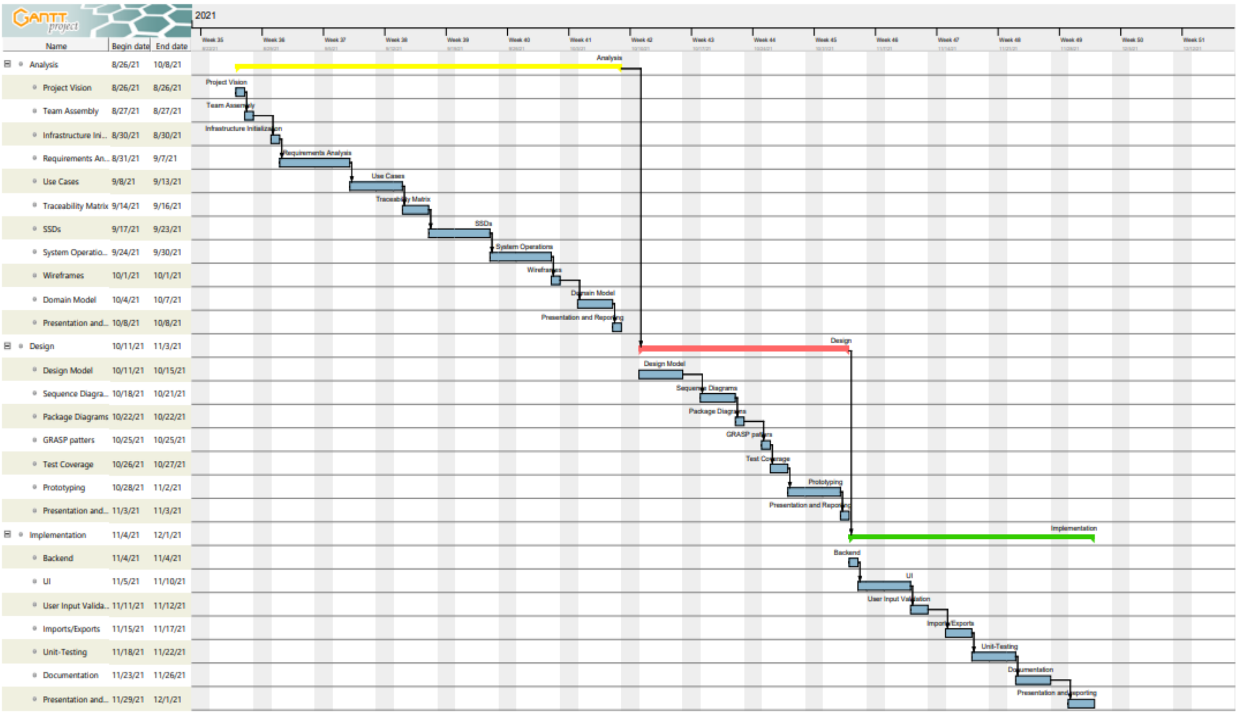
Baylor Burgers (Iteration II Documentation)

By Francis Boyle, Patrick Boyle, Timmy Frederiksen, Johnny Acosta

Reminder of Project Vision

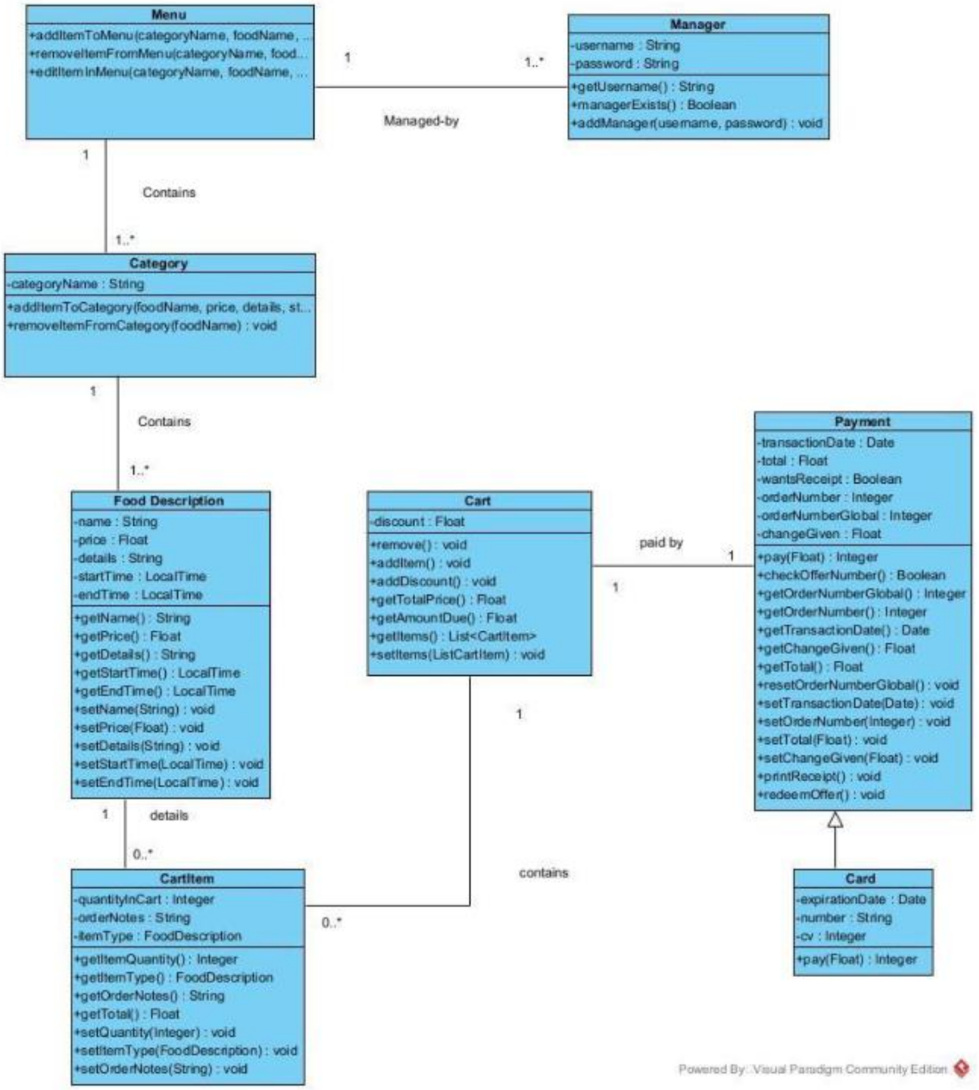
Our vision for Baylor Burgers is a self-order kiosk for restaurants. We plan to implement features such as allowing customers to add, delete and customize items for their order. We also plan to allow for ownership/managers to change the menu from the system. The system will also facilitate payment and allow customers to redeem deals. The system will also place orders to the restaurant so the employees can begin preparing the order as soon as possible. Our top priority is providing a simple, easy-to-understand user interface and allow for the quickest possible order time, streamlining the ordering process for the customer and the restaurant.

