Software Design Specification

For

Yummy

Submitted by

Intuitive

|  |  |
| --- | --- |
| **Instructor:** | Dr. Gregory Hislop |
| **Team Members:** | Nik Bournelis, Mark Canda, Wen Fei Hao, Zeng Sheng Liu, Edgar Paz |
| **Cycle:** | 2 |
| **Date Submitted:** | 02-21-2013 |

Document template copyright 2005-2011, Gregory W. Hislop. Use permitted under Creative Commons license CC-BY-NC-SA. See http://creativecommons.org/licenses/by-nc-sa/3.0/.

Grading Rubric - Design Specification

This rubric outlines the grading criteria for this document. Note that the criteria represent a plan for grading. Change is possible, especially given the dynamic nature of this course. Any change will be applied consistently for the entire class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Achievement** | **Minimal** | **Exemplary** | **Pts** | **Score** |
| **Content** | Section(s) missing, not useful, inconsistent, or wrong. | Provides all relevant information correctly and with appropriate detail |  |  |
| Introduction |  |  | 10 |  |
| Architectural Description |  |  | 10 |  |
| Interface Description |  |  | 10 |  |
| Detailed Design |  |  | 40 |  |
| **Grammar and Spelling** | Many serious mistakes in grammar or spelling | Grammar, punctuation, and spelling all correct | 10 |  |
| **Expression** | Hard to follow or poor word choices | Clear and concise. A pleasure to read | 10 |  |
| **Tone** | Tone not appropriate for technical writing | Tone is consistently professional |  |  |
| **Organization** | Information difficult to locate | All information is easy to find and important points stand out | 10 |  |
| **Layout** | Layout is inconsistent, visually distracting, or hinders use | Layout is attractive, consistent, and helps guide the reader |  |  |
| **Late Submission** |  |  |  |  |
| **Total** |  |  | 100 | **~88** |

Table of Contents

[1 Introduction 5](#_Toc349216937)

[1.1 Scope 5](#_Toc349216938)

[1.2 Definitions, Acronyms, and Abbreviations 5](#_Toc349216939)

[2 Architectural Description 5](#_Toc349216940)

[3 Interface Description 8](#_Toc349216941)

[3.1 User Interface 8](#_Toc349216942)

[3.2 Data Interface 8](#_Toc349216943)

[3.3 Programming Interface 8](#_Toc349216944)

[4 Detailed Design 9](#_Toc349216945)

Table of Contributions

The table below identifies contributors to various sections of this document.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Section** | **Writing** | **Editing** |
|  | **Entire Document** | All | All |
| 1 | Introduction | zl |  |
| 1.1 | Scope | zl |  |
| 1.2 | Definitions, Acronyms, and Abbreviations | zl |  |
| 2 | Architectural Description | wh, ep, nb | wh, zl, mc |
| 3 | Interface Description |  |  |
| 3.1 | User Interface | zl, ep, nb | wh |
| 3.2 | Data Interface | zl | wh |
| 3.3 | Programming Interface | N/A | N/A |
| 4 | Detailed Design | zl, wh, mc, ep, nb | wh, mc, zl |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Scope

Yummy is an Android application aimed to help vendors increase facility with which customers can place orders. Yummy will allow customers to view vendors’ menus and place their order within the app via integration with PayPal and Drexel’s Dragon Dollar service. It will also enable vendors to view sales data and analysis based on purchases made via Yummy. An internet connection is required for the use of the app.

## Definitions, Acronyms, and Abbreviations

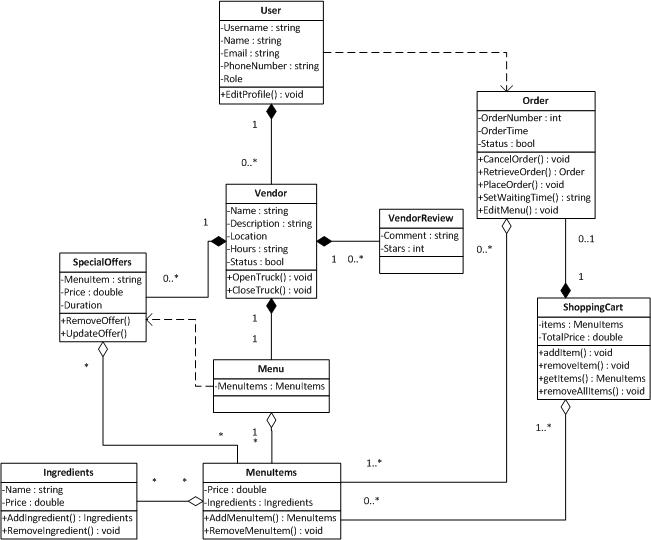
“System” or “Yummy” will refer to either the web application backend or the Android application frontend. If a distinction is necessary, the terms “backend” or “frontend” will be used.

# Architectural Description

**2.1 Classes**

The object classes are described below. Figure 1 shows these classes within the Yummy system and their relationships.

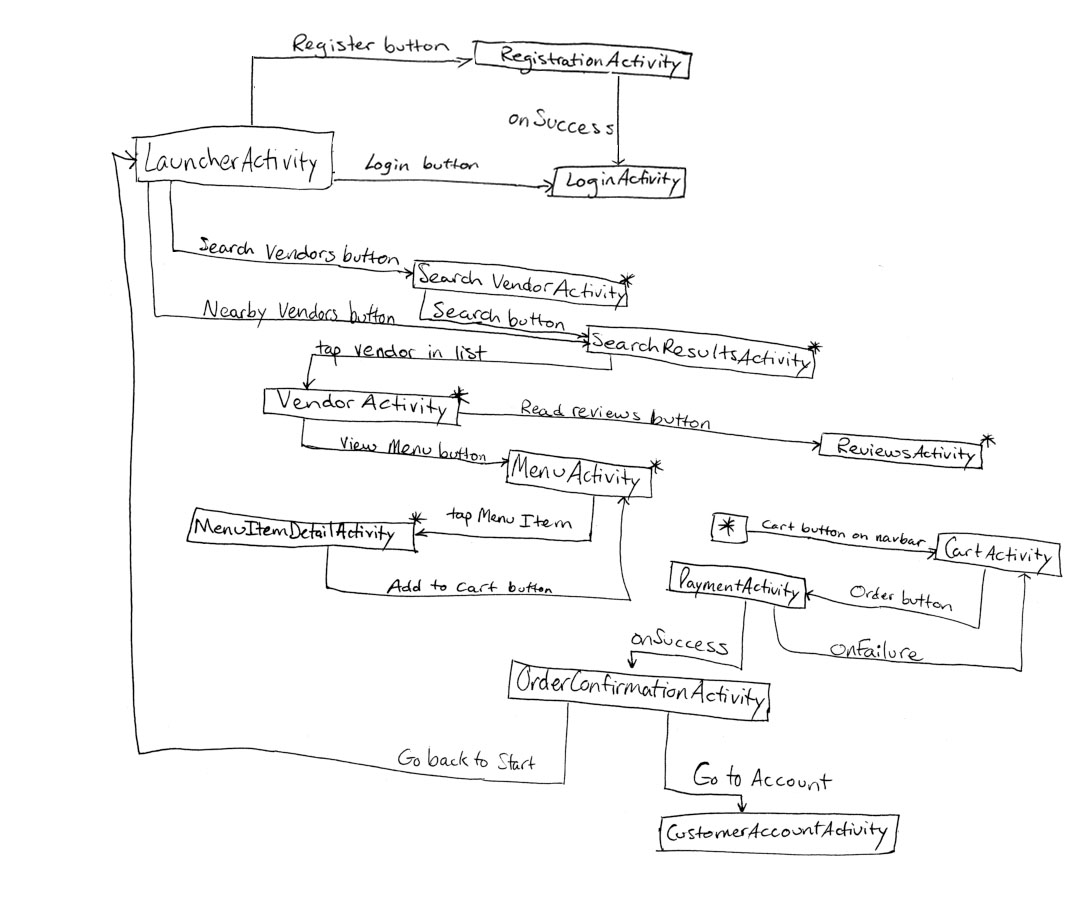
1. **User Class:** holds both, vendor and customer of the yummy system. It has a dependency relationship with the Order class because orders strictly depend on users and the change in structure or behavior of one will affect the other. It also has an aggregation relationship with the Vendor class. Each user can have 0 to many Vendors and each Vendor can belong to only 1 User.
2. **Order class**: contains the order information. It has a dependency relationship with User class and composition relationships with ShoppingCart and MenuItems classes. Each ShoppingCart has 0 to 1 Orders. Each Order has 1 to many MenuItems.
3. **ShoppingCart class:** contains order with items and total price. It has composition relationships with Order and MenuItems classes. Each ShoppingCart has 0 to many MenuItems and MenuItems can belong to many ShoppingCart.
4. **Vendor class:** vendor class contains the food truck information. It has aggregation relationships with User, VendorRating, Menu and SpecialOffers classes. Each Vendor can have 0 to many SpecialOffers, only 1 Menu and 0 to many Vendor Ratings.
5. **VendorRating class:** it contains the ratings and reviews of each Vendor. It has an aggregation relationship with Vendor. Each VendorRating can only belong to 1 Vendor.
6. **Menu class:** it contains the list of MenuItems. It has an aggregation relationship with Vendor and a composition relationship with MenuItems. Each Menu belongs to only 1 Vendor and each Menu is composed of many MenuItems. It also has a dependency relationship with SpecialOffers.
7. **Ingredients class:** it contains the ingredient information. It has composition relationships with SpecialOffers and MenuItems.
8. **MenuItems class:** It contains the information about each item in the menus. It has composition relationships with Order, Menu, ShoppingCart and Ingredients. MenuItems is composed of ingredients, ShoppingCart, Menu and Orders are all composed of MenuItems.
9. **SpecialOffers class:** contains MenuItems on a special offer. It has a composition relationship with MenuItems, aggregation relationship with Vendor and a dependency relationship with Menu. SpecialOffers is composed of MenuItems. Each Vendor can have 0 to many SpecialOffers. Menu class is affected if SpecialOffers class is modified.



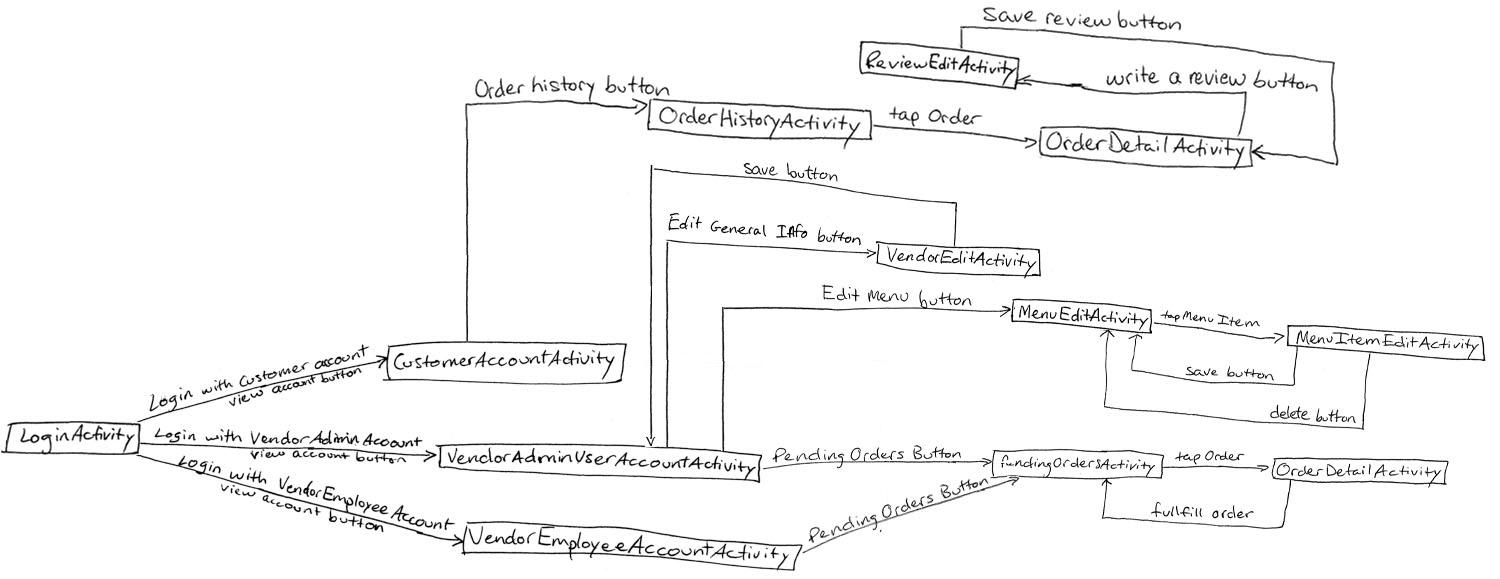
Figure

**2.2 UI Flow Diagram**

The following is a UI flow diagram showing the different Activities and how the user may move between them given certain actions. Note that in Android development, an Activity represents each different “screen.” Therefore, we have treated “screen” and “activity” synonymously.



**UI flow from LoginActivity:**



# Interface Description

## User Interface

This section defines the data exchange between users and the system

* + - * Vendor Admin interface - this is the interface for the Vendor Admins to view and modify their information, menu, and inventory, as well as perform analysis.
      * Customer interface - this is the interface for customers to search for Vendors (by name and location), view Vendor’s menus, and place orders.

## Data Interface

This section defines the data interaction between Yummy and any external systems without users’ involvement

* Transaction (System – PayPal/Drexel) – Yummy will send order information to PayPal/Drexel’s Dragon Dollar service for payment processing
* Vendor Location (System – Google map) – System will send Vendor location information to Google Maps in order to properly display the Vendor’s location on a map.

## Programming Interface

There are no programming interfaces that will be available to any external system.

# Detailed Design

**4.1 Frontend**

**F1 - Ingredients**

**Type:** Object

**Requirement:** This entity meets Requirement R.V.3.

**Description:** The Ingredients embodies the particular ingredient in each menu item.

**Name:** Ingredients

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Name | String | Name of ingredient |
| Price | Double | Price of ingredient, 2 decimal place precision |

**Methods:**

void addIngredient(name, price)

void removeIngredient()

string getName()

double getPrice()

void setName(name)

void setPrice(price)

**F2 – Launcher**

**Type:** Object

**Requirement:** This entity meets Requirement 3.1.

**Description:** Starts up the application and the Launcher Activity

**Name:** Launcher

**Data:** N/A

**Methods:**

void launchApp()

**F3 - Menu**

**Type:** Object

**Requirement:** This entity meets Requirement R.V.3

**Description:** The Menu object reflects the list of menu items available for a particular vendor.

**Name:** Menu

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| MenuItems | MenuItems | List of MenuItem objects |

**Methods:**

string[] getMenuItems()

void setMenuItems(MenuItems[])

**F4 - MenuItem**

**Type:** Object

**Requirement:** This entity meets Requirement R.C.3

**Description:** The MenuItem object is representative of the different items found in a menu.

**Name:** MenuItem

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Name | String | Name of menuitem |
| Price | Double | Price of ingredient, 2 decimal place precision |
| Ingredients | Ingredients | Ingredients contained in MenuItem |
| Active | Boolean | is the MenuItem currently on the menu |

**Methods:**

void addMenuItem()

void removeMenuItem(MenuItem)

string getName()

double getPrice()

void Ingredients[] getIngredients()

boolean active()

void setName(name)

void setPrice(price)

void setIngredients(Ingredients[])

void setActive(active)

**F5 - Order**

**Type:** Object

**Requirement:** This entity meets Requirement R.C.4.

**Description:** The Order object contains the information of orders made by the users and is handled between both the user and vendor.

**Name:** Order

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| ID | Integer | Identifies the order |
| Duration | Integer | Time it takes to cook order in minutes |
| Status | Enum | Indication of whether the order has been completed |
| TotalPrice | Double | Price of the total order |
| Item | ArrayList<MenuItem> | All items in the order |

**Methods:**

int getOrderIDNumber()

Function: Retrieve the ID number of the order.

Precondition:

Postcondition:

int getDuration()

Function: Retrieve the Duration of the order.

Precondition:

Postcondition:

boolean getStatus()

Function: Retrieve the Status of the order.

Precondition:

Postcondition:

void setWaitingTime(int duration)

Function: Give the duration of how long the order will take.

Precondition:

Postcondition:

ArrayList<MenuItems> getOrderItems()

Function: Retrieves the array of MenuItems.

Precondition:

Postcondition:

double getTotalOrderPrice()

Function: Retrieves the total price of the order.

Precondition:

Postcondition:

**F6 - ShoppingCart**

**Type:** Object

**Requirement:** This entity meets Requirement R.C.5.

**Description:** Compilation of MenuItems for a given Order.

**Name:** ShoppingCart

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Items | MenuItems |  |
| TotalPrice | Double | Price of total number of items and will be a 2 decimal precision. |

**Methods:**

void addItem(MenuItem item)

Function: Adds one item to the MenuItem array (E16 cart interface).

Precondition: Item exists in the database for this function.

Postcondition: Item is added to the checkout list. Updates total price.

void removeItem(MenuItem item)

Function: Removes one item from the MenuItem array (E16 cart interface).

Precondition: Items exist in the shopping cart list.

Postcondition: Item is removed from the shopping cart list. Updates total price.

ArrayList<MenuItems> getItems()

Function: Retrieves the array of MenuItems.

Precondition:

Postcondition:

void setItems(ArrayList<MenuItem> item)

Function: Give the cart an array list of MenuItem.

Precondition:

Postcondition:

void removeAllItems()

Function: Removes all items in the MenuItem array.

Precondition: IsSure boolean must be set to true.

Postcondition: Removes all items in MenuItem array. Updates total price.

IsSure boolean is set to false.

double getTotalPrice()

Function: Retrieves the total price of the cart.

Precondition:

Postcondition:

void updateTotalPrice()

Function: Sets the value of TotalPrice

Precondition:

Postcondition: TotalPrice is updated with current list of items.

**F7 - SpecialOffers**

**Type:** Object

**Requirement:** This entity meets Requirement R.C.8.

**Description:** The SpecialOffers object holds the deals that the vendor may wish to implement at any time.

**Name:** SpecialOffers

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| MenuItem | MenuItem | Name of the MenuItem undergoing special offer |
| Price | Double | Price of menuitem undergoing special offer |
| Duration | Integer | Duration of the offer expressed in days |

**Methods:**

MenuItem getMenuItem()

Function: Retrieves the MenuItem that has a special offer.

Precondition:

Postcondition:

double getPrice()

Function: Retrieves the price of the special offer.

Precondition:

Postcondition:

int getDuration()

Function: Retrieves the duration of the special offer.

Precondition:

Postcondition:

void setMenuItem(MenuItem Item)

Function: Set the MenuItem that has a special offer.

Precondition:

Postcondition:

void setPrice(double price)

Function: Give the MenuItem a special offer price.

Precondition:

Postcondition:

void setDuration(int duration)

Function: Give the MenuItem a time that the special offer exists for.

Precondition:

Postcondition:

**F8 - User**

**Type:** Object

**Requirement:** This entity meets Requirement 3.1.2.

**Description:** The User object defines a user in our application. It will hold the basic information of a user in their profile.

**Name:** User

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Username | String |  |
| Name | String |  |
| Email | String |  |
| PhoneNumber | String |  |
| Role | String |  |

**Methods:**

string getUsername()

string getName()

string getEmail()

string getPhoneNumber()

string getRole()

void setUserName(username)

void setName(name)

void setEmail(email)

void setPhoneNumber(phoneNumber)

void setRole(role)

**F9 - Vendor**

**Type:** Object

**Requirement:** This entity meets Requirement 3.1.2.

**Description:** The Vendor object contains the information of the particular food truck/vendor.

**Name:** Vendor

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Name | String |  |
| Description | String |  |
| Location | String |  |
| OpenTime | Integer |  |
| CloseTime | Integer |  |
| Status | Boolean |  |
| WaitTime | Integer | Expressed in minutes |

**Methods:**

void openTruck()

Function: Indicate to users that the truck is open for business.

Precondition:

Postcondition: IsOpen boolean value will be set to true.

void closeTruck()

Function: Indicate to users that the truck is closed for business.

Precondition:

Postcondition: IsOpen boolean value will be set to false.

string getName()

Function: Retrieve the name of the vendor.

Precondition:

Postcondition:

string getDescription()

Function: Retrieve the description of the vendor.

Precondition:

Postcondition:

string getLocation()

Function: Retrieve the location of the vendor.

Precondition:

Postcondition:

int getOpenTime()

Function: Retrieve the time that the vendor opens for business.

Precondition:

Postcondition:

int getCloseTime()

Function: Retrieve the time that the vendor closes business for the day.

Precondition:

Postcondition:

boolean getStatus()

Function: Retrieve the current status of the vendor’s business.

Precondition:

Postcondition:

int getWaitTime()

Function: Retrieve the amount of time a user would have to wait for his or her order to be ready.

Precondition:

Postcondition:

void setName(name)

Function: Set the name of the vendor.

Precondition:

Postcondition:

void setDescription(description)

Function: Give the vendor a description.

Precondition:

Postcondition:

void setLocation(location)

Function: Give the vendor a location.

Precondition:

Postcondition:

void setOpenTime(time)

Function: Give the vendor a time that it opens for business.

Precondition:

Postcondition:

void setCloseTime(time)

Function: Give the vendor a time that it closes for business.

Precondition:

Postcondition:

void setStatus(status)

Function: Assign the vendor a status of either “open” or “closed.”

Precondition:

Postcondition:

void setWaitTime(waitTime)

Function: Set the period of time a user will need to wait for his or her food to be prepared.

Precondition: WaitTime must be greater than or equal to zero.

Postcondition:

**F10 - VendorReview**

**Type:** Object

**Requirement:** This entity meets Requirement R.C.6.

**Description:** A VendorReview object represents a review and rating a customer has left for a particular vendor.

**Name:** VendorReview

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Comment | String |  |
| Stars | Integer |  |

**Methods:**

string getComment()

int getStars()

void setComment(comment)

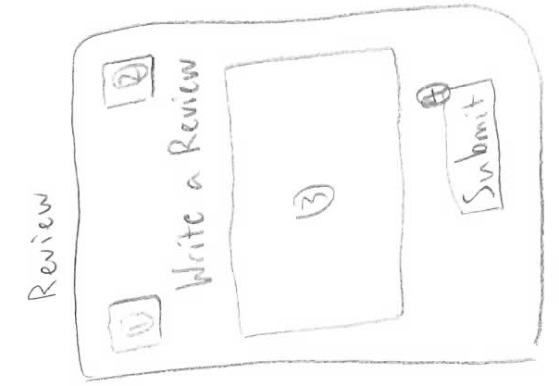
void setStars(stars)

**F11 – Review (write)**

**Type:** Screen

**Requirement:** This entity meets Requirement R.C.6.

**Description**

****

**Notes**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. A textarea box for a review to be written.

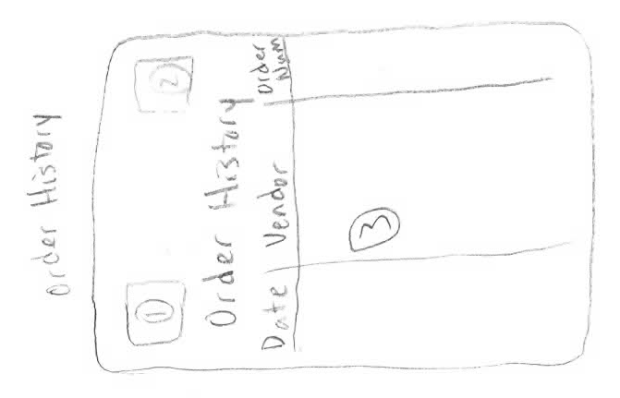
4. The Submit button will move to the Order History screen.

**F12 – Order History**

**Type:** Screen

**Requirement:** This entity meets Requirement R.C.5.

**Description**



**Notes**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

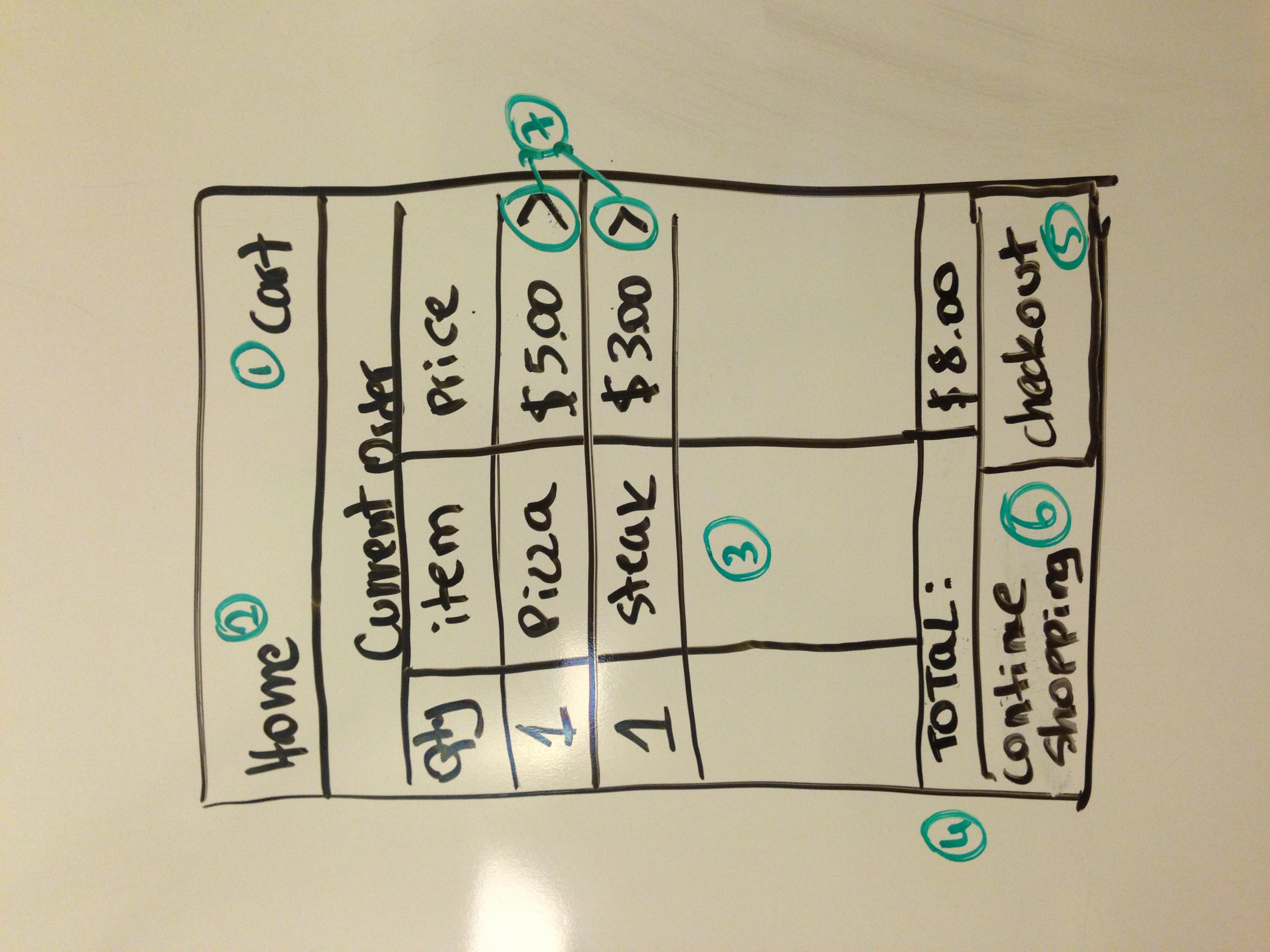
3. A scroll list of all the previous orders the user has made with his account is listed.

**F13 – Cart**

**Type:** Screen

**Requirement:** This entity meets Requirement R.C.4.

**Description**



**Notes**

1. A Cart button will move to the Cart screen and display the Cart.

2. A Home button to return to the Launch screen of the application.

3. A scrollable list of all the menu items and the corresponding information is listed.

4. A total price of the menu items in the Cart will be displayed.

5. A Checkout button will transition to the Payment screen.

6. A Continue Shopping button that will return to the menu screen

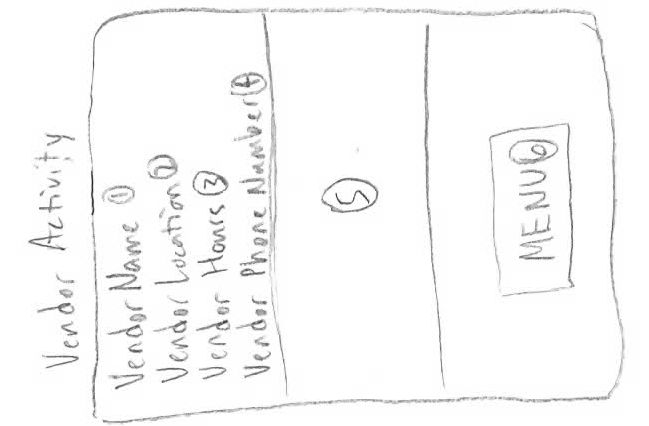
7. Arrow buttons next to each item that takes to the menu item screen (F28)

**F14 – Vendor**

**Type:** Screen

**Requirement:** This entity meets Requirement 3.1.2.

**Description**



**Notes**

1. The name of the Vendor.

2. The location of the Vendor.

3. The hours of operation of the Vendor.

4. The phone number of the vendor.

5. A local map of the area highlighting the vendor’s location

6. The Menu button will move to the Menu Screen

**F15 – Search Results**

**Type:** Screen

**Requirement:** This entity meets Requirement 3.1.2.

**Description**



**Notes**

1. The expression that was searched for.

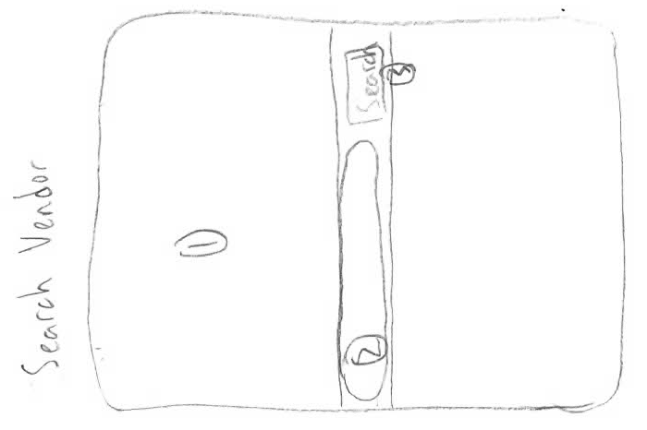
2. A scrollable list of the Vendors returned in the search.

**F16 – Search Vendor**

**Type:** Screen

**Requirement:** This entity meets Requirement 3.1.2.

**Description**



**Notes**

1. A map of the area and all Vendors’ locations throughout.

2. The expression being searched for.

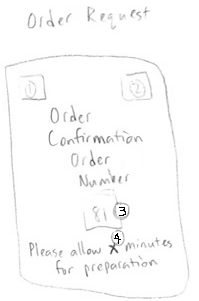
3. The Search button leads to the Search Results screen

**F17 – Order Request**

**Type:** Screen

**Requirement:** This entity meets Requirement R.C.5.

**Description**



**Notes**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. The Order Number.

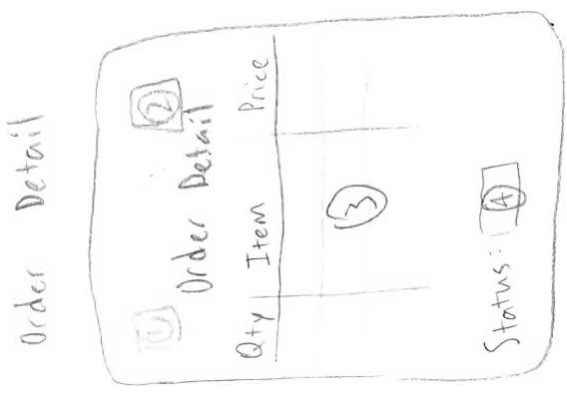
4. The amount of time allotted before the order is ready for pickup.

**F18 – Order Detail**

**Type:** Screen

**Requirement:** This entity meets Requirement R.C.5.

**Description**



**Notes**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. A scrollable list of all the menu items and the corresponding information.

4. The status of the order (whether it is ready for pickup or not).

**F19 – Reviews (Read)**

**Type:** Screen

**Requirement:** This entity meets Requirement R.C.7.

**Description**



**Notes**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

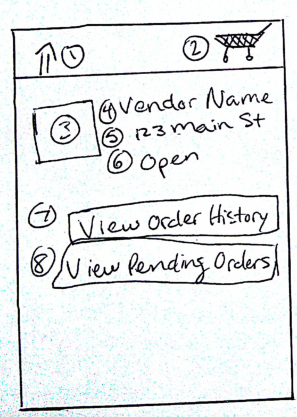
3. A scrollable list of reviews written by other users.

**F20 – VendorEmployeeActivity**

**Type:** Screen

**Requirement:** This entity partially fulfills Requirement R.V.1

**Description:**



1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Vendor logo or picture

4. Vendor.Name attribute

5. Vendor.Location attribute

6. Vendor.Status attribute

7. Button to go to OrderHistoryActivity

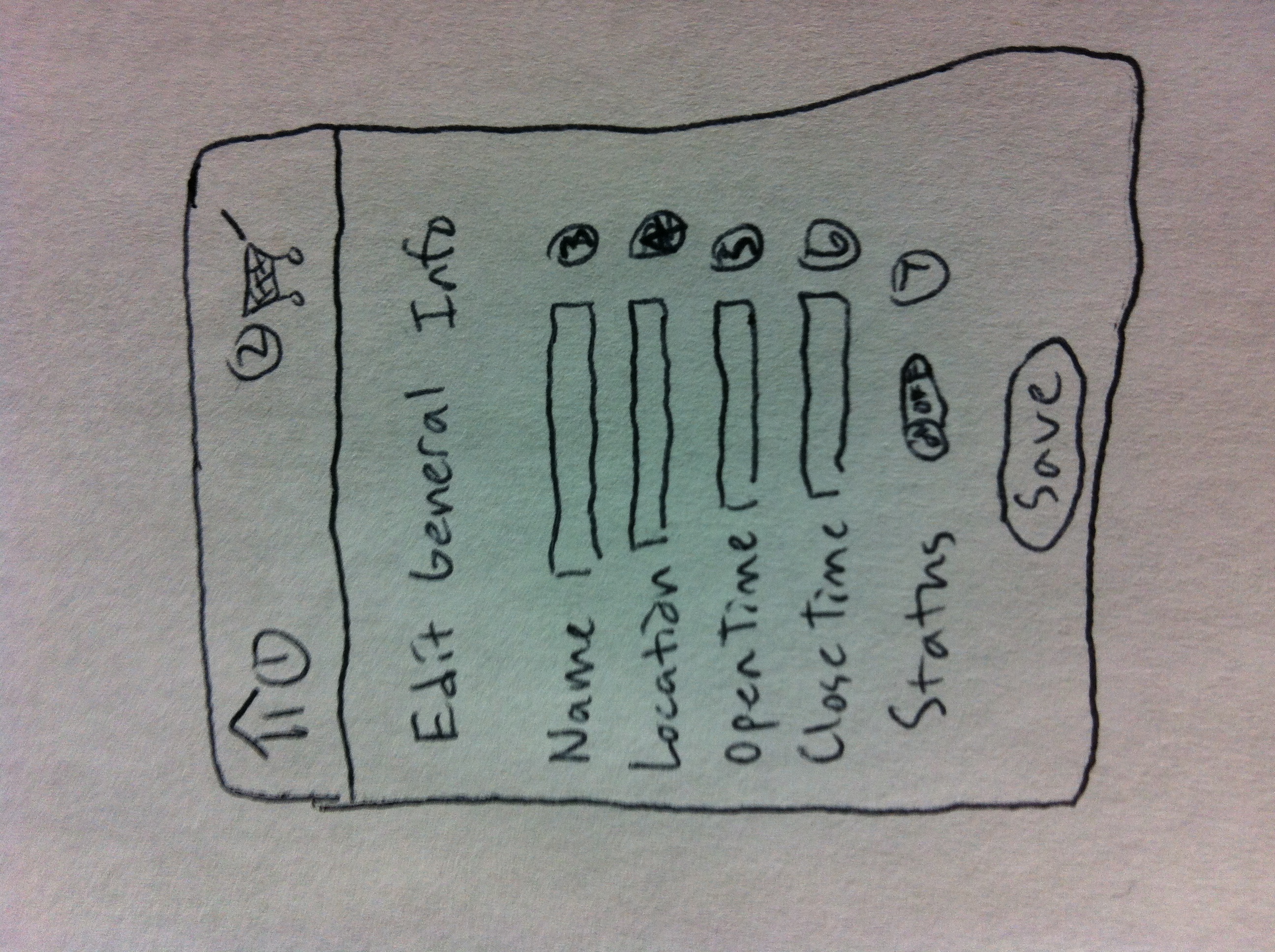
8. Button to go to PendingOrdersActivity

**F21 - Name:** VendorEditActivity

**Type: Screen**

**Requirement:** This entity fulfills R.V.4

**Description:**



Notes

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Field to edit Vendor.Name attribute

4. Field to edit Vendor.Location attribute

5. Field to edit Vendor.OpenHours attribute

6. Field to edit Vendor.CloseHours attribute

7. Switch to edit Vendor.Status attribute (will switch between Open and Closed).

**F22 – MenuItemEditActivity**

**Type:** Screen

**Requirement:** This entity meets Requirement R.V.2

**Description**



This screen will allow the user to edit properties of specific item on a menu.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Photo of this menu item

4. Field to edit MenuItem.Name attribute

5. Field to edit MenuItem.Price attribute

6. Button to add an Ingredient to this MenuItem

7. Button to remove an ingredient from this MenuItem

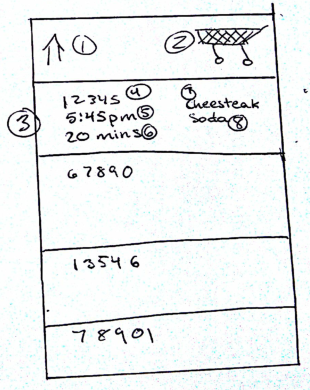
8. List of Ingredients this MenuItem currently has

**F23 – PendingOrdersActivity**

**Type:** Screen

**Requirement:** This entity meets Requirement R.V.10

**Description**

****

This screen shows all orders that have not yet been picked up and are in the process of being prepared.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Each list item represents an Order. When tapped, the OrderDetailActivity is launched

4. Order.OrderNumber attribute

5. Order.TimeCreated attribute

6. Countdown timer showing the amount of time left until the Order.EstimatedTime expires.

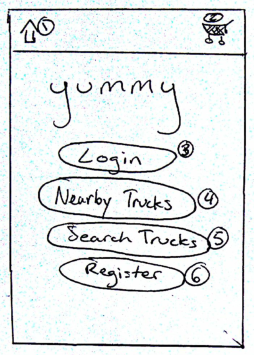
7 & 8. A small list of MenuItems that are part of the order. Only the first two items will be shown

**F24 – LauncherActivity**

**Type:** Screen

**Requirement:** This entity meets Requirement 3.1.2

**Description**

****

This screen is the first screen the user will see. It is the entry point to the application.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Button will launch the LoginActivity

4. Button will launch SearchResultsActivity with vendors sorted by distance from user

5. Button will launch SearchVendorActivity

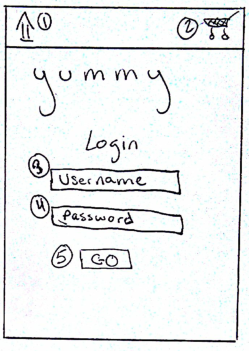
6. Button will launch RegistrationActivity

**F25 – LoginActivity**

**Type:** Screen

**Requirement:** This entity partially fulfills requirement R.C.10

**Description**

****

This screen allows the user to login using their username and password.

**Methods:**

String authentication(String userName, String password)

Function: searches and verifies that the username and password match.

Precondition: username and password must exist in the database.

Postcondition: username and password is a match.

void login()

Function: connects the android app to the web service.

Precondition: web service must be implemented and running.

Postcondition: android app is connected to the web service for username and password verification.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Field for User.username

4. Field for User.password

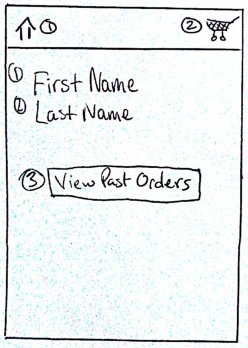
5. Button to launch one of the account Activities if login information is correct

**F26 – CustomerAccountActivity**

**Type:** Screen

**Requirement:** This entity partially fulfills requirement R.C.10

**Description**

****

This screen allows the user to login using their username and password.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

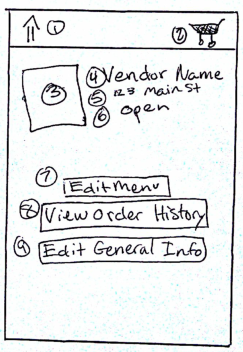
3. Button to show past orders

**F27 – VendorAdminAccountActivity**

**Type:** Screen

**Requirement:** This entity partially fulfills requirement R.C.10

**Description**

****

This screen allows the user to login using their username and password.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Vendor logo or picture

4. Vendor.Name attribute

5. Vendor.Location attribute

6. Vendor.Location attribute

7. Vendor.Status attribute

8. Button to launch OrderHistoryActivity

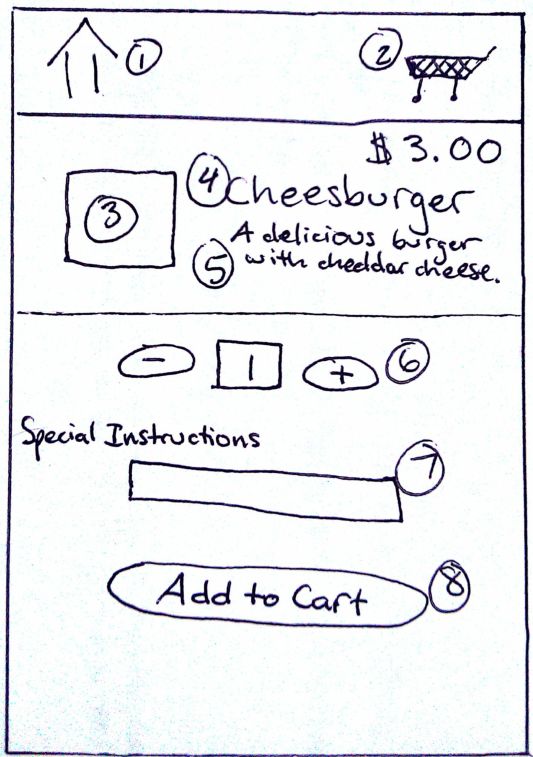
9. Button to launch VendorEditActivity

**F28 – MenuItemActivity**

**Type:** Screen

**Requirement:** This entity fulfills requirement R.C. 4

**Description**

****

This screen is used in two circumstances:

A. When the user selects a menu item from the MenuActivity screen

* The quantity picker (6) will be initialized to 1.
* The button (8) text will be “Add to Cart” and will launch the AddMenuItemToCartConfimationDialog (F29)

B. When the user selects a menu item from the CartActivity screen

* The quantity picker (6) will be initialized to the amount of the menu item currently in the user’s cart.
* The button (8) text will be “Update” and will launch a confirmation dialog.

**Notes:**

1. A Home button to return to the Launch screen of the application.

2. A Cart button will move to the Cart screen and display the Cart.

3. Menu item picture

4. MenuItem.Name attribute

5. MenuItem.Description attribute

6. Number picker to choose quantity to add to cart

7. Text field to input special preparation instructions for the Vendor to follow

8. Button that adds the specified quantity of this MenuItem to the cart

Text changes to “Update” if this screen is reached from the cart.

**F29 – AddMenuItemToCartConfirmationDialog**

**Type:** UI Dialog Box

**Requirement:** This entity partially fulfills requirement \_\_\_\_\_

**Description**

****

A dialog to confirm to the user the menu item and quantity added to the cart.

**Notes:**

1. The number in the quantity picker (8 in F28) at the time the user added the item to the car.

2. Button to dismiss the dialog

3. Button to return the user to the Menu

**4.2 Backend**

**B1 - Ingredients**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement R.V.3.

**Description:** The Ingredients embodies the particular ingredient in each menu item.

**Name:** Ingredients

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Name | String | Name of ingredient |
| Price | Double | Price of ingredient, 2 decimal place precision |

**Methods:**

void deleteIng(String name)

Function: Deletes ingredient entry from the database.

Precondition: Data entry must exist in the database.

Postcondition:

string getName()

Function: Retrieves the name of the ingredient from the database.

Precondition:

Postcondition:

double getPrice()

Function: Retrieves the price of the ingredient from the database.

Precondition:

Postcondition:

void setName(String name)

Function: Include ingredient into the database.

Precondition: Entry does not already exist in database.

Postcondition:

void setPrice(String name, double price)

Function: Include price of ingredient into the database.

Precondition:

Postcondition:

**B2 - MenuItem**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement R.C.3

**Description:** The MenuItem object is representative of the different items found in a menu.

**Name:** MenuItem

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Name | String | Name of menuitem |
| Price | Double | Price of ingredient, 2 decimal place precision |
| Ingredients | Ingredient[] | Ingredients contained in MenuItem |
| Active | Boolean | is the MenuItem currently on the menu |

**Methods:**

void deleteMenuItem(String name)

Function: Deletes the MenuItem entry from the database.

Precondition: Data entry must exist in the database.

Postcondition:

string getName()

Function: Retrieves the name of the MenuItem from the database.

Precondition:

Postcondition:

double getPrice()

Function: Retrieves the price of the MenuItem from the database.

Precondition:

Postcondition:

boolean isActive()

Function: Retrieves value of isActive from the database.

Precondition:

Postcondition:

void setName(String name)

Function: Sets MenuItem into the database.

Precondition: Entry does not already exist in the database.

Postcondition:

void setPrice(String name, double price)

Function: Sets price of the MenuItem into the database.

Precondition:

Postcondition:

void setIngredients(String name, Ingredients[] ingredient)

Function: Sets ingredients to MenuItem into the database.

Precondition: Ingredients must exist in the database.

Postcondition:

void setActive(boolean active)

Function: Sets value for active into the database.

Precondition:

Postcondition:

**B3 - Order**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement R.C.4.

**Description:** The Order object contains the information of orders made by the users and is handled between both the user and vendor.

**Name:** Order

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| ID | Integer | Identifies the order |
| Duration | Integer | Time it takes to cook order in minutes |
| Status | Enum | Indication of whether the order has been completed |
| TotalPrice | Double | Price of the total order |
| Item | ArrayList<MenuItem> | All items in the order |

**Methods:**

int getOrderIDNumber()

Function: Retrieve the ID number of the order from the database.

Precondition:

Postcondition:

int getDuration()

Function: Retrieve the Duration of the order from the database.

Precondition:

Postcondition:

boolean getStatus()

Function: Retrieve the Status of the order from the database.

Precondition:

Postcondition:

void setWaitingTime(int duration)

Function: Set the duration of how long the order will take into the database.

Precondition:

Postcondition:

double getTotalOrderPrice()

Function: Retrieves the total price of the order in the database.

Precondition: Total price is not null.

Postcondition:

**B4 - SpecialOffers**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement R.C.8.

**Description:** The SpecialOffers object holds the deals that the vendor may wish to implement at any time.

**Name:** SpecialOffers

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| MenuItem | MenuItem | Name of the MenuItem undergoing special offer |
| Price | Double | Price of menuitem undergoing special offer |
| Duration | Integer | Duration of the offer expressed in days |

**Methods:**

MenuItem getMenuItem()

Function: Retrieves the MenuItem that has a special offer from the database.

Precondition:

Postcondition:

double getPrice()

Function: Retrieves the price of the special offer from the database.

Precondition:

Postcondition:

int getDuration()

Function: Retrieves the duration of the special offer from the database.

Precondition:

Postcondition:

void setMenuItem(MenuItem Item)

Function: Set the MenuItem that has a special offer into the database.

Precondition:

Postcondition:

void setPrice(double price)

Function: Give the MenuItem a special offer price into the database.

Precondition:

Postcondition: Price

void setDuration(int duration)

Function: Give the MenuItem a time that the special offer exists for into the database.

Precondition:

Postcondition:

**B5 - User**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement 3.1.2.

**Description:** The User object defines a user in our application. It will hold the basic information of a user in their profile.

**Name:** User

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Username | String |  |
| Name | String |  |
| Email | String |  |
| PhoneNumber | String |  |
| Role | String | Role can be either “guest,” “customer,” “employee,” “vendor admin” |

**Methods:**

string getUsername()

Function: Retrieve the username associated with the profile from the database.

Precondition:

Postcondition:

string getName()

Function: Retrieve the name of the user associated with the profile from the database.

Precondition:

Postcondition:

string getEmail()

Function: Retrieve the email address associated with the profile from the database.

Precondition:

Postcondition:

string getPhoneNumber()

Function: Retrieve the phone number associated with the profile from the database.

Precondition:

Postcondition:

string getRole()

Function: Retrieve the role of the user associated with the profile from the database.

Precondition:

Postcondition:

void setUserName(username)

Function: Set the username that will be associated with the profile into the database.

Precondition:

Postcondition:

void setName(name)

Function: Set the name of the user that will be associated with the profile into the database.

Precondition:

Postcondition:

void setEmail(email)

Function: Set the email address that will be associated with the profile into the database.

Precondition:

Postcondition:

void setPhoneNumber(phoneNumber)

Function: Set the phone number associated with the profile into the database.

Precondition:

Postcondition:

void setRole(role)

Function: Set the role of the user associated with the profile into the database.

Precondition:

Postcondition:

**B6 - Vendor**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement 3.1.2.

**Description:** The Vendor object contains the information of the particular food truck/vendor.

**Name:** Vendor

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Name | String |  |
| Description | String |  |
| Location | String |  |
| OpenTime | Integer |  |
| CloseTime | Integer |  |
| Status | Boolean |  |
| WaitTime | Integer | Expressed in minutes |

**Methods:**

string getName()

Function: Retrieve the name of the vendor from the database.

Precondition:

Postcondition:

string getDescription()

Function: Retrieve the description of the vendor from the database.

Precondition:

Postcondition:

string getLocation()

Function: Retrieve the location of the vendor from the database.

Precondition:

Postcondition:

int getOpenTime()

Function: Retrieve the time that the vendor opens for business from the database.

Precondition:

Postcondition:

int getCloseTime()

Function: Retrieve the time that the vendor closes business for the day from the database.

Precondition:

Postcondition:

boolean getStatus()

Function: Retrieve the current status of the vendor’s business from the database.

Precondition:

Postcondition:

int getWaitTime()

Function: Retrieve the amount of time a user would have to wait for his or her order to be ready.

Precondition: WaitTime must be greater than or equal to zero.

Postcondition:

void setName(name)

Function: Sets the name of the vendor into the database.

Precondition:

Postcondition:

void setDescription(description)

Function: Sets the vendor description into the database.

Precondition:

Postcondition:

void setLocation(location)

Function: Sets the vendor location into the database.

Precondition:

Postcondition:

void setOpenTime(time)

Function: Sets the vendor time that it opens for business into the database.

Precondition:

Postcondition:

void setCloseTime(time)

Function: Sets the vendor time that it closes for business into the database.

Precondition:

Postcondition:

void setStatus(status)

Function: Sets the vendor status of either “open” or “closed” into the database.

Precondition:

Postcondition:

void setWaitTime(waitTime)

Function: Set the period of time a user will need to wait for his or her food to be prepared into the database.

Precondition: WaitTime is greater than or equal to zero.

Postcondition:

**B7 - VendorReview**

**Type:** cakePHP Model Class

**Requirement:** This entity meets Requirement R.C.6.

**Description:** A VendorReview object represents a review and rating a customer has left for a particular vendor.

**Name:** VendorReview

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Notes** |
| Comment | String |  |
| Stars | Integer |  |

**Methods:**

string getComment()

Function: Retrieve comment from the database.

Precondition:

Postcondition:

int getStars()

Function: Retrieve the number of stars from the database.

Precondition:

Postcondition:

void setComment(comment)

Function: Input written comment into the database.

Precondition:

Postcondition:

void setStars(stars)

Function: Input the number of stars into the database.

Precondition:

Postcondition: