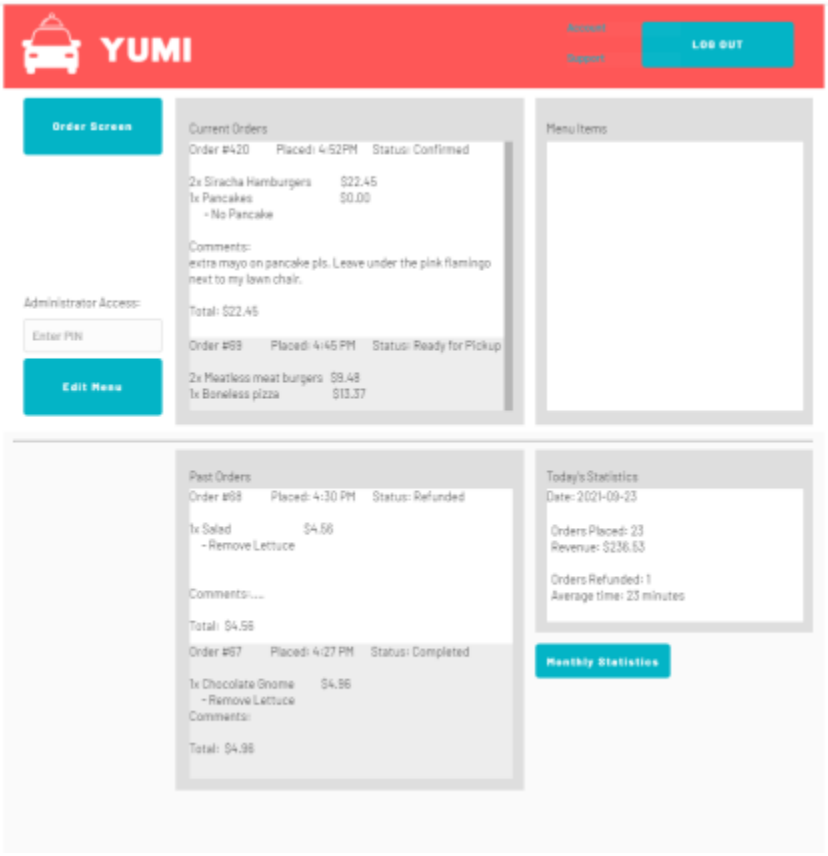
The UML diagram is shown on the following page. To view the text please zoom into the PDF or view from [GUI UML and Sequence Diagrams - diagrams.net]