GANTT Diagram



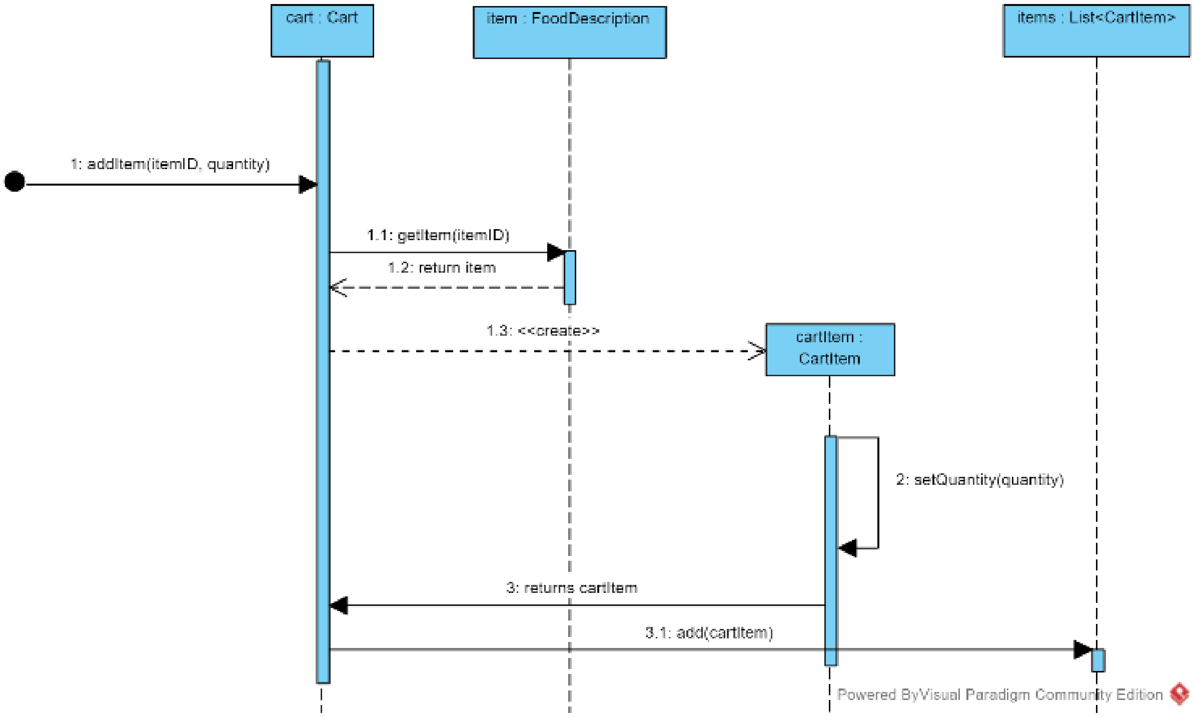
Name	Begin date	End date
⊟ Analysis	8/26/21	10/8/21
Project Vision	8/26/21	8/26/21
Team Assembly	8/27/21	8/27/21
Infrastructure Ini...	8/30/21	8/30/21
Requirements An...	8/31/21	9/7/21
Use Cases	9/8/21	9/13/21
Traceability Matrix	9/14/21	9/16/21
SSDs	9/17/21	9/23/21
System Operatio...	9/24/21	9/30/21
Wireframes	10/1/21	10/1/21
Domain Model	10/4/21	10/7/21
Presentation and...	10/8/21	10/8/21
⊟ Design	10/11/21	11/3/21
Design Model	10/11/21	10/15/21
Sequence Diagra...	10/18/21	10/21/21
Package Diagrams	10/22/21	10/22/21
GRASP patters	10/25/21	10/25/21
Test Coverage	10/26/21	10/27/21
Prototyping	10/28/21	11/2/21
Presentation and...	11/3/21	11/3/21
⊟ Implementation	11/4/21	12/1/21
Backend	11/4/21	11/4/21
UI	11/5/21	11/10/21
User Input Valida...	11/11/21	11/12/21
Imports/Exports	11/15/21	11/17/21
Unit-Testing	11/18/21	11/22/21
Documentation	11/23/21	11/26/21
Presentation and...	11/29/21	12/1/21

Class Diagram

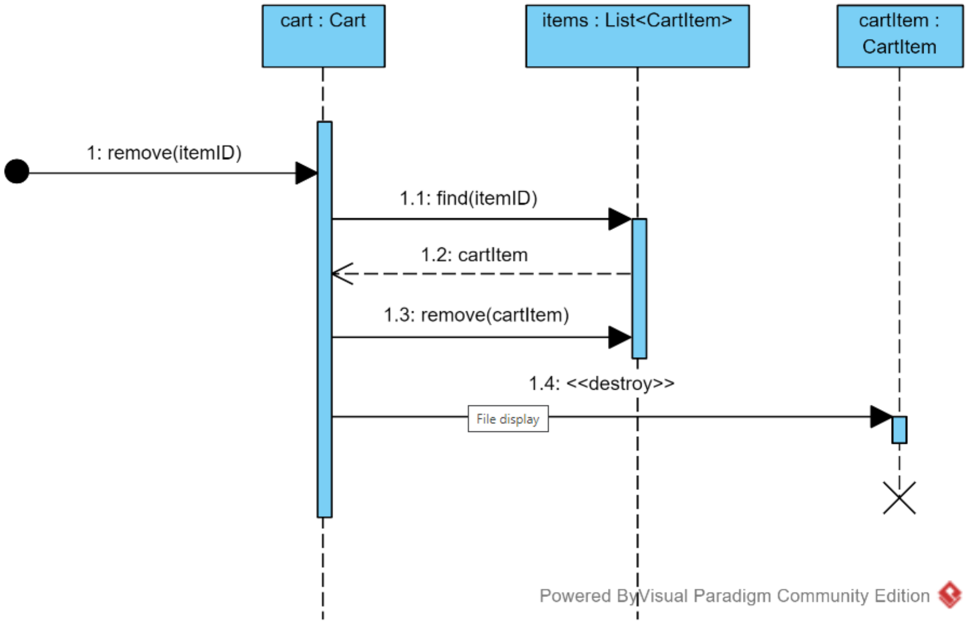

Menu
+addItemToMenu(categoryName, foodName, ...
+removeItemFromMenu(categoryName, food...
+editItem InMenu(categoryName, foodName, ...
1
1..*
Managed-by
Manager
-username : String
-password : String
+getUsername() : String
+managerExists() : Boolean
+addManager(username, password) : void
1
Contains
1..*
Category
-categoryName : String
+addItemToCategory(foodName, price, details, st...
+removeItemFromCategory(foodName) : void
1
Contains
1..*
Food Description
-name : String
-price : Float
-details : String
-startTime : LocalTime
-endTime : LocalTime
+getName() : String
+getPrice() : Float
+getDetails() : String
+getStartTime() : LocalTime
+getEndTime() : LocalTime
+setName(String) : void
+setPrice(Float) : void
+setDetails(String) : void
+setStartTime(LocalTime) : void
+setEndTime(LocalTime) : void
Cart
-discount : Float
+remove() : void
+addItem() : void
+addDiscount() : void
+getTotalPrice() : Float
+getAmountDue() : Float
+getItems() : List<CartItem>
+setItems(ListCartItem) : void
paid by
1
1
Payment
-transactionDate : Date
-total : Float
-wantsReceipt : Boolean
-orderNumber : Integer
-orderNumberGlobal : Integer
-changeGiven : Float
+pay(Float) : Integer
+checkOfferNumber() : Boolean
+getOrderNumberGlobal() : Integer
+getOrderNumber() : Integer
+getTransactionDate() : Date
+getChangeGiven() : Float
+getTotal() : Float
+resetOrderNumberGlobal() : void
+setTransactionDate(Date) : void
+setOrderNumber(Integer) : void
+setTotal(Float) : void
+setChangeGiven(Float) : void
+printReceipt() : void
+redeemOffer() : void
1
1
details
0..*
contains
CartItem
-quantityInCart : Integer
-orderNotes : String
-itemType : FoodDescription
+getItemQuantity() : Integer
+getItemType() : FoodDescription
+getOrderNotes() : String
+getTotal() : Float
+setQuantity(Integer) : void
+setItemType(FoodDescription) : void
+setOrderNotes(String) : void
0..*
Card
-expirationDate : Date
-number : String
-cv : Integer
+pay(Float) : Integer
Powered By: Visual Paradigm Community Edition

Sequence Diagrams

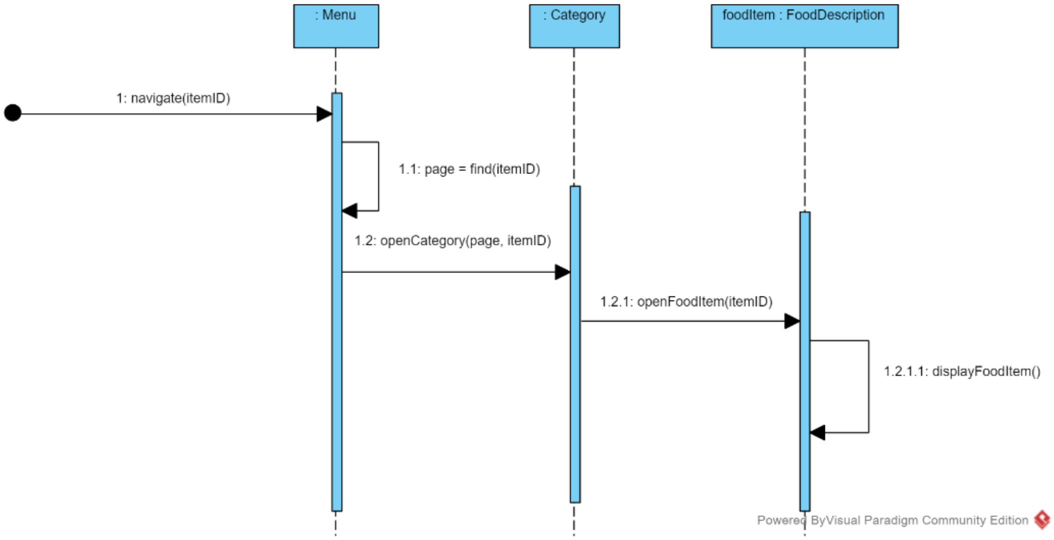
Add Item To Cart SD



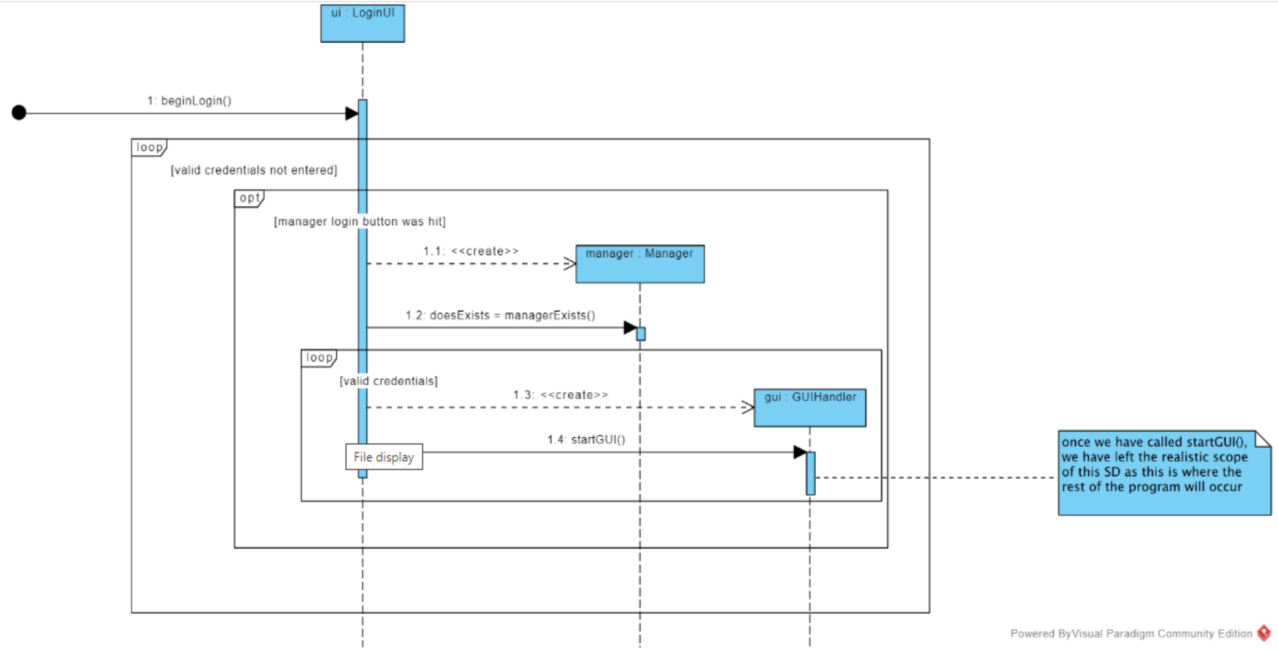
Remove Item From Cart SD



Searh For Item SD

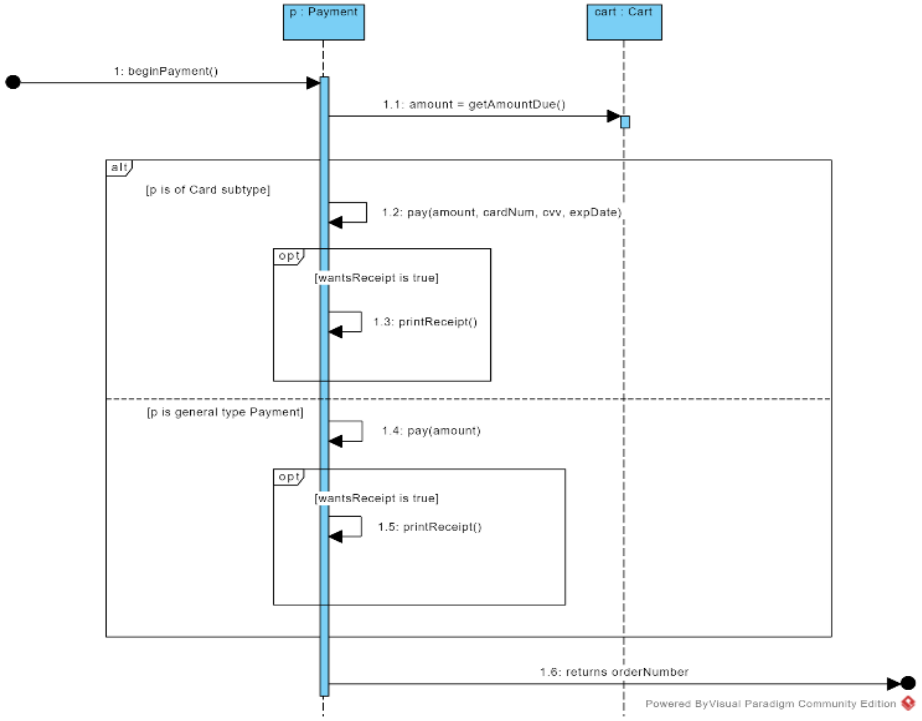

: Menu
: Category
foodItem : FoodDescription
1: navigate(itemID)
1.1: page = find(itemID)
1.2: openCategory(page, itemID)
1.2.1: openFoodItem(itemID)
1.2.1.1: displayFoodItem()
Powered By Visual Paradigm Community Edition

Manager Login SD

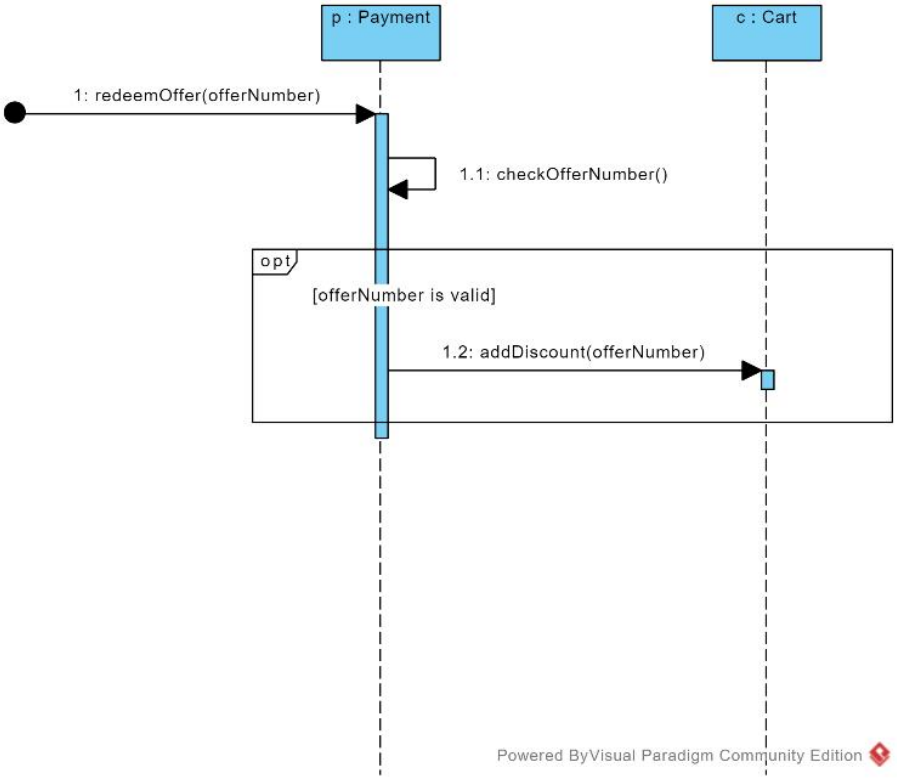


once we have called startGUI(), we have left the realistic scope of this SD as this is where the rest of the program will occur

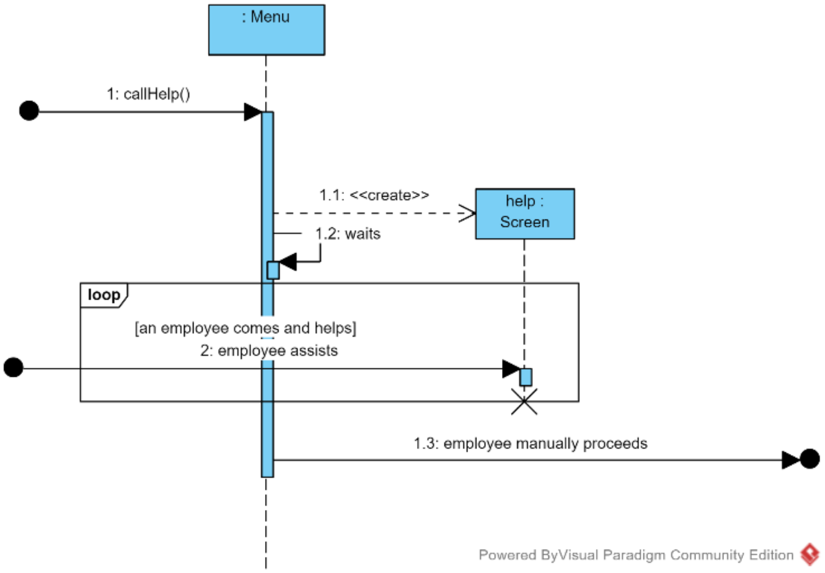
Pay SD



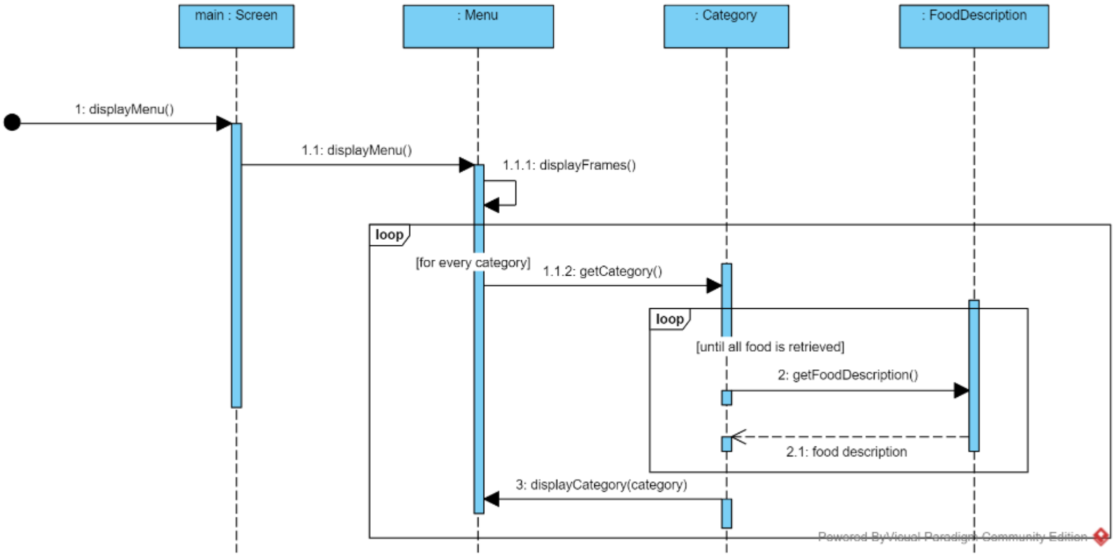
Redeem Offer SD


p : Payment
c : Cart
1: redeemOffer(offerNumber)
1.1: checkOfferNumber()
opt
[offerNumber is valid]
1.2: addDiscount(offerNumber)
Powered ByVisual Paradigm Community Edition

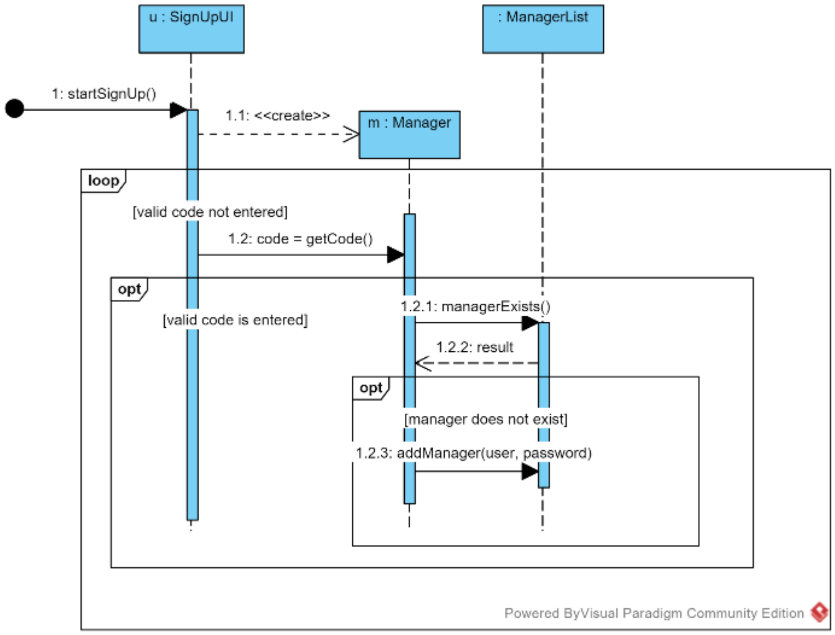
Call Help SD


: Menu
1: callHelp()
1.1: <<create>>
help : Screen
1.2: waits
loop
[an employee comes and helps]
2: employee assists
1.3: employee manually proceeds
Powered ByVisual Paradigm Community Edition

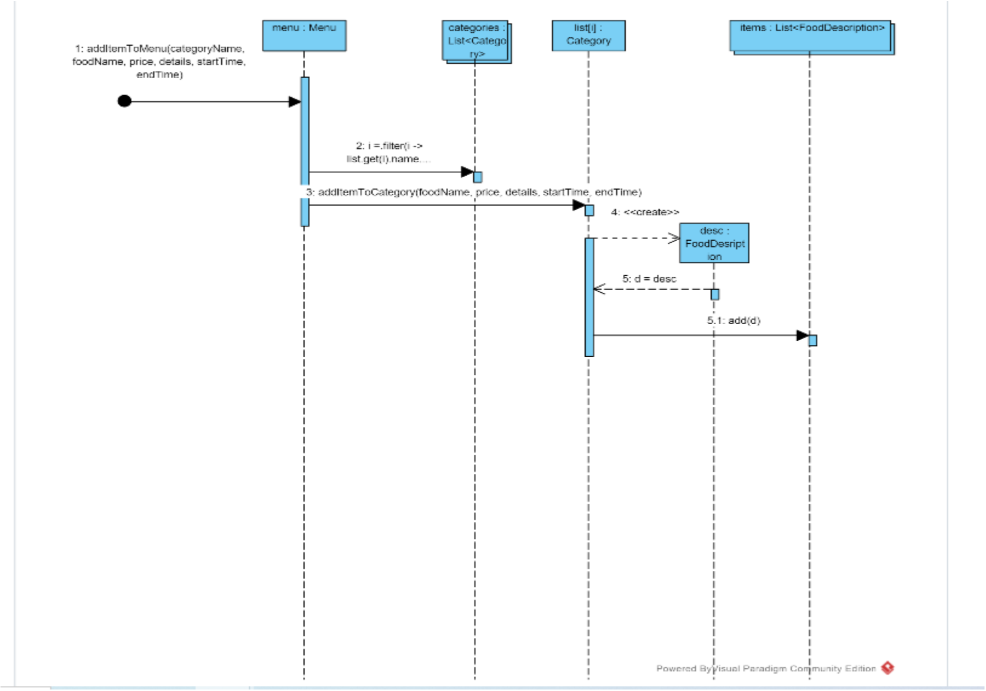
Display Menu SD



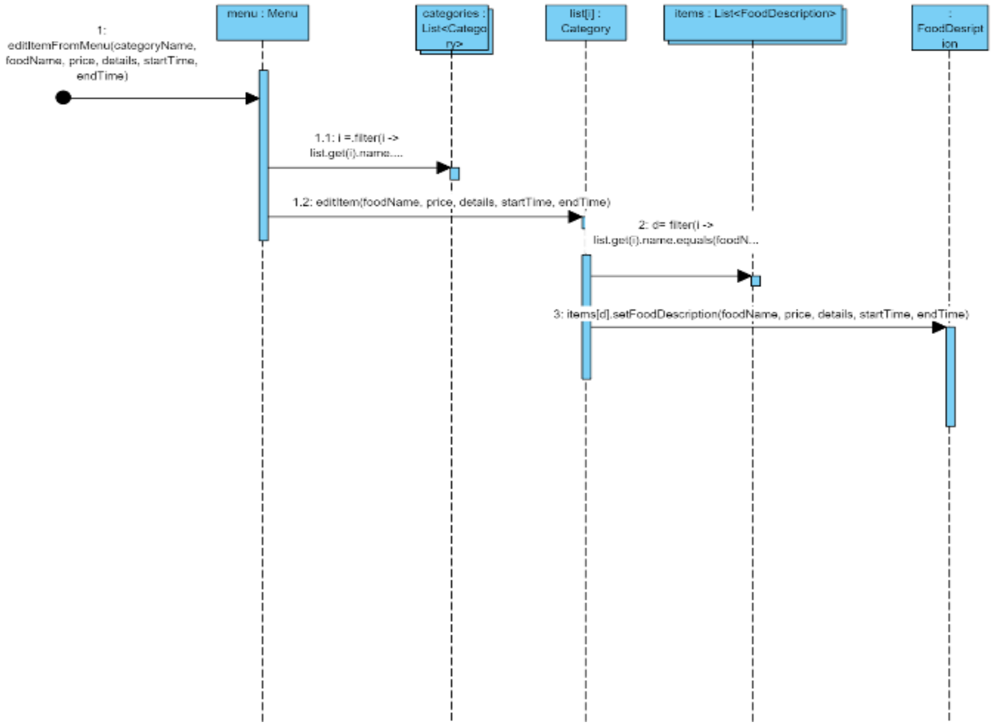
Manager Sign-Up SD



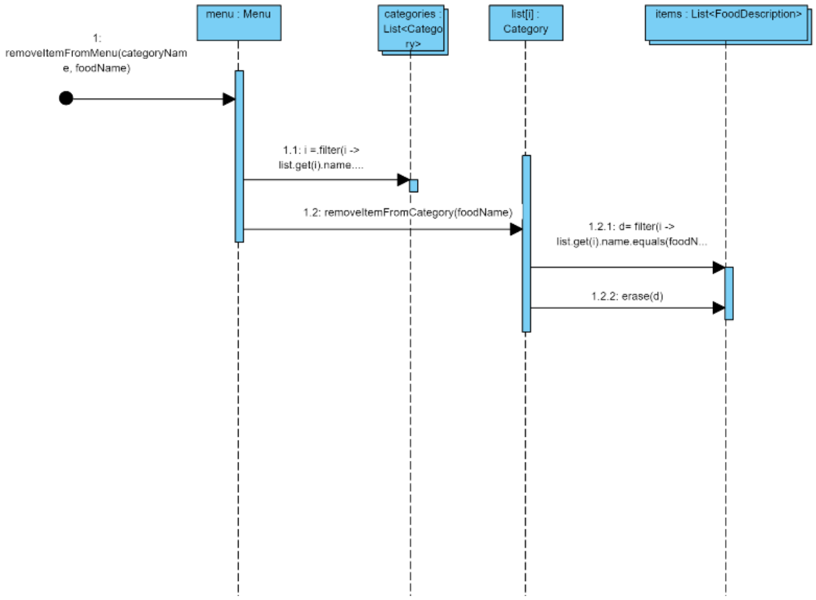
Add Item To Menu SD


menu : Menu
categories : List<Category>
list[i] : Category
items : List<FoodDescription>
1: addItemToMenu(categoryName, foodName, price, details, startTime, endTime)
2: i =.filter(i -> list.get(i).name....
3: addItemToCategory(foodName, price, details, startTime, endTime)
4: <<create>>
desc : FoodDesription
5: d = desc
5.1: add(d)
Powered ByVisual Paradigm Community Edition

Edit Item From Menu


menu : Menu
categories : List<Category>
list[i] : Category
items : List<FoodDescription>
: FoodDescription
1: editItemFromMenu(categoryName, foodName, price, details, startTime, endTime)
1.1: i =.filter(i -> list.get(i).name....
1.2: editItem(foodName, price, details, startTime, endTime)
2: d= filter(i -> list.get(i).name.equals(foodN...
3: items[d].setFoodDescription(foodName, price, details, startTime, endTime)

Remove Item From Menu


menu : Menu
categories : List<Category>
list[i] : Category
items : List<FoodDescription>
1: removeItemFromMenu(categoryName, foodName)
1.1: i =.filter(i -> list.get(i).name....
1.2: removeItemFromCategory(foodName)
1.2.1: d= filter(i -> list.get(i).name.equals(foodN...
1.2.2: erase(d)

GRASP

Listed below the concepts in our design class diagram:

Menu

This class has not been seriously implemented in our prototype, but we can expect this class to be responsible for keeping track of the different Food Description objects available at a certain location/time/etc by using category as an indirection between the two to lower coupling. Thus, this class would be considered an Information Expert.

Category

This class has not been implemented yet, but we can expect it to be an information expert. Also making these categories allows for a more understandable code design, helping with high cohesion and low coupling.

Food Description

This class is responsible for tracking the attributes of a particular menu item, thus making this an Information Expert. Also, we chose to separate the Food Description from the Cart Items so that we would not have to track multiple copies of the same data from the Food Description class. This would be an example of high cohesion as we are separating data into multiple classes to reduce redundancy.

Cart Item

This class is responsible for keeping track of how many items of a particular Food Description a customer has ordered as well as tracking the special requests placed by customers; therefore, this is an example of an Information expert. Also, we chose to separate the Food Description from the Cart Items so that we would not have to track multiple copies of the same data from the Food Description class. This would be an example of high cohesion as we are separating data into multiple classes to reduce redundancy.

Cart

This class is responsible for keeping track of the items in a customer's order while they have not finished adding new items or are still ordering. This is an example of an information expert as it keeps these items accounted for.

Manager

This class is responsible for the basic login system and is mostly a class so that our GUI classes don't have to track it. This is an example of pure fabrication as we did not ***need*** this class, but we decided it would be easier to understand if it were separate.

Payment

This Payment class is responsible for handling all payment events, however, we decided to separate the card payment out into a subclass of this general type, through polymorphism, to make sure our design is cohesive and supports low coupling.

Card

This class is responsible for handling Payment events made with Card, we chose to separate this from the general type of Payment because it will work fundamentally differently and thus demonstrates higher cohesion.

LoginUI and other unimplemented UI classes

These classes are responsible for handling and managing user interaction with the system, and thus can be identified as Controller classes, these classes are how the user will interact with the menu and other objects/classes within our system. This also can be described with the ‘Don’t Talk to Strangers’ GRASP principle as we don’t want the user handling the internals of our system. As we further implement these classes, this may end up being where most of the internal instances of classes are created, which would make some of the classes Creator classes; however, we have yet to implement some of these functionalities thus far and that remains to be seen.

Test Coverage Plan

Add Item to Cart - 0/2

- Check quantity added is a valid quantity
- Check that the cart is updated

Remove Item from Cart - 0/2

- Check quantity to remove from food item is a valid quantity
- Check that the cart is updated after the removal of the quantity of an item, or the complete item.

Search for Item - 0/1

- Check that proper input navigates to right menu item

Manager Login - 0/2

- Check that invalid login credentials don’t gain entrance
- Check that valid credentials do gain entrance

Manager Sign-Up - 0/3

- Check that there aren’t two managers with the same information
- Check that the manager is able to log in afterwards
- Check that the manager is only able to register given the correct credentials

Redeem Offer - 0/5

- Check that invalid code gets ignored
- Check that non-alphanumeric characters are automatically invalid
- Check that valid offer gives correct discount
- Check that discount is applied correctly to price
- Check that NULL code is not fatal to the program

Pay - 0/12

- Check that card payment has valid card number length
- Check that card payment has only numbers
- Check that card payment has valid CVV length
- Check that card payment has only numbers
- Check that card expiration date was given a valid month number
- Check that card expiration date was given a valid year (not expired card)
- Check that card payment is valid given valid parameters
- Check that NULL values for each parameter for card payment does not make the program fail
- Check that a NULL amount for non-card payment does not make the program fail
- Check that if a payment is less than the amount due that the payment is not accepted
- Check that sufficient payment amount for non-card payments is accepted
- Check that the order number is returned correctly and is incrementing the counter

Add Item to Menu - 0/2

- Check that when a manager adds an item it appears with the correct information
- Check that the item goes to the correct category

Delete Item from Menu - 0/1

- Check that the item no longer appears in the menu

Edit Item in Menu - 0/3

- Check that when an item is edited it exists and has the new information

- Check that when items change across categories they pop up in the new category and not in the old category
- Check that items do not appear twice after editing

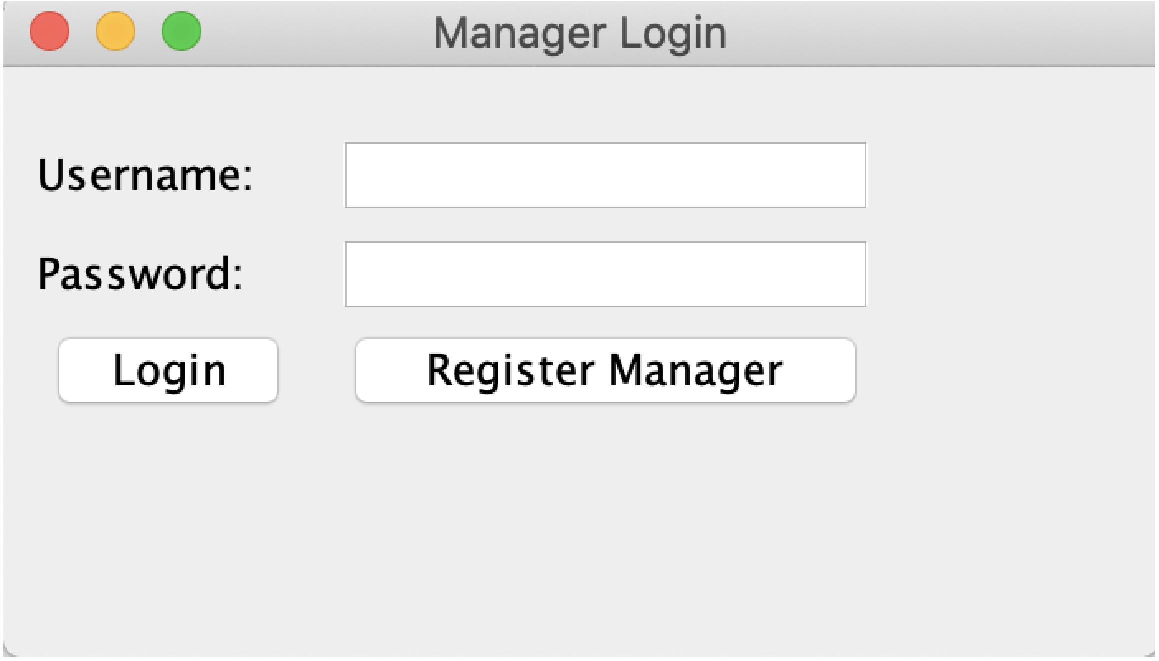
Display Menu - 0/2

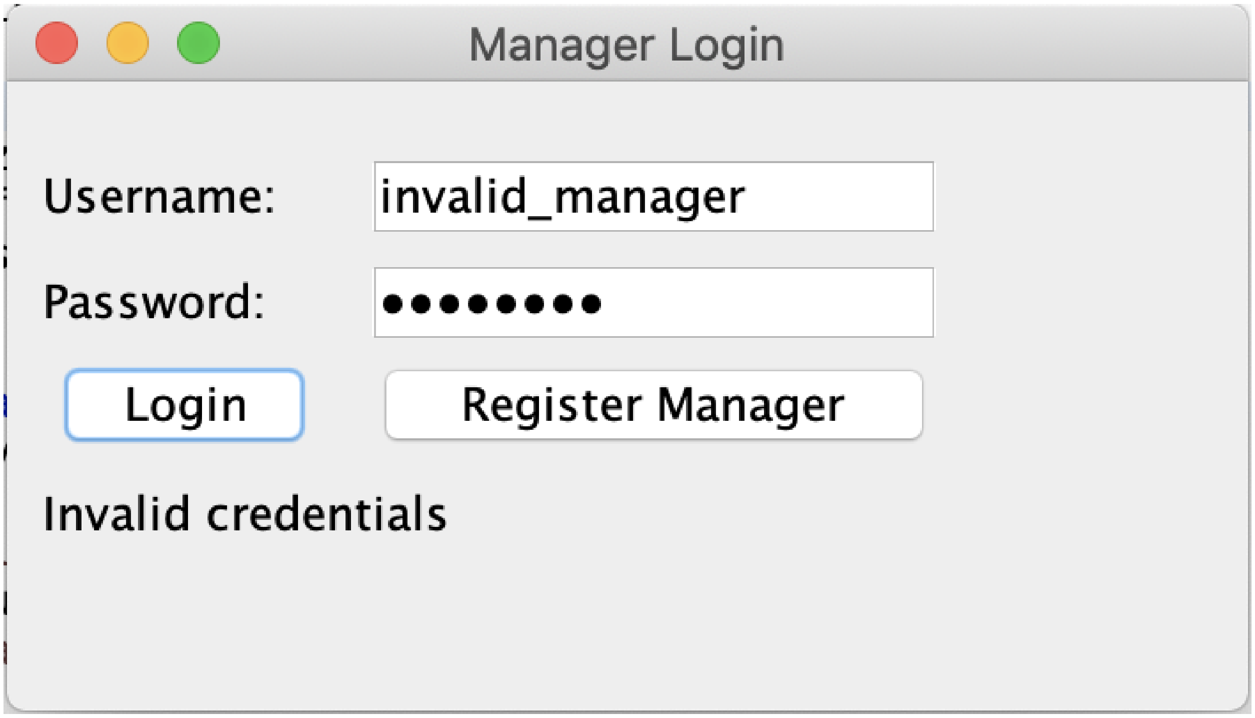
- Check that the menu displays from the start screen
- Check that menu displays when on any other screen of the menu that is not the main menu screen

Call Help - 0/1

- Check that a message shows to indicate help is on the way

USER INTERFACE DEMO IMAGES

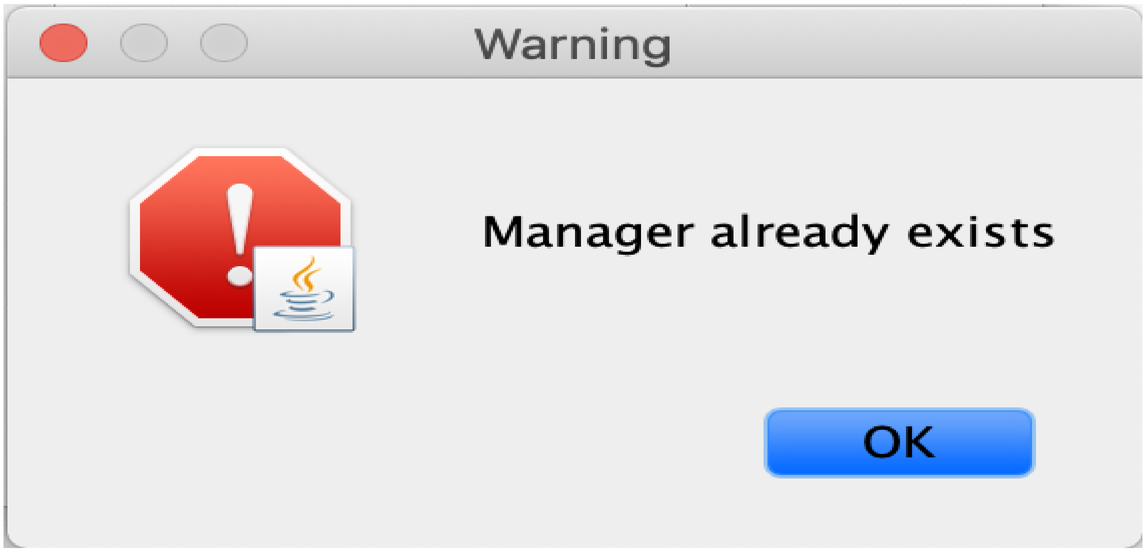




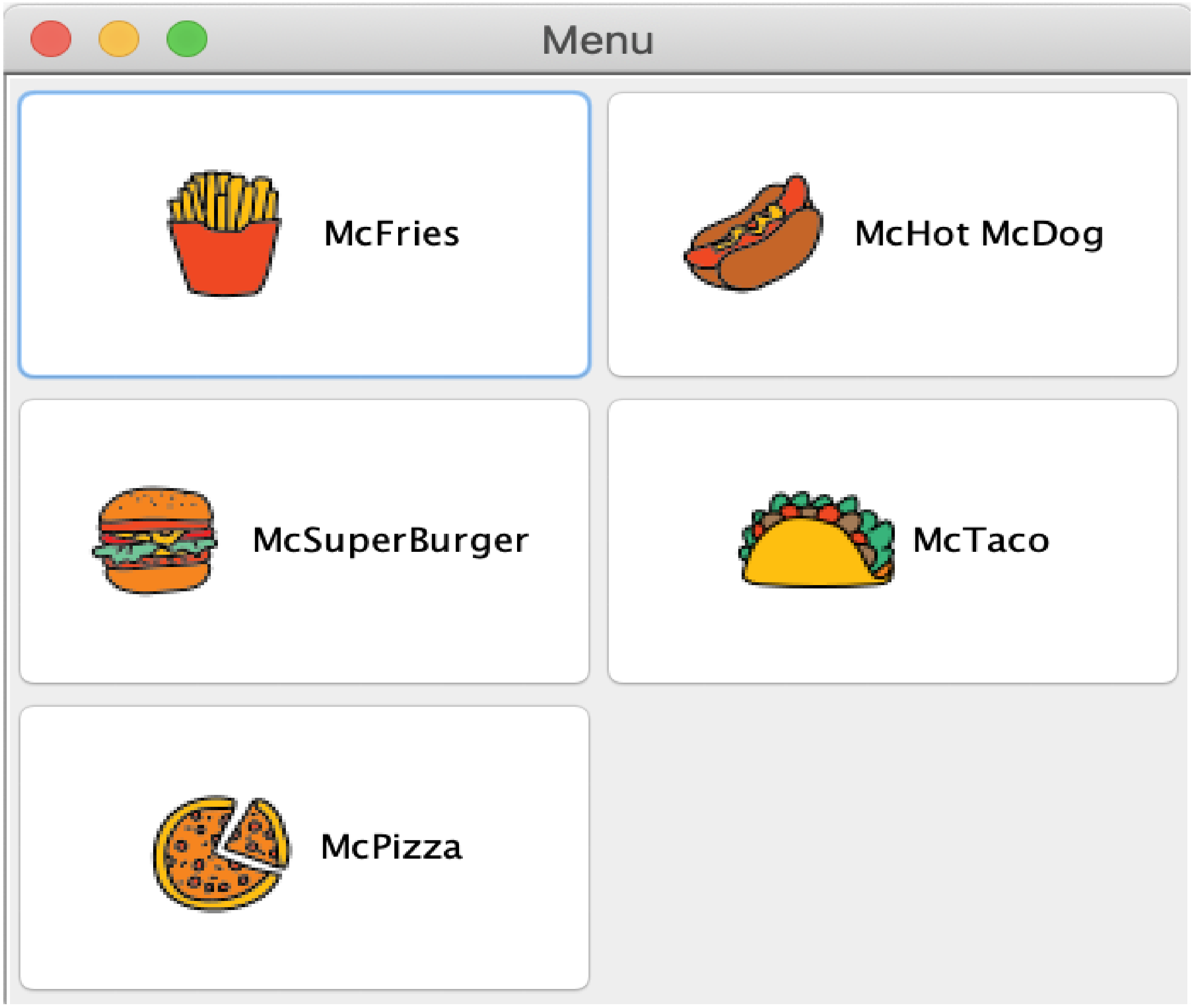
Username: invalid_manager

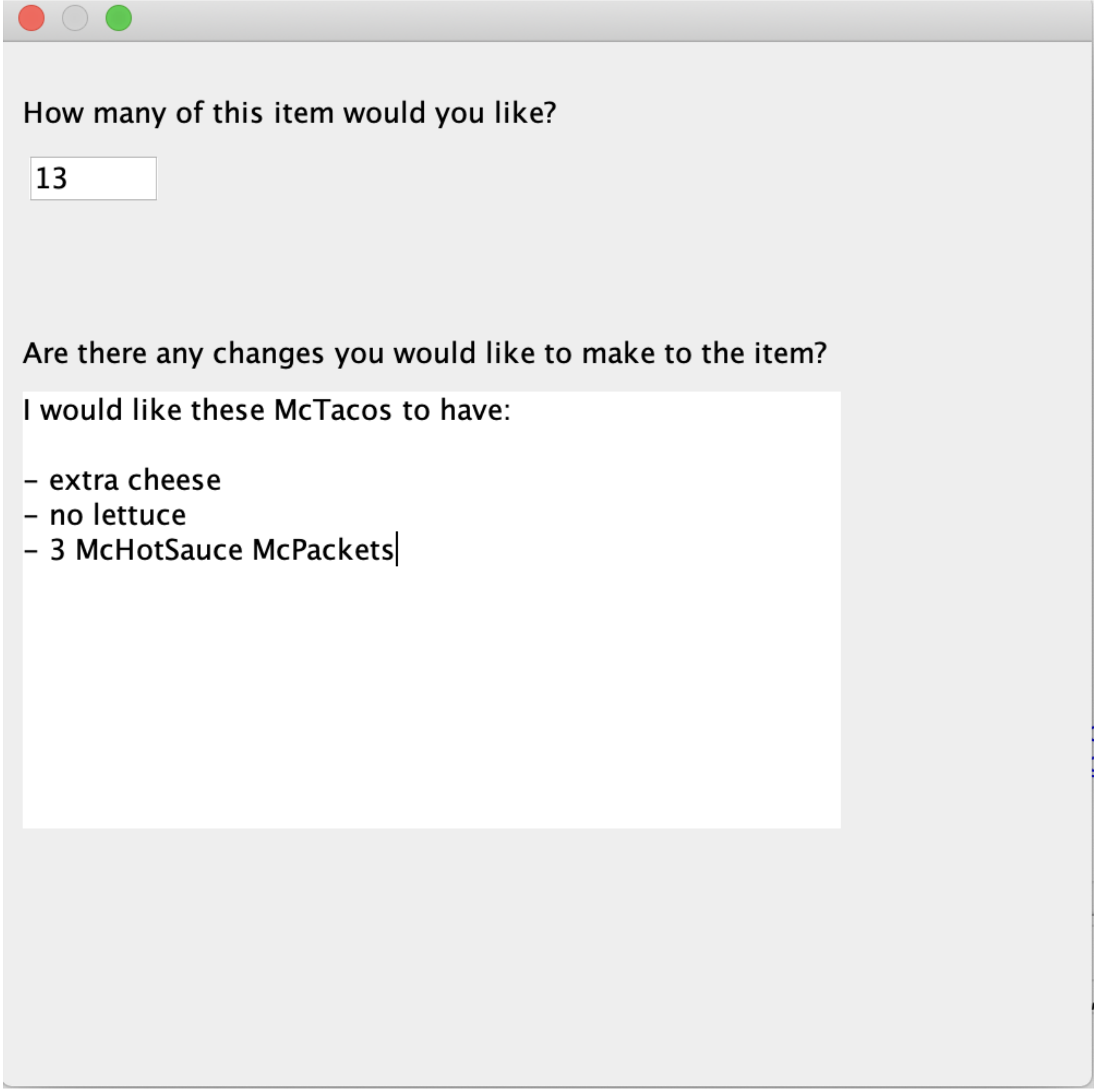
Password: ●●●●●●●●●

Invalid credentials



Manager already exists





How many of this item would you like?

13

Are there any changes you would like to make to the item?

I would like these McTacos to have:

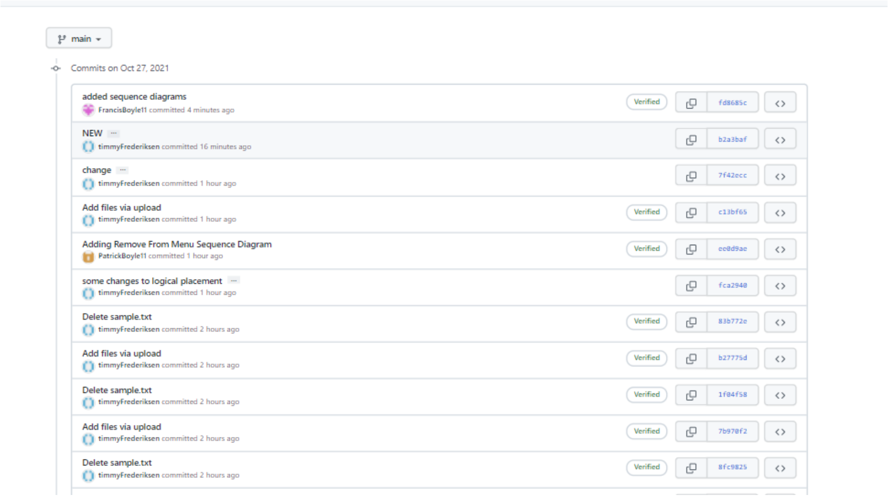
- extra cheese
- no lettuce
- 3 McHotSauce McPackets

Git, Website, Trello Links & Issue Tracking

Website: https://csi3471-kiosk-project.weebly.com

Trello: https://trello.com/b/xjHC0LGq/prowling-bears

GitHub: https://github.com/timmyFrederiksen/GroupProjectFall2021



Commits on Oct 27, 2021

added sequence diagrams
FrancisBoyle11 committed 4 minutes ago
Verified fd8685c

NEW
timmyFrederiksen committed 16 minutes ago
b2a3baf

change
timmyFrederiksen committed 1 hour ago
7f42ecc

Add files via upload
timmyFrederiksen committed 1 hour ago
Verified c13bf65

Adding Remove From Menu Sequence Diagram
PatrickBoyle11 committed 1 hour ago
Verified ec0d9ae

some changes to logical placement
timmyFrederiksen committed 1 hour ago
fca2940

Delete sample.txt
timmyFrederiksen committed 2 hours ago
Verified 83b772e

Add files via upload
timmyFrederiksen committed 2 hours ago
Verified b27775d

Delete sample.txt
timmyFrederiksen committed 2 hours ago
Verified 1f04f58

Add files via upload
timmyFrederiksen committed 2 hours ago
Verified 7b970f2

Delete sample.txt
timmyFrederiksen committed 2 hours ago
Verified 8fc9825

Issues Resolved: 18

Timecards Report

Francis Boyle – 14 hours

- Attended Meetings
- Updated Iteration 1
- Test Coverage
- Three Sequence Diagrams
    - SD: Add Item to Cart, Remove Item from Cart, Search for an Item
- Assembled Documentation PDF

Patrick Boyle – 13 hours

- Attended Meetings
- Updated Iteration 1
- Class Diagram
- Three Sequence Diagrams
    - SD: Edit, Add, Remove Item to Menu

Timmy Frederiksen – 15 hours

- Attended Meetings
- User Interface DEMO
- Updated Iteration 1
- Three Sequence Diagrams
    - Pay, Redeem Offer, Manager Login
- Test Coverage
- Updated website
- GRASP

Jonathan Acosta - 12 hours

- Attended Meetings
- Updated Iteration 1
- Test Coverage
- Three Sequence Diagrams
    - SD: Call Help, Display Menu, Manager Sign-Up