Software Design Specification

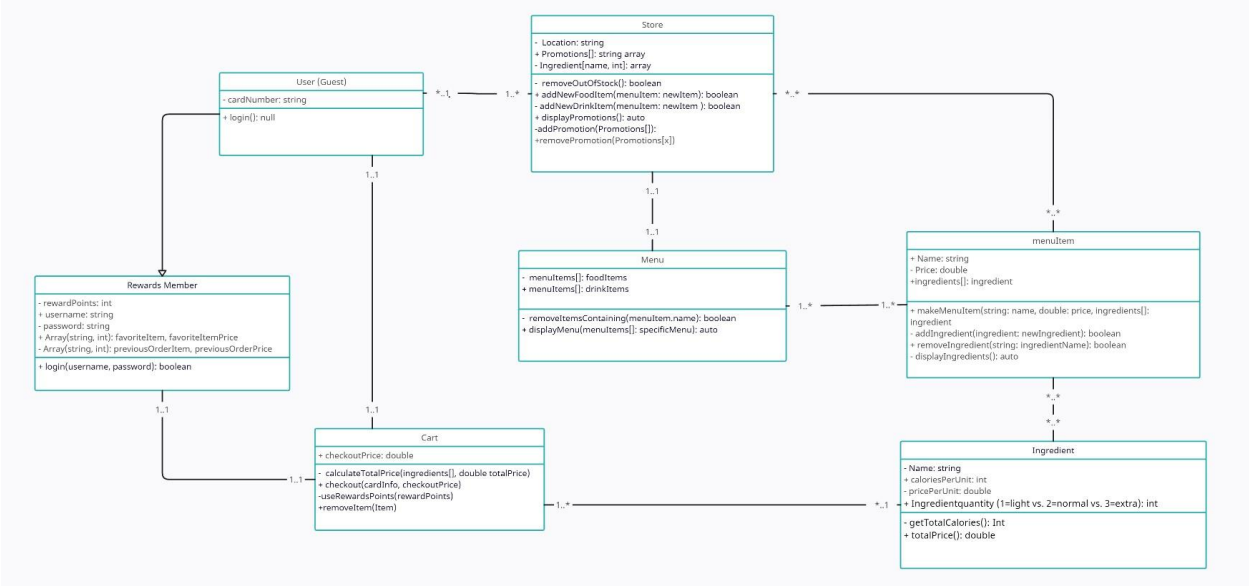
by Chaz Gabelman, Eric Soto,
Gerardo Reza, and Luke Patterson
(Group 9)

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. Users are able to add existing items to their cart, or customize their own items using the Cafe's listed ingredients. The software is also capable of distinguishing which items are in stock depending on the location they are ordering from. There is a menu page listing all items and ingredients for customers to add to their shopping cart, a shopping cart page to review your order and remove any items you don't want anymore, and a payment page to input name/card information and process payment. The software also contains a rewards system that takes in how many points members have accumulated, and allows them to use their points in the checkout page.

Classes



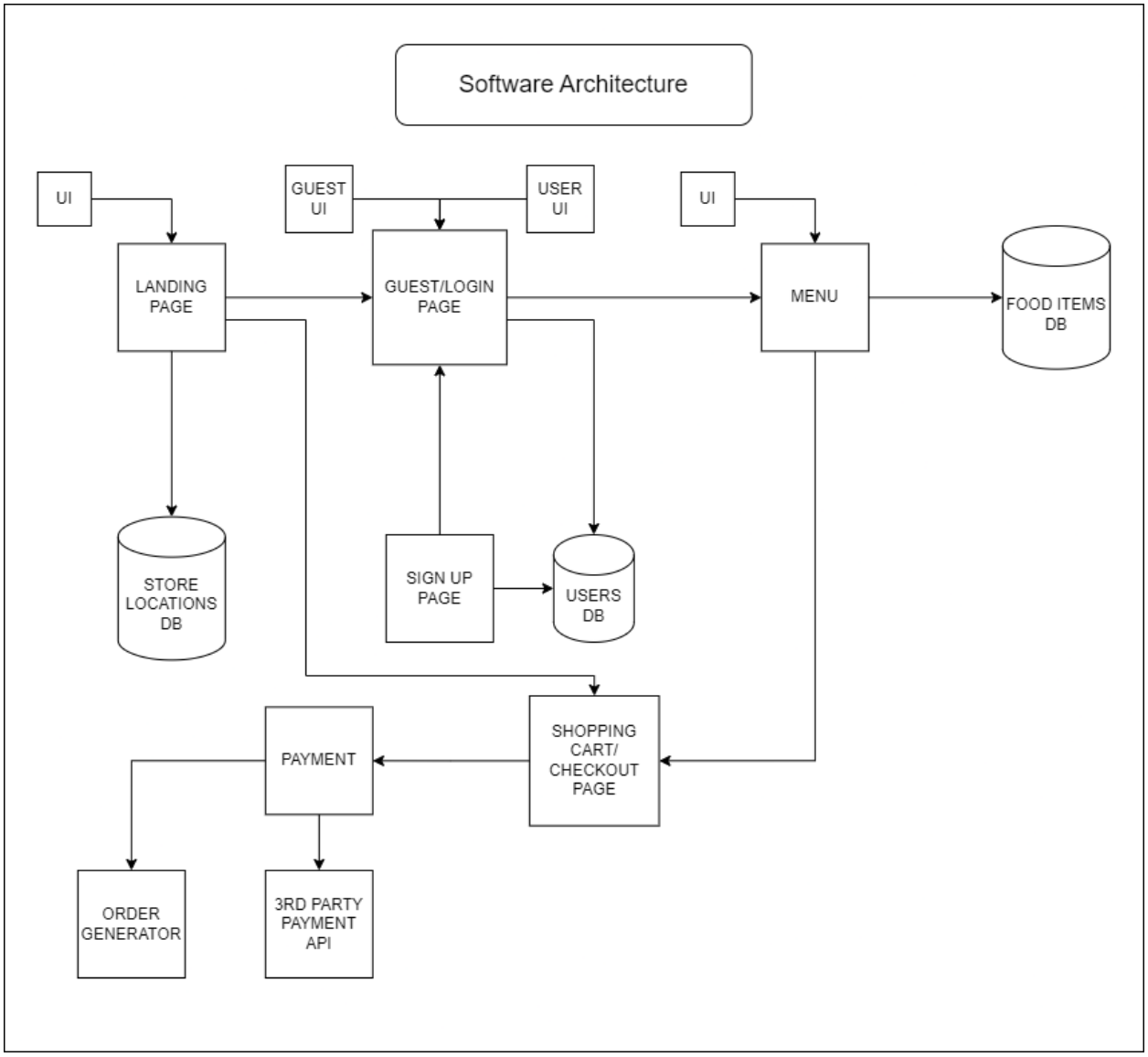
Class Descriptions

- User (Guest)
    - Attributes:
        - Has a string type attribute for Card Number, which would be sent to the Cart class in order to complete the checkout() method.
    - Methods:
        - Has a Login() method with no parameter to automatically login a guest
- Rewards Member
        - Subclass of the User class, inherits attributes and methods from User class
    - Attributes:
        - Has integer type attribute for RewardsPoints, calculates how many points the rewards member has for discounts
        - Has a string type attribute for Username and Password, used for logging into their rewards account
        - Has an array that takes in a string and integer for each element, string representing their favorite item, integer representing the price correlating to that item
        - Has another array that takes in a string and integer for each element, string representing their previous order item, and integer representing the price correlating to that item
    - Methods:
        - Has a login method, that takes in the Username and Password attributes as parameters, uses them to verify the rewards members account and login
- Store
    - Attributes:
        - Location string: used to display the store's address on the website
        - Promotions array: array of strings, used for storing relevant promotions
        - Stock array: array of integer-string pairings which represents the current stock of each ingredient at the store.
        - Menu object: stores the food and drink items the store offers.
    - Methods:
        - removeOutOfStock() uses the stock array to determine which ingredients are out of stock. It then iterates through the menu items present on each menu and removes the menuItems for which that ingredient is included.
        - addNewFoodItem(menuItem) takes in a menuItem and adds it to the foodMenu object, returns true if successful, return false if the menuItem already exists on the food menu.
        - addNewDrinkItem(menuItem) takes in a menuItem and adds it to the drinkMenu object, returns true if successful, returns false if the menuItem already exists on the drink menu
        - displayPromotions() displays promotions from the promotions array

- Menu
  - Attributes:
    - foodItem array: Array of menuItems which stores all available food items
    - drinkItem array: Array of menuItems which stores all available drink items
    - ingredientName: name of a specific ingredient
    - itemName: name of a specific menu item
    - AddExtra(ingredientName): allows to add extra quantity of a specified ingredient
    - AddLight(ingredientName): allows to remove a certain amount of a specified ingredient
  - Methods:
    - getMenuItem(String itemName): Returns a menuItem object present within either the foodItem or drinkItem array containing the given name
    - removeItemsContaining(String ingredientName) takes in string containing the name of an ingredient and removes any menu item in both the food and drink item arrays that contain an ingredient with the same name
    - displayMenu() takes MenuItems array from MenuItem class and displays it
    - AddExtra(ingredientName): allows to add extra quantity of a specified ingredient
    - AddLight(ingredientName): allows to remove a certain amount of a specified ingredient
    - getTotalCalories() provides an integer showing how many calories the specified item contains (caloriesPerUnit * units)
    - getTotalPrice() provides a double showing how much money the ingredient costs (pricePerUnit * units)
- Menu Item
  - Attributes:
    - String ItemName: Name of the menu item
    - Double price: Price of the menu item
    - Ingredients array: An array of ingredient objects within the menu item.
  - Methods:
    - makeMenuItem(String ingredientName, Double price, ingredients[] ingredients) menuItem constructor which takes in a name, a price, and an array of ingredients
    - addIngredient(Ingredient ingredient) takes in an ingredient object and adds it to the ingredient array of the current menuItem
    - removeIngredient(String ingredientName) used to remove an ingredient from within the ingredient array with the given name
    - displayIngredients() displays all the names of the ingredients within the current menuItem object.
    - updatePrice() iterates through all the ingredient objects, adding up each ingredients price and then updating the price of the menuItem with the sum

- Ingredient
  - Attributes:
    - Int caloriesPerUnit: The number of calories per unit of the ingredient
    - Double pricePerUnit: The price per unit of the ingredient
    - IngredientName: name of the ingredient
    -
  - Methods:
    - AddExtra(ingredientName): allows to add extra quantity of a specified ingredient
    - AddLight(ingredientName): allows to remove a certain amount of a specified ingredient
    - getTotalCalories() provides an integer showing how many calories the specified item contains (caloriesPerUnit * units)
    - getTotalPrice() provides a double showing how much money the ingredient costs (pricePerUnit * units)
- Cart
  - Attributes:
    - Double price: Stores the current price of all the items within the cart
    - menuItem array: Stores the current menu items in the cart
  - Methods:
    - addItem(menuItem item): Adds a menu item object to the menuItem array, and updates the price to reflect the updated total
    - getPrice(): Returns the value stored in the price variable
    - updatePrice(Double newPrice): Changes the value stored in price to the newPrice
    - checkOut(cardInfo, checkoutPrice) receives the user's card information and the checkout price to charge the customer for their order
    - UseRewardsPoints() allows user to apply a discount to their price prior to checkout, depending on the amount of points they have accumulated

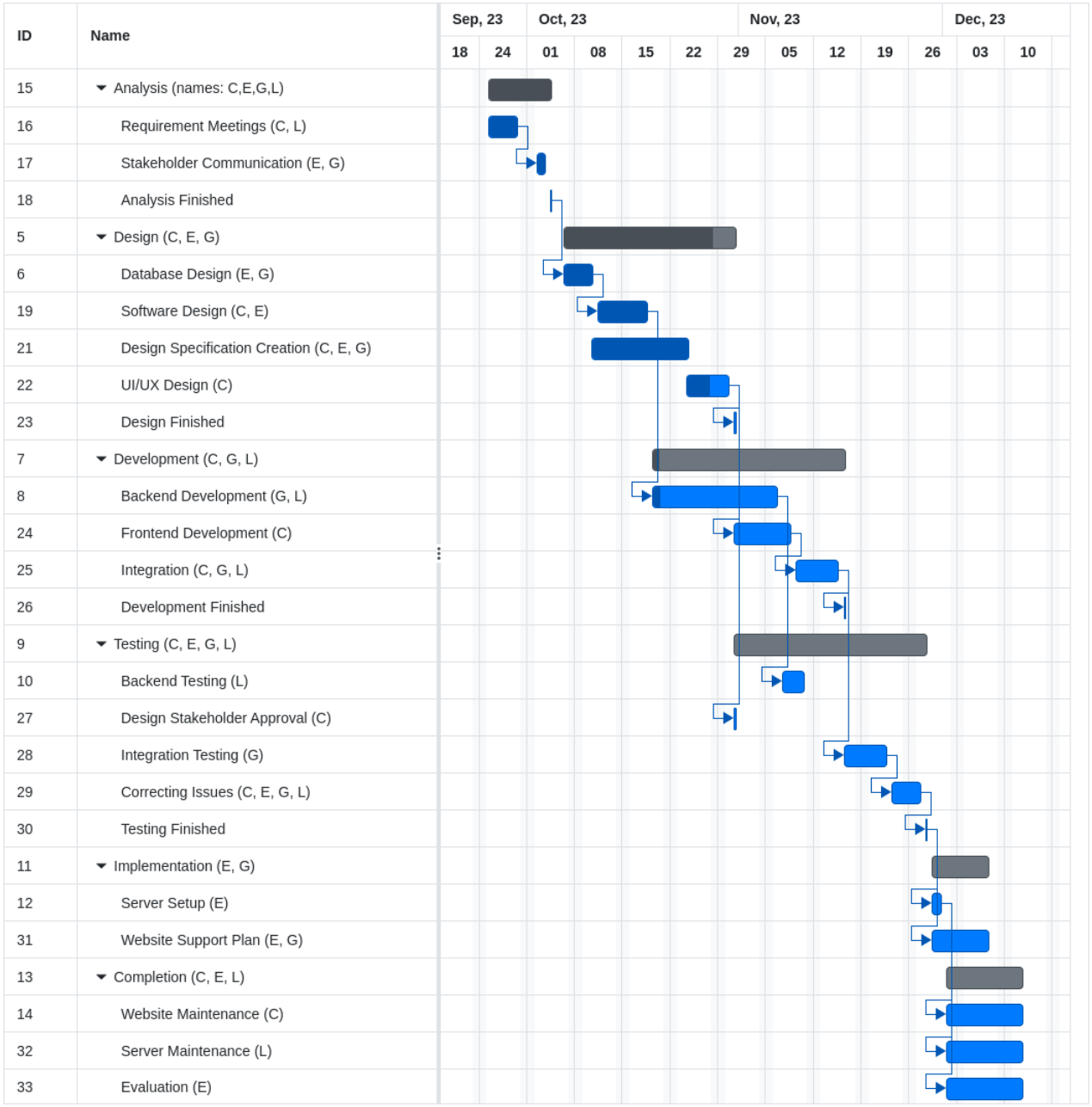
Software Architecture / Component Connections



- The **landing page** is connected to a **store location DB and guest/login page.**
- The **guest/login page** is connected to **sign up page and users DB**.
- The **menu** connects to **food items DB and to shopping cart/checkout page** since you are adding menu items.
- The **landing page** would be connected to a **shopping cart/checkout page** if the user decides to access it from the home page.
- The **shopping cart/checkout page** would be connected to a **payment page** where the guest/user enters or uses saved payment information.
- The **payment page** connects to a **third party payment API** for processing payments.
- The **payment page** would also be connected to an **order generator** that sends an email receipt and order number.

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.)*



Verification Test Plan:

- Unit Testing
  single code “unit” (e.g., method)
  - ➢ 1st Test Set: User Class - log-in

Test Case #	Username	Password	Known Username/Password	Classes Covered	Expected Output
1 Unit	Dog123	IHaveADog12	Dog123, IHaveADog12	User Class	Success
2 Unit	dog123	ihaveadog12	Dog123, IHaveADog12	User Class	Failed, incorrect username and password
3 Unit	Dog123	HaveDog12	Dog123, IHaveADog12	User Class	Failed, incorrect password
4 Unit	mydog123	IHaveADog12	Dog123, IHaveADog12	User Class	Failed, incorrect username

  - ➢ 2nd Test Set: Rewards Member Class - login

Test Case #	Member	Check Rewards	Existing Rewards	Coupon	Classes Tested	Expected Output
1 Unit	True	Yes	0	$0	User Class	0 points
2 Unit	False	Yes	0	$0	User Class	Error, user does not exist
3 Unit	Yes	Yes	50	$5	User Class	50 points, $5 off
4 Unit	Yes	Yes	100	$10	User Class	100 points, $10 off

- Integration Testing
  interfaces among integrated units
  - ➢ 1st Test Set: (Adding a drink menuItem to a store’s drink menu)

Test Case #	Method call	Classes tested	Expected output
1 [normal use case]	addDrinkItem(menuItem Drink); Store.drinkMenu.displayMenu();	Store, menuItem,	True (success) drinkMenu should display newly

		menu	added drink
2 [invalid data type inputted]	addDrinkItem(String itemName);	Store, menuItem, menu	ERROR, input type not valid
3 [menuItem object already exists in drinkMenu's menuItem array]	addDrinkItem(menuItem identicalDrink); this.drinkMenu.display();	Store, menuItem, menu	False (menuItem not added, drink already exists in drinkMenu) drinkMenu should be displayed with no changes

➢ 2nd Test Set: (Adding an ingredient to a menu item)

Test Case #	Method call	Classes Tested	Expected Output
1 [normal use]	menuItem.addIngredient(ingredient newIngredient); menuItem.displayIngredients();	menuItem, menu, Ingredient	True (success) Ingredient names should be displayed with new ingredient name present
2 [wrong data type inputted]	menuItem.addIngredient(String name); menuItem.displayIngredients();	menuItem, menu, Ingredient	ERROR, input type not valid
3 [ingredient object being added already exists in the ingredient array]	menuItem.addIngredient(ingredient identicalIngredient); menuItem.displayIngredients();	menuItem, menu, Ingredient	False (failure) Ingredient object already exists in ingredientArray, Ingredient names should be displayed with no changes

- System Testing
  complete system
  ➢ 1st Test Set: (Placing an order as member)

Test Case #	Member	Location	Menu Items	Out of Stock Items	Rewards Points Used	Classes Tested	Expected Output

1 System	Yes	San Diego	Carrot Cake, Milk	Chocolate Cake	Yes	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Success, coupon applied
2 System	Yes	La Mesa	Coffee, Lemon Bar	Almond Scone	No	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Success, no coupon applied
3 System	Yes	Los Angeles	Blueberry Scone	Blueberry Scone	Yes	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Error, menu item is out of stock

➢ 2nd Test Set (Placing an order as guest)

Test Case #	Member	Location	Menu Items selected	Out Of Stock Items	Rewards Points Used	Classes Tested	Expected Output
1 System	No	San Diego	Ham, Cheese, White Bread	Chocolate cake	No	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Success
2 System	No	San Dego	Ham, Cheese, White Bread	None	No	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Error, invalid location
3 System	No	Lakeside	Ham, Cheese, White Bread	None	Yes	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Error, not a rewards member
3 System	No	San Diego	Ham, Cheese, White Bread	Cheese	No	User, Menu, MenuItem, ShoppingCart, Store, Ingredients	Error, menu item out of stock

Test Set Description:

- ➢ **What features are you testing?**
    - **Log-in test** (Unit test set #1): comparing both username and password strings entered by the user with the known username and passwords in the system (tests User class).
    - **Adding drinks to the menu test** (Integration test set #1): Our first integration feature we are testing is the ability for the store to add drink menu items to their drink menu.
    - **Adding ingredients to the menu item** (Integration test set #2): Our second integration feature we are testing is the ability for ingredients to be added to a menu item.
    - **Rewards test** (Unit test set #2): system checks if the guest is a member and if it has existing rewards, applying discounts corresponding to the available rewards of the user (tests User class).
    - **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 the features that our system has 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.

- ➢ **What are the test sets/what do they do?**
    - **Log-In:** our application includes a system where a customer can create an account, setting a password and username of their choice; these being **string** data types. Once an account is created, it is stored in a database. Each time a user tries to log into their account, the system will compare the user's input with the information stored in the database to see if they match. This test checks for incorrect username, incorrect password, and the combination of both.
    - **Adding drinks to the menu:** The data type used here are an **array of strings**, which stores specific drink items from the menu, allowing employees to either add or remove to the menu any drink that is newly in-stock. A database containing this information is used to compare if the drink that is being added is already on the menu or not, which will throw an error if it is. It will also check if wrong data types are used, and won't be accepted as input.
    - **Adding ingredients to the menu**: Similarly to how drinks are added to the menu, an **array of strings** contains the names of the ingredients that

can be either available or unavailable at a certain time. If an ingredients is tried to be added to the menu, the input will first be compared to what is stored in the database; if the wrong data type is used, an error is thrown, if an already-existing ingredient is on the menu, no changes should be made, and an ingredient name which is not yet on the list will be added to the array and successfully display on the menu.