MainLoginScreen
+ customer : Button
+ restaurant: Button
+ driver: Button
+ handleCustomerButton(KeyEvent e): void
+ handleRestaurantButton(KeyEvent e): void
+ handleDriverButton(KeyEvent e): void
LoginScreen
- username : TextField
- password : TextField
- login: Button
- forgotPassword: Button
- SignUp : Button
+ handleLoginButton() : void
+ handleForgotPassButton() : void
+ handleSignUpButton() : void
SignupScreen
+username:TextField
+ password: TextField
+ name: TextField
+ signup: Button
+ handleSignupButton():void
CustomerSignup
+ username : TextField
+ password : TextField
+ name: TextField
+ signup: Button
+ addUser(username: string, password:string):void
+ handleSignupButton():void
RestaurantSignup
Note: Restaurant Signup page will prompt the user to send an email to a given address to signup their restaurant
DriverSignup
+ username : TextField
+ password : TextField
+ name: TextField
+ signup: Button
+ handleSignupButton():void
CustomerLogin
+ getRestaurantList() : void
+ handleLoginButton() : void
+ handleForgotPassButton() : void
+ handleSignUpButton() : void
RestaurantLogin
+ getRestaurantDashboard() : void
+ handleLoginButton() : void
+ handleForgotPassButton() : void
+ handleSignUpButton() : void
DriverLogin
+ getDriverDashboard() : void
+ handleLoginButton() : void
+ handleForgotPassButton() : void
+ handleSignUpButton() : void
DashboardScreen
- listDisplay: ListDisplay
- accountContext:AccountContext
-renderList(): void
+renderDash():void
+accountButton(): void
ListDisplay
- items: ListItem[]
+ renderList(): void
+ itemOnClick(): void
AccountContext
- buttonArray: Button[]
+ buttonRender(button): void
ListItem
- name: String
- description: String
RestaurantContext
+ openSettings(): void
DriverContext
+ openSettings(): void
CustomerContext
+ openSettings(): void
Note: the openSettings method is a placeholder for anything that may be specific to the sidebar menu of a certain user type.
DriverPaymentScreen
+displayDeliveryPay():void
+displayWeeklyPay():void
+getDriverPay():double
+getDriverWeeklyPay():doul
CustomerList
+ fetchRestaurants(): void
+ fetchMenu(restaurantName):void
DriverList
+ fetchTasks(): void
RestaurantList
+ fetchOrders(): void
Note: The "fetch" methods will pull the appropriate data from the database and place them in ListItem objects. The objects are stored in ListDisplay.items and shown on screen via the renderList method.
MenuScreen
+ canEdit: boolean
+ displayMenu(): void
CustomerMenuScreen
+ selectionItem: List
+ addItemButton: Button
+ removeItemButton:Button
+ checkoutButton: Button
+handleAddItem(): void
+handleRemoveItem(): void
+handleCheckoutButton(): void
RestaurantMenuScreen
+updateMenuButton:Button
+handleUpdateMenu():item
CustomerPaymentScreen
+ payButton: Button
+ creditCardNumber: TextField
+ creditCardExpiry: TextField
+ creditCardCode: TextField
+displayOrderCost():void
+getOrderCost():double
+handlePayment(): void
RestaurantListScreen <<Container>>
+Header <<Component>>
+Search <<Component>>
+List <<Component>>
+Categories<<Component>>
+Footer<<Component>>
Displays
Message Notifications
MessagingScreen
+ textbox: TextField
+ sendButton: Button
+handleSendButton():void
+displayMessages():void
Header <<Component>>
+ logOut : Button
+ yumLogo : Logo <<Component>>
+ Account : Button
+ Support : Button
+ ViewDelivery : Button
+ ViewNotifs : Button
View Delivery Status
MapScreen
+map: Map
+dashBoardButton: Button
+restaurantsButton:Button
+getMap(): Map
+displayMap(): Void
+handleDashboardButton():void
+handleRestaurantButton():void
SearchBar <<Component>>
+ SearchBox : Form
+ SearchButton : Button
+ FilterList : List
Categories <<Component>>
CategoryButton1 : Button
CategoryButton2 : Button
CategoryButton3 : Button
CategoryButton4 : Button
ViewAll : Button
viewCategories() : void
Popup
All_Categories <<Cointainer>>
+fetchCategories() : List
+generateGrid(categories: List) : void
+buildFilteredList() : void
Note: The All_Categories container will show a tiled card view of all food categories that a user my want to filter their selection by.
Note: Ability to filter restaurants by category may not be implemented in the first version of the application
List <<Container>>
+Items : List<RestaurantCard>
Displays
RestaurantCard <<Component>>
+ logo : Img
+ name: String
+ rating : float
+ estimatedTime : int
+ deliveryFee : float
+ tags: list<string>
+ toMenu() : void
RestaurantOrderScreen
+orderConfirmButton: Button
+orderReadyButton : Button
+orderPickedUpButton : Button
+displayOrderCost():void
+orderConfirm():void
+orderReady():void
+orderPickedUp():void
Extends

Potential Screen Designs

Potential screen designs for the restaurant dashboard give an idea of the style and layout of the application. Note these screenshots are early prototypes and are subject to change as the design progresses.

Restaurant Dashboard Design:



Restaurant Cashier Screen Design: