PARROT CAFE

Software Design Specification

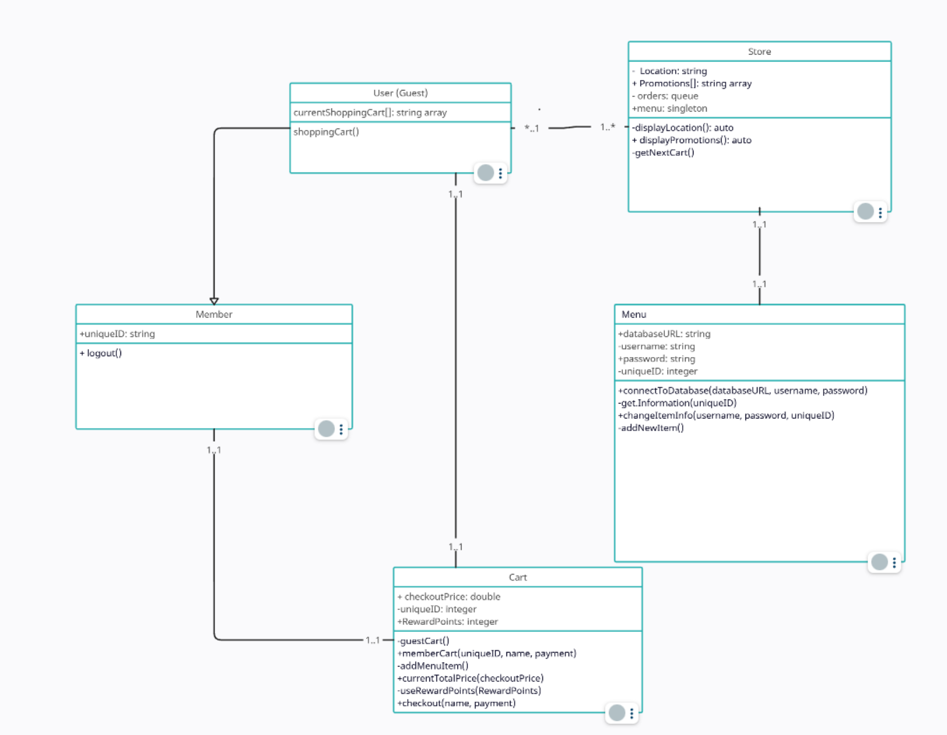
by Chaz Gabelman, Eric Soto,
Gerardo Reza, and Luke Patterson
(Group 9)

Software for a Cafe

Overview

This software is for customers to be able to use a guest or rewards account to order items from a Cafe. The software is capable of determining which items are in stock. There is a menu page listing all menu items and their information including price, calories, and ingredients, from which customers can choose which items they want to add to their shopping cart. There is a shopping cart page to review their order and to allow for the removal of any items they do not want anymore. Finally, included on the shopping cart page is a payment area where customers can choose to checkout by either utilizing the information tied to their reward account or by inputting their payment information and name if they do not have a reward account. The software also contains a rewards system that takes in how many points members have accumulated via previous purchases and allows them to use their points on the checkout page for a discount.

Classes



Class Descriptions

(if no return type is specified, assume that that method has a void return type)

- Guest
  - Attributes:
    - Stores the guest’s current shopping cart object which will contain the customer’s selected menu items for an order.
  - Methods:
    - Has a constructor which is called when an initial connection is made to the website. This constructor creates an empty guest shopping cart (utilizing the empty parameter shopping cart constructor) using a “has-a” relationship between the guest class and the shopping cart class.
    - Has a login method that takes the customer’s username and password as parameters which is then compared with the users’ database for verification. If verification is successful, a rewards member object is constructed using the information from the database which gives the user access to their account.
    - The login method utilizes two helper functions, namely the compareUsername and comparePassword functions which take the given information and compare them against a singular row in the user database. If the comparison is made and the two are found to be equal, these functions will return true and if the comparison fails, these functions should return false.
- Member
    - The subclass of the Guest class inherits attributes and methods from the Guest class
  - Attributes:
    - Has integer type attribute for their uniqueID in the database table which allows for the specific retrieval and editing of information present within the user database.
  - Methods:
    - Has a logout method, that takes no parameters, changing the user’s access level back to that of a normal customer via constructing a new guest object, while also destroying the reward class object and its included shopping cart.
- Store
  - Attributes:
    - The store class stores the physical address of the location in a string variable.
    - Current promotions being run by the store are also stored as strings in an array of strings.
    - The store contains a queue of carts that represents orders sent in by customers.
    - The store contains a single instance of the menu class, specifically intending a singleton structure. This menu instance provides the sole connection to the menu item database.
  - Methods:

        - Has a method for displaying the promotions currently stored in the promotions array.
        - Has a method for displaying the location currently stored in the location attribute.
        - Has a method for getting the next cart from the queue of carts in order for the store to process the next order.