- **Rewards:** One of the perks of having an account is that points can be earned with each purchase.For each 50 points earned, a $5 coupon is given to the user.. The test here checks if the customer is a member, and if it is, it checks for coupons and discounts that have been earned before. The data type used here are **integers.**
- **Guest placing an order:** Guest users will have the ability to input their card information into a **string** attribute which will be sent to the **Cart** class to complete the **checkout()** method when they place an order. When they place an order one or more menu items will be placed in the Cart class and they will be ready to checkout. If a menu item is out of stock then an error will appear before the payment process occurs. Guest users do not have access to a rewards balance.
- **Member placing an order:** Member users order items just like guests except they have the ability to access and use their rewards they accumulate from previous purchases. This is only the case if the user logs in successfully from the **login** method in the Rewards Member class. Once the member is ready to checkout with a cart full of menu items they will have the option to use their coupons. Members get a $5 dollar coupon for every 50 reward points.

➢ **How do your selected tests cover the targeted feature?**

- Each test compares user input with the corresponding data types stored in their respective databases; for the user and member class, the input is compared and throws an error if the input information is non-existent or doesn't match what's in the database. For menu items and drinks, if the input is the appropriate data type, but it is non-existent in the database, then the item in question will be added to the database and then displayed on the menu; this results in lower probabilities of encountering an error on each feature.

Data Management Strategy:
Mysql probably

Databases needed:

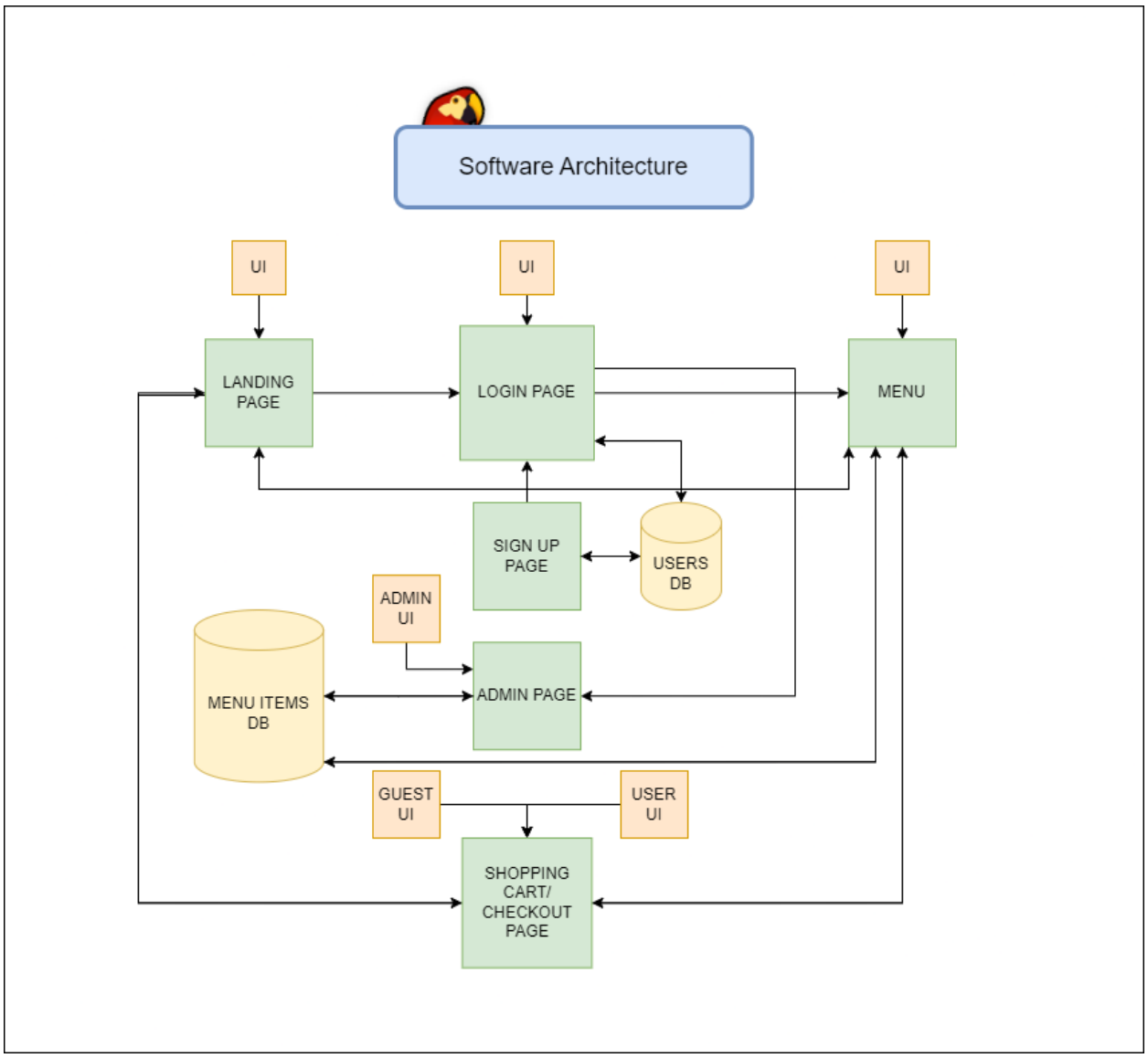
- Users database, to keep their information private and limit hacking vulnerability
- Store Location database could be combined with the Food Items database, I don't see a reason they have to be separate and it could make querying easier. We would just have to add extra lines pointing to the merged database.

Tables needed:

- Menu item table
    - Stores individual menu items: their name, price, calories, stock
- Menu meals table: stores the items we sell like drinks or dishes, and would access the "menu item" table through a foreign key
    - This way we don't have to have every individual item as a column in the table with a true/false
    - Unless you have another idea
- User table:
    - Username, Password, Location, card number (encrypted)
        - We would need a way to store their favorite items but I'm not sure what the best way to go about that is (maybe it could be another table in the Users DB that has a foreign key to the Users table so the Users DB would be Users & Favorite Items DB)
- Store table:
    - For individual stores
    - Has location, would access "stock" from "menu item table" through a foreign key, hours maybe?, (insert more here ig)
- If there's any other tables you think we need let me know

Then

Software Architecture Diagram:



NEW CLASS STRUCTURE

(assumption #1: The store is handling payment manually, if somebody wants to figure out how 3rd party payment processors work, go ahead.)
Single store model (only 1 restaurant exists)
User info database:

- Unique ID
- Name
- Password hash
- Access level (customer vs admin)
- Payment info, if applicable
- Rewards points, if applicable

Menu database:

- Unique ID
- ingredientName
- Ingredients
- Calories
- Price
- Picture

➢ Store server interface (can be **modified** by store admins)

- Has location information

- Displays the menu (via accessing the menu database)
- **Edits** the menu (via interacting with the menu database)
- Stores Placed Orders (queue of carts)
- Handles login/account registration (via interacting with the member info database)
- Handles payment (info in cart)

Our program users can be divided into 2 categories: Customers and Administrators

- Customers can be further divided into Guests and Rewards Members
  - All customers will have a cart
    - Guests:
      - Can login (via reading from member info database)
      - Can create an account (via adding to member info database)
      - placeOrder(String name, payment info)
    - Rewards Members:
      - Can logout
      - Can access and edit profile information (contact info, payment info, etc…) via interacting with member info database
      - placeOrder() (name + payment information already on file, updates member info database +1 reward point for every $10 dollars spent, subtracts used rewards points)

Customer places order (adding their name and payment info if necessary) -> calls store's payment method -> if payment goes through, moves cart to the stores placed order queue

- Admins
  - nextOrder() returns next order in the order queue

- Cart
  - Contains customer name (autofilled via member info database, if applicable)
  - Contains customer payment info (autofilled via member info database, if applicable)
  - Contains whether the customer will use their rewards points towards this order (auto false if guest)
  - Contains the number of rewards points being used towards the order (autofilled via member info database if member elects to use their reward points)
  - Stores an array of menu item unique identifiers
    - Customers view the menu and make a selection. Selecting a menu item places its unique identifier (via database) into an array of menu items
  - Stores the price of the items in the cart totaled (taking into account rewards points, each reward point used = -$1)
  - addMenuItem(unique identifier) (adds the identifier to the menu item array and updates price via interacting with menu database)