- Menu
    - Attributes:
        - Contains multiple strings containing the necessary information for establishing a connection to the menu items database including the database URL, username, and password.
    - Methods:
        - Contains a method to connect to the database utilizing the URL, username, and password stored in the menu attributes.
        - Contains getter methods for getting information including the name, price, calories, ingredients, and stock from the menu item database utilizing the uniqueID of the menu item in question.
        - Contains setter methods for changing information of already existing menu items in the menu item database utilizing the uniqueID of the menu item. This method requires the inputting of the admin's username and password in order to prevent unauthorized changing of the menu item database. (Returns true if successful, false if not).
        - Finally, the menu also includes a method for adding new entries to the menu item database. Similar to the setter function, this method requires the inputting of the admin's credentials in order to prevent unauthorized changing of the menu item database.

- Cart
    - Attributes:
        - The cart contains a double used to store the current price of all the items currently in the cart.
        - The menu items are stored in an integer array where each integer stored in the array is the uniqueID of a menu item within the menu item database.
        - The uniqueID of the customer who owns this cart will be stored as an integer. This will be null if a guest owns this cart, otherwise, it will be filled by the given parameter to the cart constructor.
        - The name and the payment information of the customer who owns this cart are stored in a string. This will be null if this cart belongs to a guest, and will be autofilled utilizing the user database if this cart belongs to a reward member.
        - Finally, a boolean representing whether the customer has elected to use their accrued reward points on this cart's order will be stored in the cart's attributes. This is by default false but can be changed utilizing the useRewardPoints method.

- Methods:
    - The cart utilizes two different constructor methods depending on whether it is created for a guest or for a reward member. The no-parameter constructor creates an empty cart with default null values for the uniqueID, name, and payment information attributes. The single-parameter constructor takes in an integer representing the reward customer's uniqueID on the user database. The name and the payment information attributes will be filled in using information from the user database.
    - Has a method for adding an integer to the menu item array, while automatically updating the total price of the cart.
    - Has a getter that returns the current total price of the cart stored in the cart's attributes.
    - Has a method that allows the customer to choose whether or not to use their accrued reward points for this order. This method should only succeed if this cart belongs to a reward member.
    - There is a checkout function which, if necessary, prompts the user for their name and payment information. Once the payment information is verified as legitimate, the cart's data will be sent to the store's server and the local instance of it will be destroyed.

Updates from Assignment 2

Our class structure has gone through a number of changes. I would attribute the motivation behind the bulk of the changes was simplification as well as an added appreciation for the usefulness of databases.

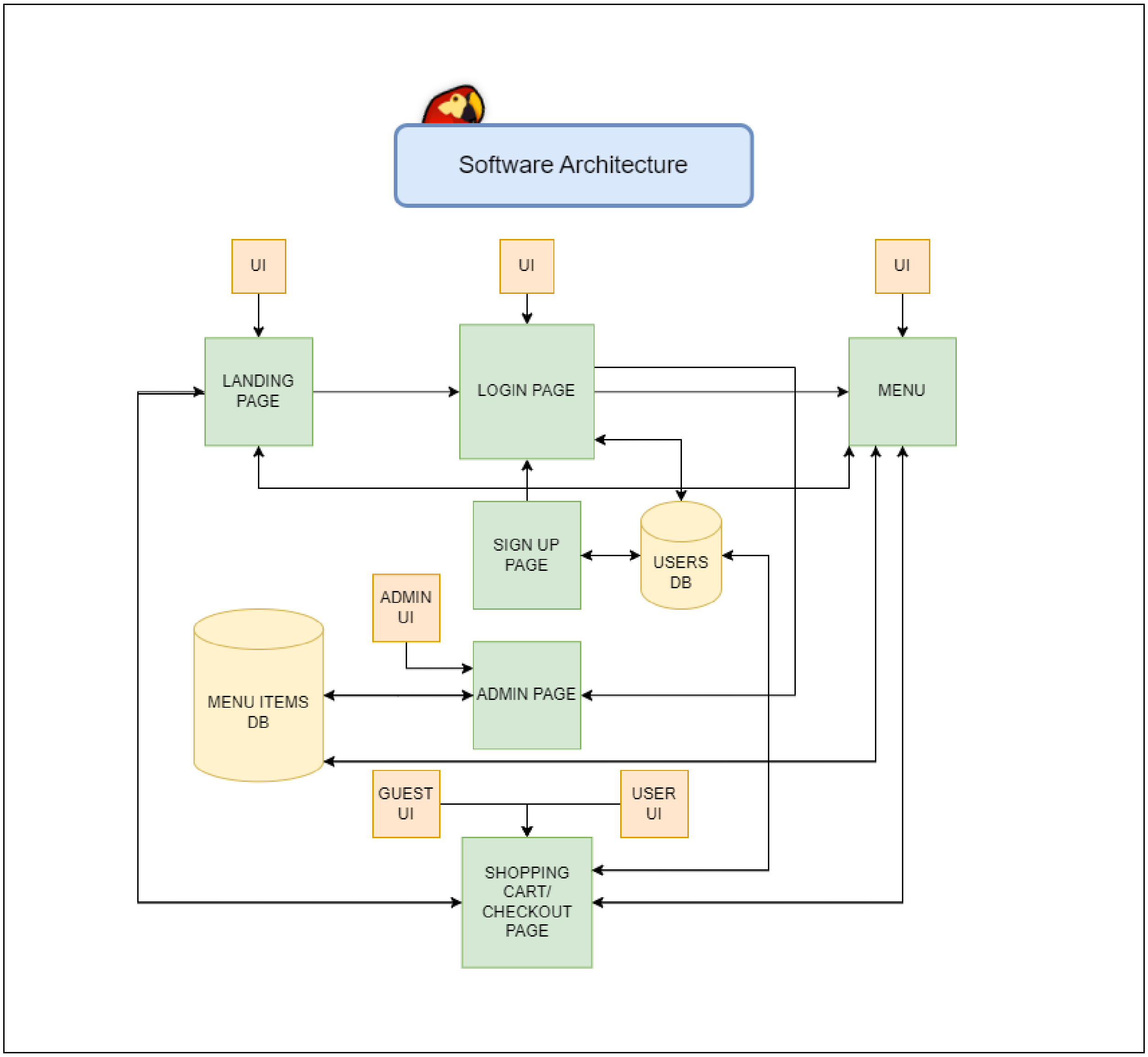
Our original structure was intended for use with multiple franchises at different locations. We intended for the site to have the added functionality of being able to pick which physical location the customer wanted their order to go to. We removed this functionality in order to focus more on fleshing out the other functionalities.

We initially had multiple layers of nested objects which formed the menu. We wanted the menu class to consist of menu items and each menu item object was supposed to consist of multiple ingredient classes. The ingredient objects were originally designed to simplify the matter of customizing existing menu items while simplifying the manner of tracking stock, and utilizing the stock to edit what should appear on the menu. In reality, it was difficult to track ingredients between these different mediums and thus we changed our representation of ingredients to a simple string.

We moved our menu to a database instead of storing it locally in the menu class. The menu class now has been changed to be an interface for interacting with the menu item database.

We did away with the menu item class as we believe we can get the information regarding each menu item from the menu item database utilizing the menu getter functions. We also believed it was unwieldy for the shopping carts to make copies of menu items the customer wanted from the menu, we wanted something more like a pointer to a unique instance of a menu item which we believe we get utilizing the menu item database in our current way.

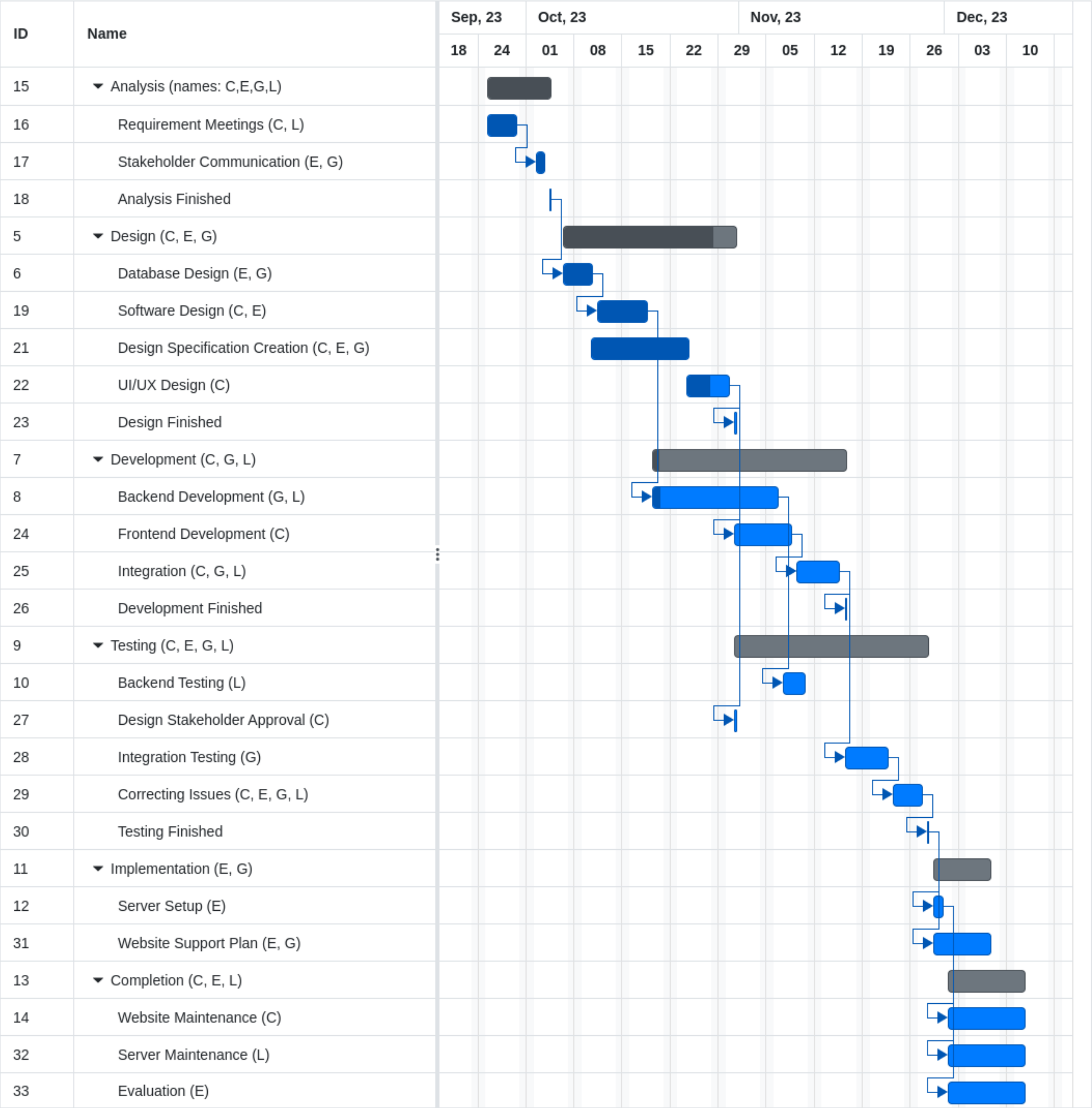
Software Architecture / Component Connections



From the landing page you will be able to go to the login, cart, or menu page. This makes it easier for users and guests to navigate through the different pages in whatever order they need. If the user decides to create an account they can do that by going to the LOGIN PAGE first then the SIGN UP PAGE. After logging in the registered user will be sent to the MENU PAGE. If the user logging in is an administrator then they will be sent to the ADMIN PAGE where they will be able to alter the MENU ITEMS DB. This page will have an ADMIN UI that enables this process. The CART PAGE will have a GUEST UI and a USER UI since only registered users get to have rewards points that will appear on that page. This would also mean that the CART PAGE is connected to the USERS DB since it needs to fetch the currently logged-in user’s rewards points.

Development Plan + Timeline

*(**please note**: Exporting the GANTT file to png/pdf collapsed several of the columns containing information including the names of those responsible, the expected start and stop date for each item, the duration of each item, and the progress on each item. The first initial of those responsible for each task was included following the name of each item (C=Chaz, E=Eric, G=Gerardo, L=Luke). The progress is represented by the portion of the bar being darkened. The full GANTT file is included in the repository.)*



ID	Name
15	Analysis (names: C,E,G,L)
16	Requirement Meetings (C, L)
17	Stakeholder Communication (E, G)
18	Analysis Finished
5	Design (C, E, G)
6	Database Design (E, G)
19	Software Design (C, E)
21	Design Specification Creation (C, E, G)
22	UI/UX Design (C)
23	Design Finished
7	Development (C, G, L)
8	Backend Development (G, L)
24	Frontend Development (C)
25	Integration (C, G, L)
26	Development Finished
9	Testing (C, E, G, L)
10	Backend Testing (L)
27	Design Stakeholder Approval (C)
28	Integration Testing (G)
29	Correcting Issues (C, E, G, L)
30	Testing Finished
11	Implementation (E, G)
12	Server Setup (E)
31	Website Support Plan (E, G)
13	Completion (C, E, L)
14	Website Maintenance (C)
32	Server Maintenance (L)
33	Evaluation (E)

Verification Test Plan:

- Unit Testing
  single code “unit” (e.g., method)
  - ➢ 1st Test Set: Guest Class - compareUsername(String givenUsername, Integer UniqueID)

Test Case #	Given Username	Unique ID	Stored Username	Expected Output
1 Unit	“Dog123”	1	“Dog123”	True
2 Unit	“dog123”	1	“Dog123”	False, incorrect username (case sensitive)
3 Unit	“Dog123”	2	“Cat123”	False, incorrect username
4 Unit	123	2	“Dog123”	ERROR, String expected, Integer given

  - ➢ 2nd Test Set: Cart class - addMenuItem(Integer uniqueID)

Test Case #	Menu item array before	Function call	Menu item array after	Expected Output
1 Unit	[1, 1, 5]	addMenuItem(6)	[1, 1, 5, 6]	Void
2 Unit	[1, 1, 5, 6]	addMenuItem(3)	[1, 1, 5, 6, 3]	Void
3 Unit	[1, 1, 5, 6, 3]	addMenuItem(“burger”)	[1, 1, 5, 6, 3]	ERROR, Integer expected, String given
4 Unit	[1, 1, 5, 6, 3]	addMenuItem(99999)	[1, 1, 5, 6, 3]	ERROR, The menu item with the given unique ID does not exist.

- Integration Testing
  interfaces among integrated units
  - ➢ 1st Test Set: Cart class - checkout() -> impacts Store class

Test Case #	Store's cart array size before	Method call	Store's cart array size after	Expected output
1 [normal use case]	2	cart.checkout()	3	Void (success) Store's cart array should have a new entry
2 [invalid payment info is given]	3	cart.checkout()	3	"ERROR, Payment information is invalid!"
3 [carde has insufficient funds]	3	cart.checkout()	3	"ERROR, the card has been declined!"

- ➢ 2nd Test Set: Menu class - setPrice(Integer uniqueID, Double price,
- ➢ String username, String password) -> impacts menu DB

Test Case #	Menu Item DB ID, price before	Method call	Menu Item DB ID, price after	Expected Output
1 [normal use]	1, 4.99	Menu.setPrice(1, 5.99, "admin", "admin")	1, 5.99	True (successful)
2 [wrong data type inputted]	1, 5.99	Menu.setPrice("one", 5.99, "admin", "admin")	1, 5.99	ERROR, Integer expected, String given
3 [Invalid admin credentials]	1, 5.99	Menu.setPrice(1, 6.99, "user", "user")	1, 5.99	False (failure) Invalid admin credentials

- System Testing

  complete system

  - ➢ 1st Test Set: (Placing an order as a member) Tests guest class -> member class -> cart class
  - ➢ Which interacts with the menu DB and interacts with the store class

Test Case #	Method Calls	Member	Menu Items Selected	Rewards Points Used	Expected Output

1 System	Guest nGuest = newGuest()
	nGuest.login(“user”, “user”)
	Member mem = newMember(1)
	mem.mCart.addMenuItem(1)
	mem.mCart.addMenuItem(2)
	mem.mCart.addMenuItem(3)
	mem.mCart.useRewardPoints()
	mem.mCart.checkout() // Valid payment info
		Yes	1, 2, 3	Yes	Success, reward points applied, the price of the cart should reflect the normal price -$1 for every 10 points applied, the user database should reflect the deduction of reward points

2 System	Guest nGuest = newGuest()
	nGuest.login(“user”, “user”)
	Member mem = newMember(1)
	mem.mCart.addMenuItem(1)
	mem.mCart.addMenuItem(2)
	mem.mCart.addMenuItem(3)
	mem.mCart.checkout() // Valid payment info
		Yes	1, 2, 3	No	Success, no reward points applied, the price of the cart should reflect the normal price

3 System	Guest nGuest = newGuest() nGuest.login(“fake”, “fake”)	Yes	N/A	Yes	ERROR, Incorrect username or password (line 2)

- ➢ 2nd Test Set (Placing an order as a guest) Tests guest class -> cart class which interacts with
- ➢ The menu item DB and interacts with the Store class

Test Case #	Method Calls	Menu Items selected	Expected Output

1 System	Guest nGuest = new Guest()
	nGuest.gCart.addMenuItem(1)
	nGuest.gCart.addMenuItem(2)
	nGuest.gCart.addMenuItem(3)
	nGuest.gCart.checkout() // Valid payment info given
		1, 2, 3	Success, store’s cart queue should have a new entry

2 System	Guest nGuest = new Guest() nGuest.gCart.addMenuItem(“One”)	None	ERROR, Integer parameter expected, String given (lines 2, 3,

	nGuest.gCart.addMenuItem(“Two”)
	nGuest.gCart.addMenuItem(“Three”)
			4)

3 System	Guest nGuest = new Guest()
	nGuest.gCart.addMenuItem(1)
	nGuest.gCart.addMenuItem(2)
	nGuest.gCart.addMenuItem(3)
	nGuest.gCart.checkout() // Valid payment info given, but the card has insufficient funds
		1, 2, 3	ERROR, the card has been declined! No change to the store’s cart array.

Test Set Description:

- **What features are you testing?**
  - **Log-in test** (Unit test set #1): We tested a core functionality necessary for the log-in feature.
  - **Adding menu items to the cart test** (Unit test set #2): We tested adding menu items to the customer’s cart.
  - **Checking out a customer’s cart** (Integration test set #1): We tested our checkout feature which handles payment and ensures that the order information is sent to the store.
  - **Updating the menu item database** (Integration test set #2): We tested our menu update feature, specifically with regard to allowing an admin to change the prices of items on the menu).
  - **Placing an order as a member** (System test set #1) This feature is tested to make sure that a member has access to their rewards balance when purchasing menu items.
  - **Placing an order as a guest** (System test set #2) One of our system's features is the ability to order menu items without creating an account. You will be able to order just like if you placed an order as a member except you will not have rewards.

Data Management Strategy:

Databases needed:

We have decided to use an SQL database because we have a lot of structured data that needs to be interconnected. Although the SQL database type has a rigid structure, the product that we have designed requires minimal flexibility. Any adjustments that may be necessary in the future would be very minimal and feasible to make to a SQL schema. The availability of foreign keys in SQL-style databases is also very important to our functionality, as we have multiple data columns that need to interact with each other. First, we will need a Users' Database. This will be an individual database in order to keep the user's information private and limit hacking vulnerability. Second, we will have a Menu Items Database for all store-related data including menu items, prices, etc.

Menu Items Database table:

Menu meals table: stores the items we sell (drinks or dishes) under ItemName, the cost of the item under Price, and amount of calories in the item under Calories, and the amount of the item left in stock under Stock, and the list of ingredients under Ingredients.

Diagram with sample data:

Primary key: ID: (Int)	Item Name: (VARCHAR)	Price: (Double)	Calories: (Int)	Stock: (Int)	Ingredients: (VARCHAR, ingredients separated by comma for parsing)
1	Vanilla Latte	4.99	200	50	Coffee, Cream, Vanilla Syrup
2	Bagel	3.99	160	30	Cream Cheese
3	Chocolate Frap	6.99	250	80	Milk, Chocolate Chips, Whip Cream, Chocolate syrup

User Database tables:

User Table: Stores Administrator privileges, Username, Password, card number, and rewards points. All fields except the primary key and rewards points will need to be encrypted. The primary key of this table will be accessed through a foreign key by the UsersItems table. This way the UserItems table is connected to the username of a specific user.

Diagram with sample data:

Primary Key: UserId: (Int)	Admin: (boolean)	Username: (VARCHAR, encrypted)	Password: (VARCHAR, encrypted)	Card Number: (Int, encrypted)	RewardsPoints: (Int)
1	false	StoreEnjoyer12	goodpass145!	2134857384294	23
2	false	CoffeeLovee	anextra$hot	9288074723801	0
3	false	imaBarista4sho	@DKafe	0487559233177	120

Users Items table: Stores a primary key in order to have multiple orders assigned to one customer. The UserId column references the user table and will store the id of the user that is associated with the order. The MenuItemID column references the Menu Item table and stores the id of the menu item that the user has “favorited”. This organization allows us to have users favorite more than one item, and have them all stored in a single table.

Diagram with sample data:

Primary Key: UserItemId: (Int)	UserId: Foreign Key: References UserId in User table (Int)	MenuItemID: Foreign Key: References Menu Item table (Int)
1	1	2
2	1	3
3	2	2