Eat Smart

Eat Smart
Software Requirements Specification For
Initial Project Draft & Design

Version 1.0

Revision History

Date	Version	Description	Author
14/04/2024	1.0	Initial Draft	Melido Enrique Bello Navarro Arber Krashi Esther Mallen Mohammad Mushfiqur Rahman
17/04/2024	1.0	Finished working on the different diagrams	Melido Enrique Bello Navarro Arber Krashi Esther Mallen Mohammad Mushfiqur Rahma
19/04/2024	1.0	Did the functionalities pseudo-code and touching on final details	Melido Enrique Bello Navarro Arber Krashi Esther Mallen Mohammad Mushfiqur Rahma

Table of Contents

1. Introduction

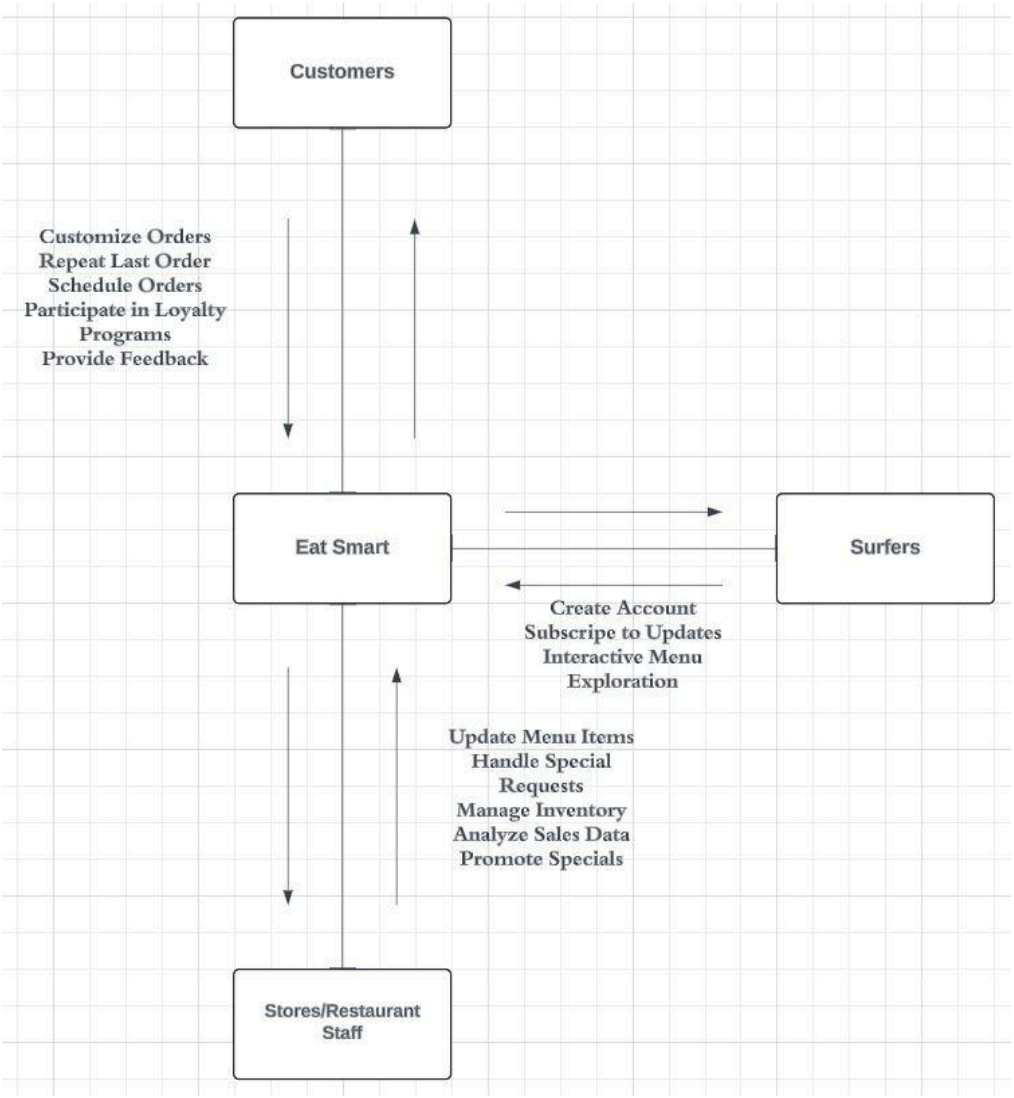
This design report will give an overview of the design for the Eat Smart system.

a. Purpose

This document is meant to detail the functionalities to be carried out by Eat Smart system as specified in the earlier specification report. This section focuses on introducing how the system is meant to function.

b. Collaboration Class Diagram

The collaboration class diagram below provides an overview of the Eat Smart system. This overview outlines the interactions of the various users with the system and the overall functionality of the system under typical use.



2. All Use Cases

Enhanced Use Cases for Customers

1. **Customize Orders**
    - Normal: A customer logs in, selects a burger, and customizes it by removing onions and adding avocado.
    - Exceptional: The customer wants to remove an ingredient not listed separately (like a spice blend in a sauce) which the restaurant cannot accommodate without altering the dish for all customers.
2. **Repeat Last Order**
    - Normal: A regular customer chooses to reorder their favorite meal from last week with just a few clicks.
    - Exceptional: The system fails to retrieve a past order due to a technical issue, or the restaurant has discontinued an item from that order.
3. **Schedule Orders**
    - Normal: A customer schedules a birthday dinner delivery for 7 PM on a specific date.
    - Exceptional: The customer tries to schedule a delivery for a time when the restaurant is unexpectedly closed or fully booked.
4. **Participate in Loyalty Programs**
    - Normal: A customer earns points after each purchase and redeems them for a free appetizer on their next order.
    - Exceptional: Points fail to accrue due to a system glitch, or the customer attempts to redeem points during a blackout period when redemptions are not allowed.
5. **Provide Feedback**
    - Normal: After dining, a customer submits a positive review and rating through the platform.
    - Exceptional: A customer tries to submit feedback but encounters a system error that prevents the review from being saved or posted.

New Use Cases for Stores

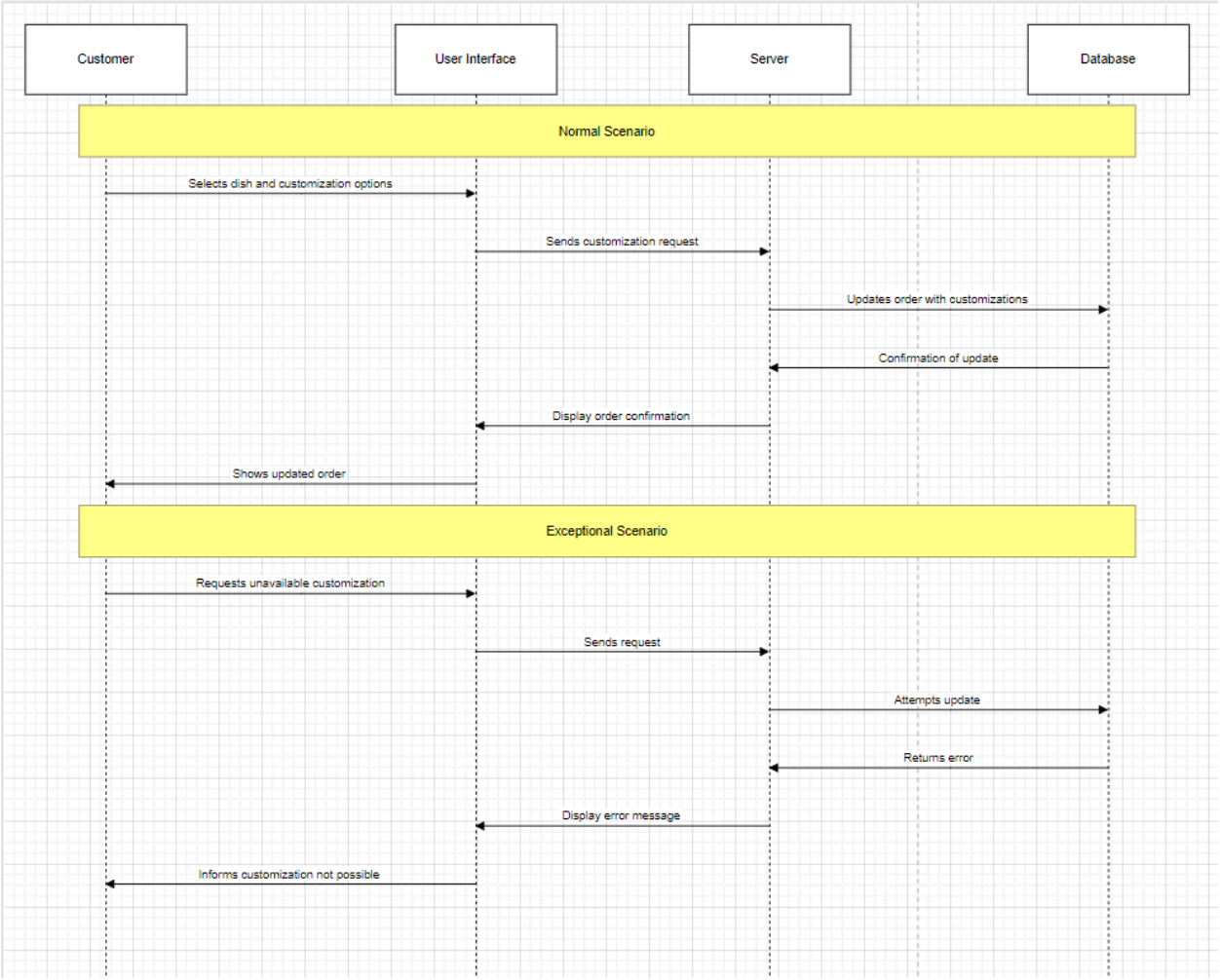
1. **Update Menu Items**
    - Normal: A chef updates the menu to include a new seasonal dish.
    - Exceptional: A menu item update is not reflected in real time due to synchronization delays, leading to customer confusion.
2. **Handle Special Requests**
    - Normal: A chef receives a request to make a dish gluten-free and successfully adjusts the recipe.
    - Exceptional: A special request is lost or not communicated to the kitchen, resulting in a customer receiving the wrong dish.
3. **Manage Inventory**
    - Normal: Inventory data helps a manager order the right amount of fresh produce for the week.
    - Exceptional: The inventory system shows incorrect data due to input errors, causing overordering or shortages.
4. **Analyze Sales Data**
    - Normal: Managers use sales analytics to identify that a particular dish is very popular on weekends and adjust staffing and stock accordingly.
    - Exceptional: Sales data is skewed by a promotional event, leading to incorrect conclusions about regular customer preferences.
5. **Promote Specials**
    - Normal: A restaurant promotes a discount on all pasta dishes through the platform, driving increased orders.
    - Exceptional: A promotion is not deactivated after the intended end date due to a system error, leading to unintended discounts.

Expanded Use Cases for Surfers

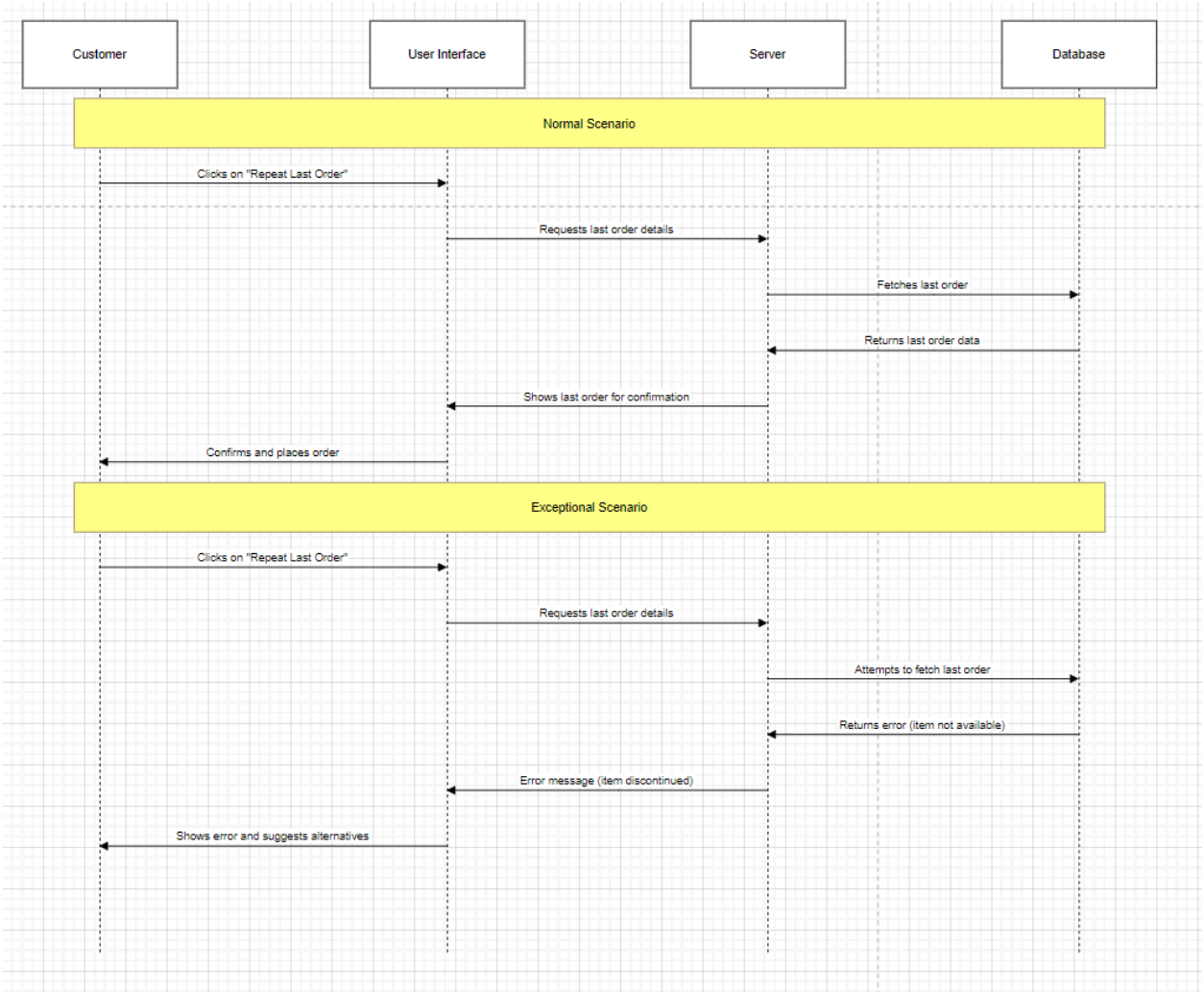
1. **Create Account**
    - Normal: A surfer easily creates a customer account using their email address.
    - Exceptional: A surfer encounters a bug during the sign-up process, such as a captcha error, which prevents account creation.
2. **Subscribe to Updates**
    - Normal: A surfer subscribes to updates from their favorite restaurant and receives emails about upcoming specials.
    - Exceptional: Email updates go to the spam folder, or the surfer receives too many emails, leading to unsubscribing.
3. **Interactive Menu Exploration**
    - Normal: A surfer filters the menu to show only vegan options and sorts by popularity.
    - Exceptional: Filtering errors occur, showing incorrect items that do not meet the dietary preferences specified.

Enhanced Use Cases for Customers - Sequence class diagrams

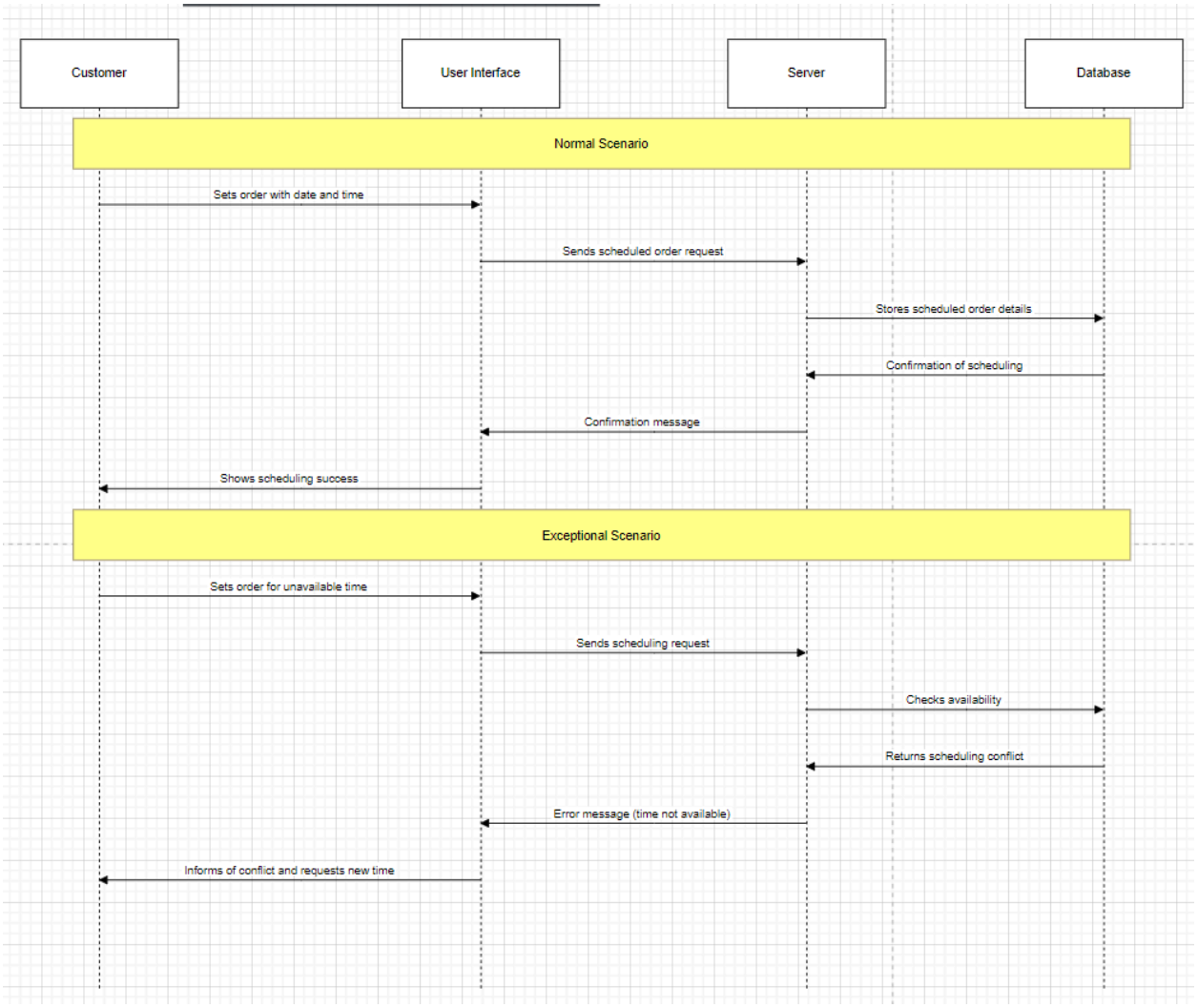
1. Customize Orders



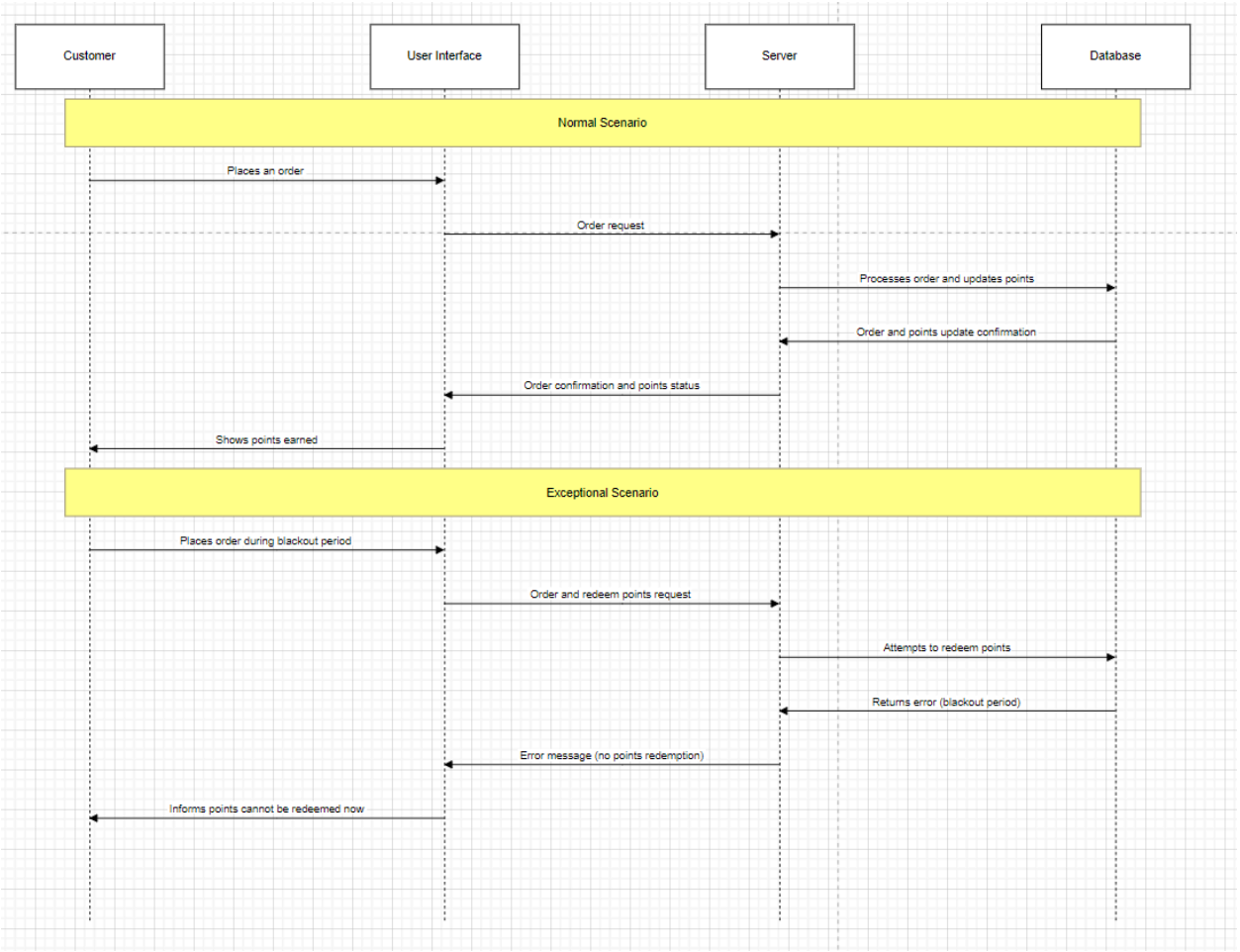
2. Repeat Last Order



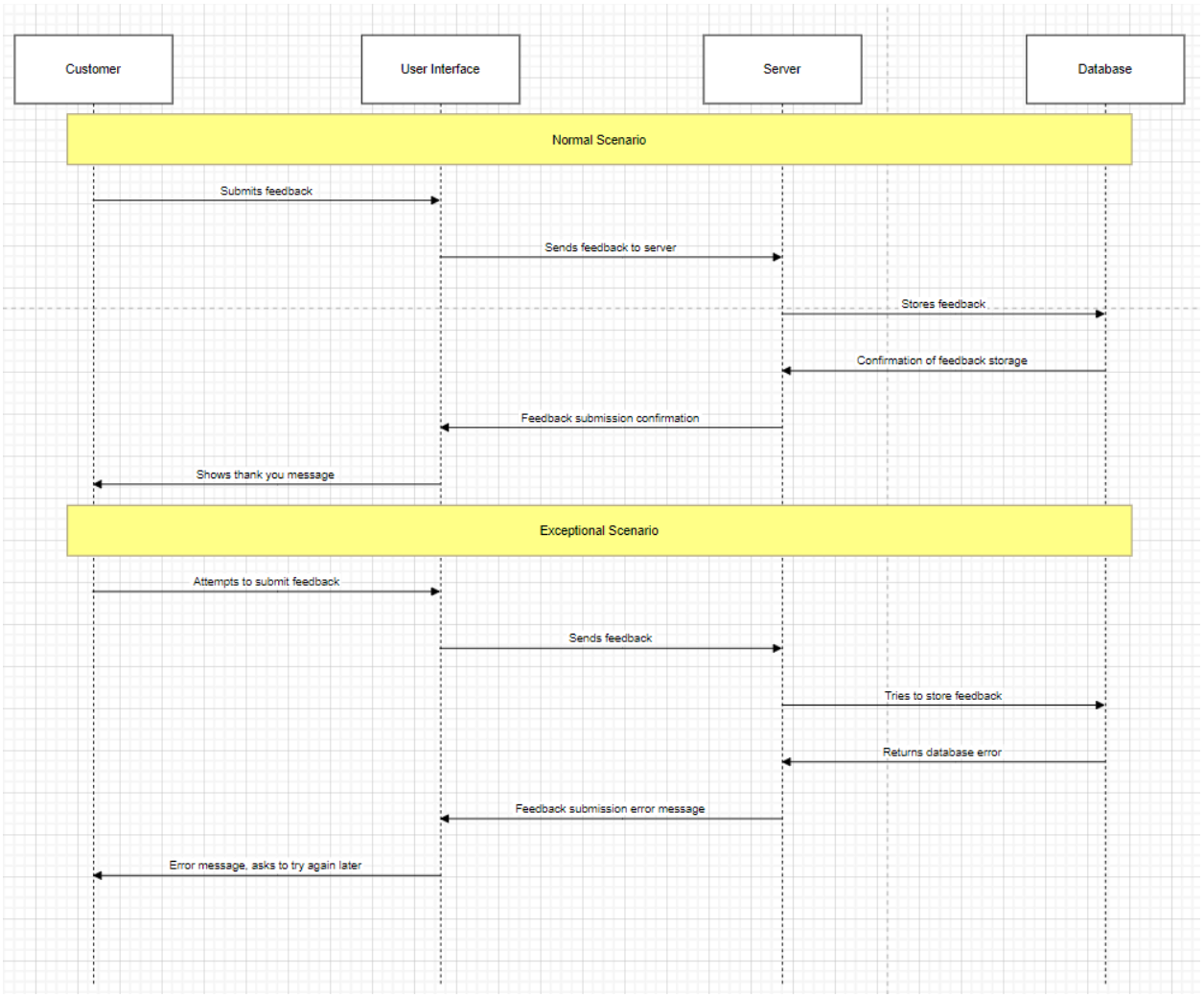
3. Schedule Orders



4. Participate in Loyalty Programs

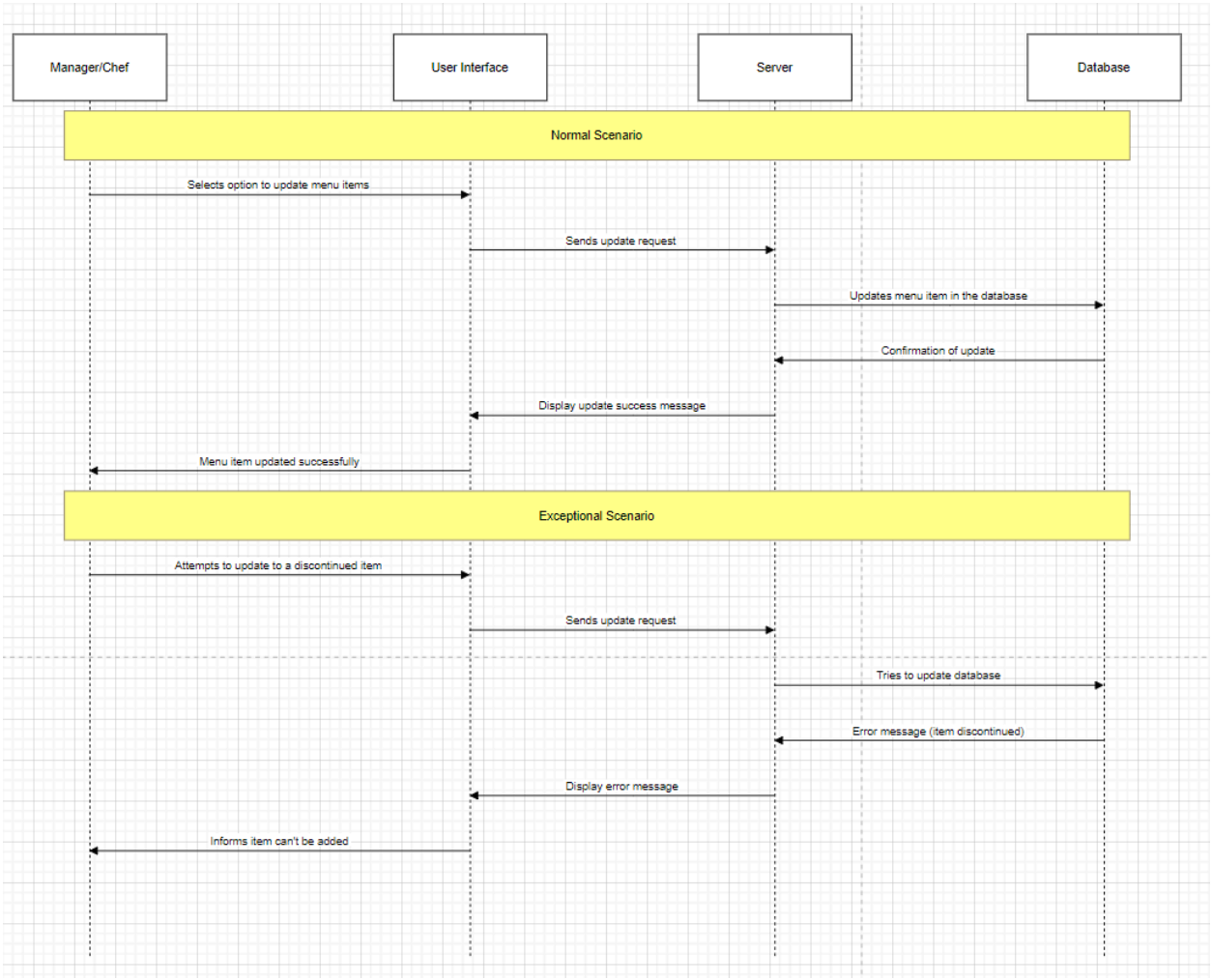


5. Provide Feedback

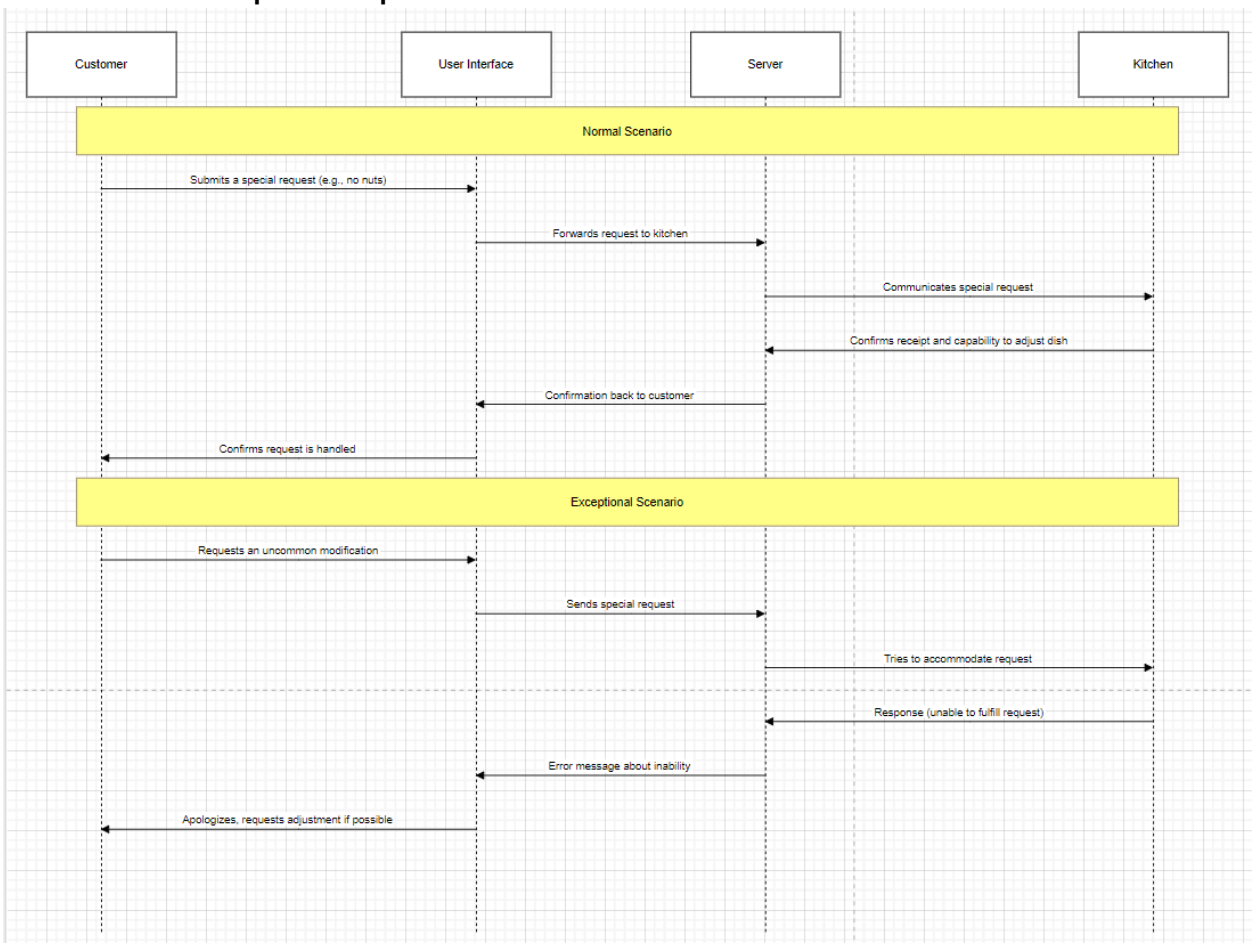


New Use Cases for Stores - Sequence class diagrams

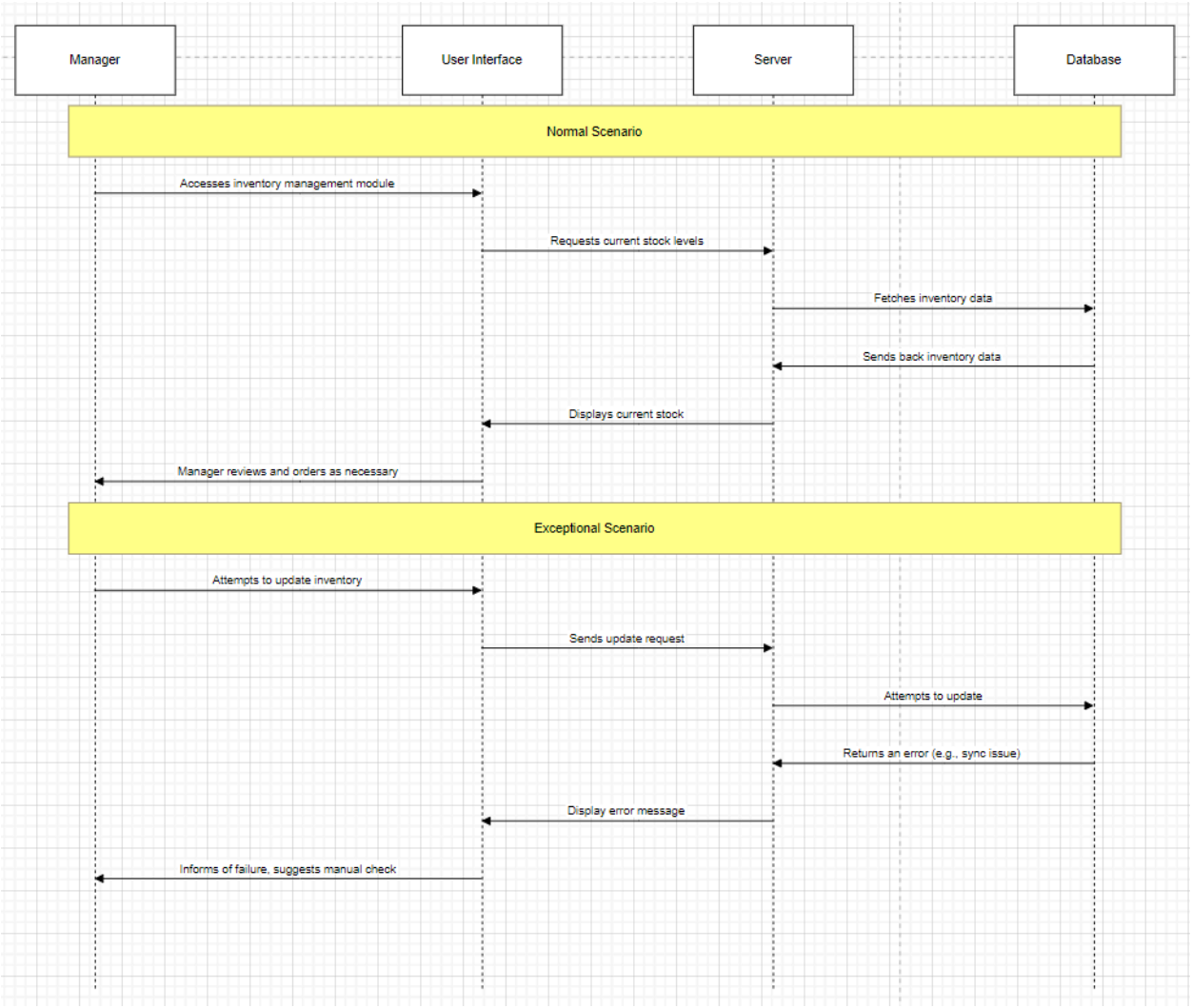
1. Update Menu Items



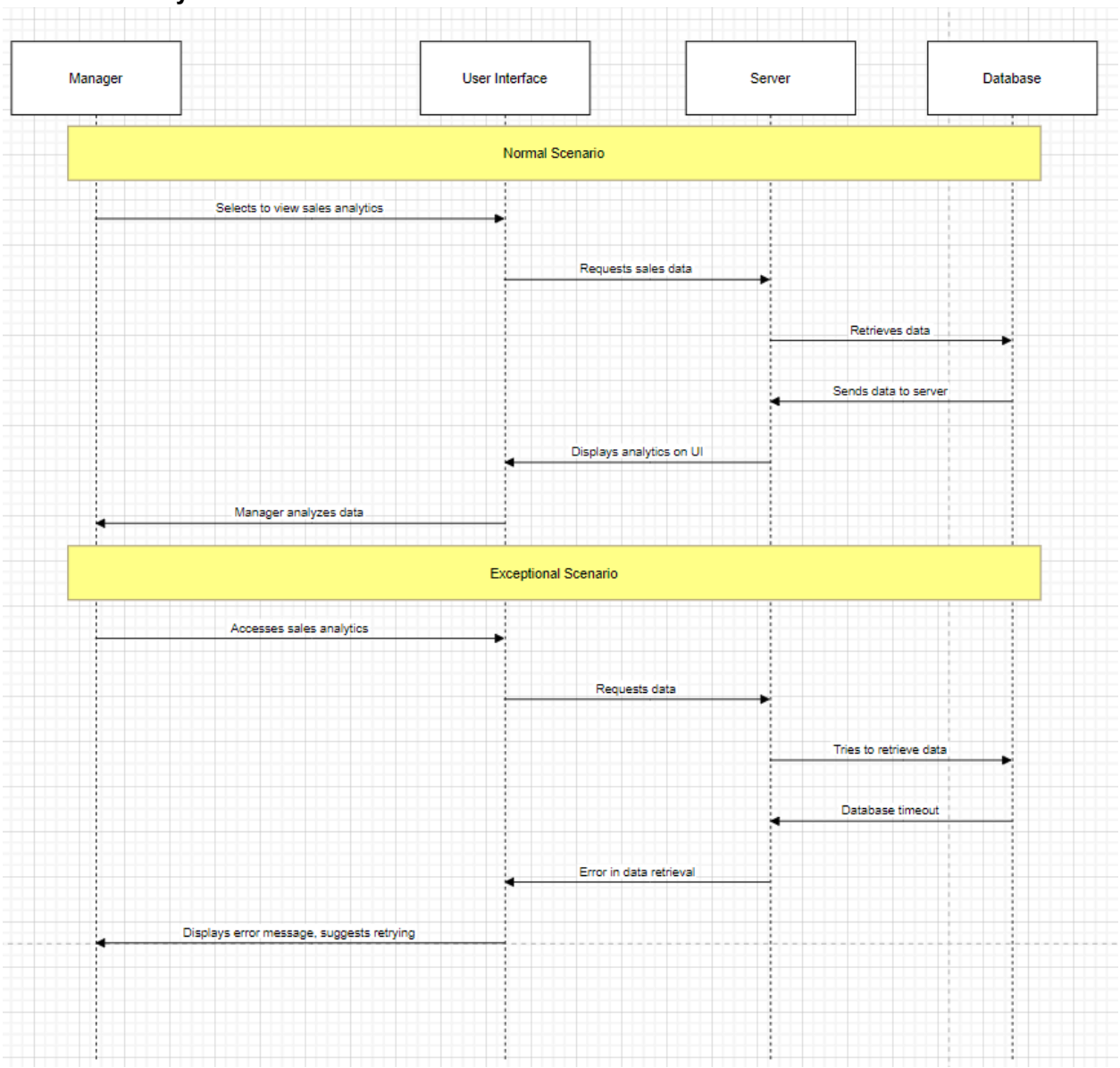
2. Handle Special Requests



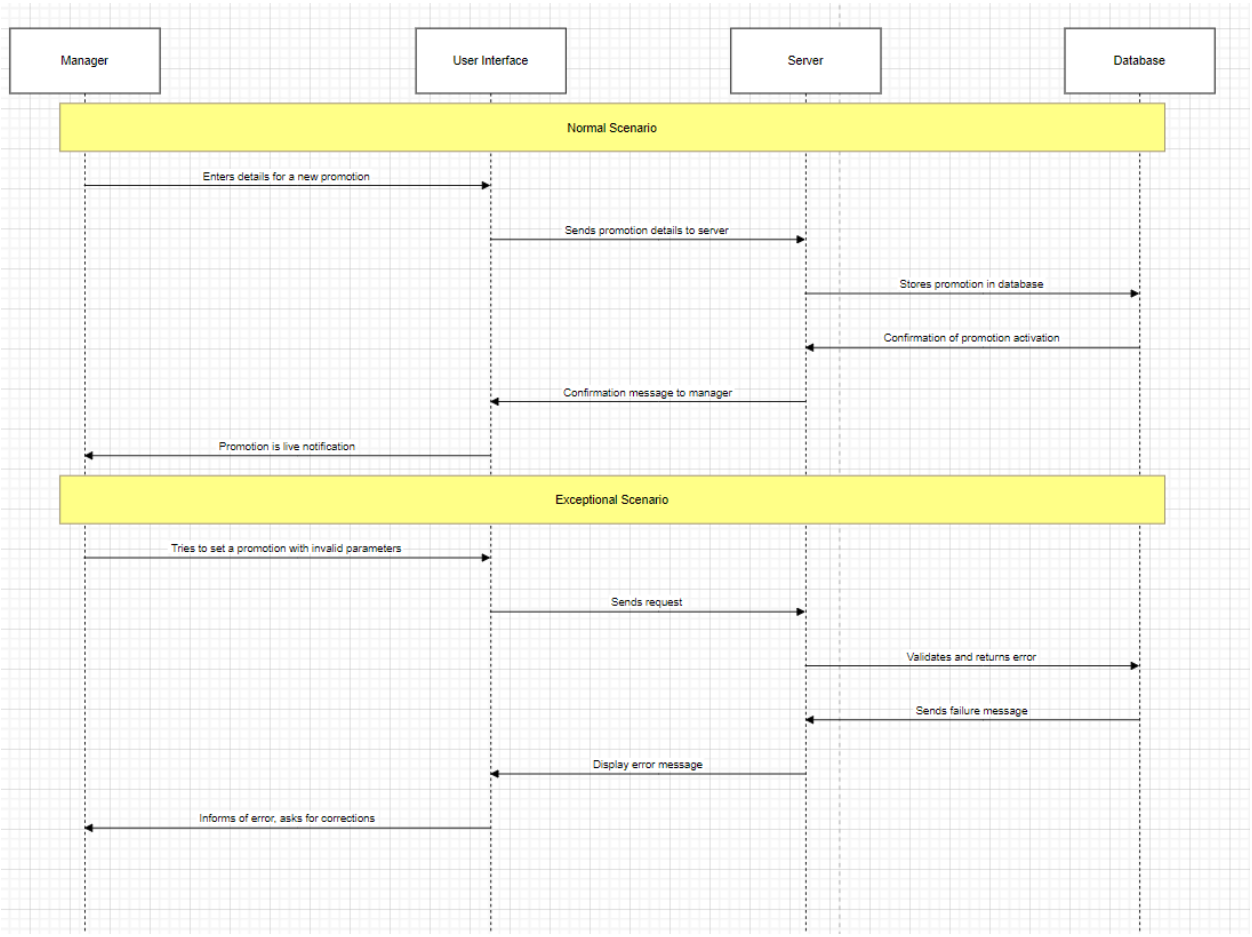
3. Manage Inventory



4. Analyze Sales Data

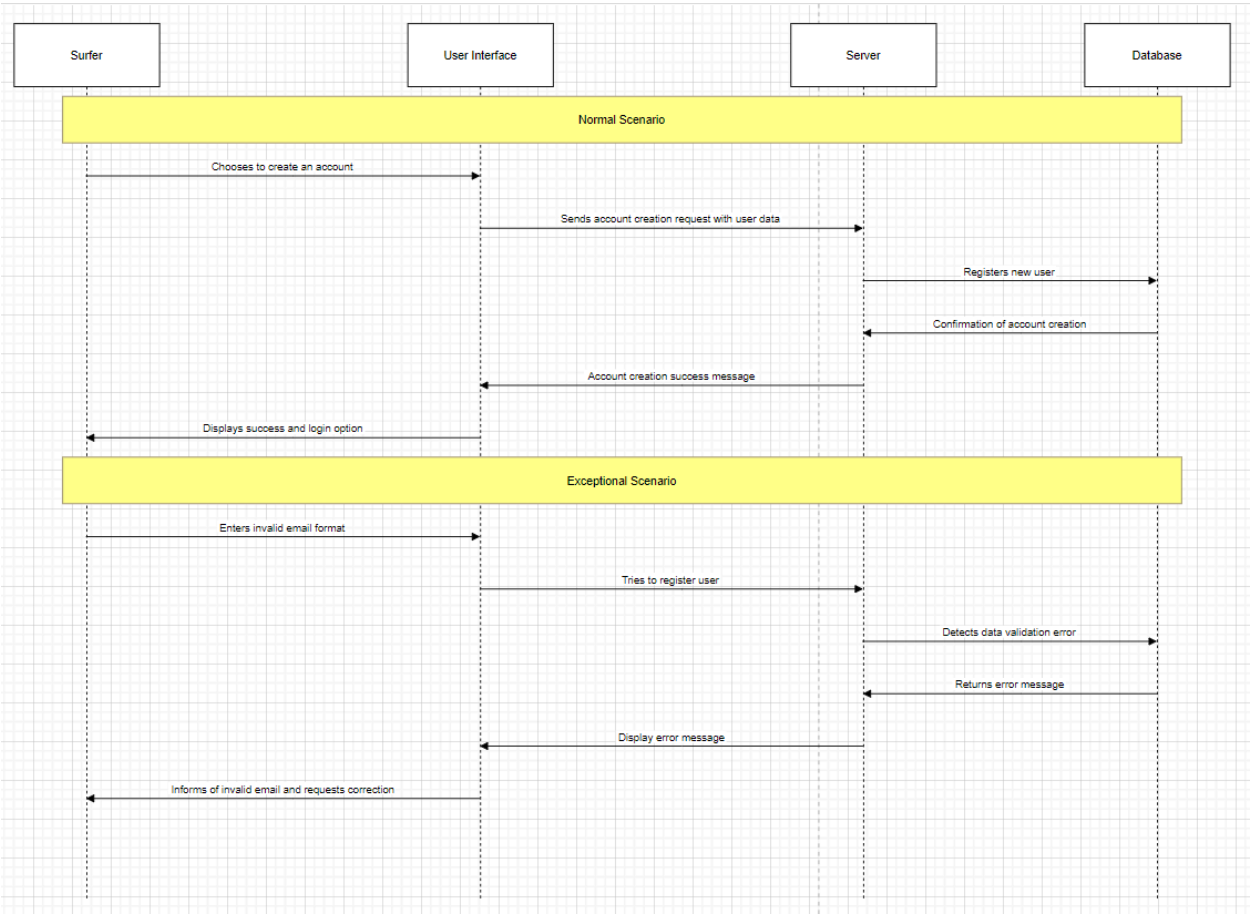


5. Promote Specials

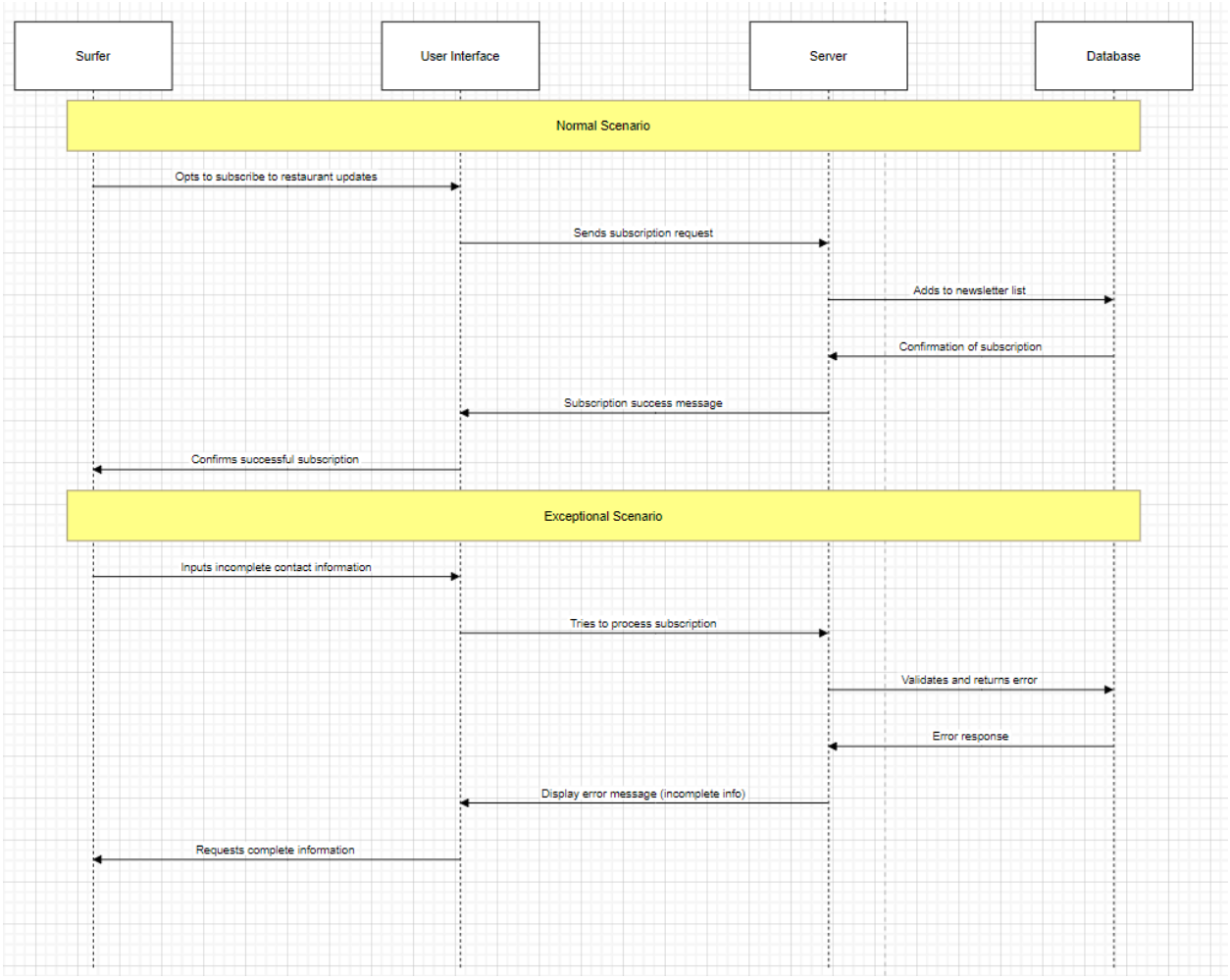


Expanded Use Cases for Surfers - Sequence class diagrams

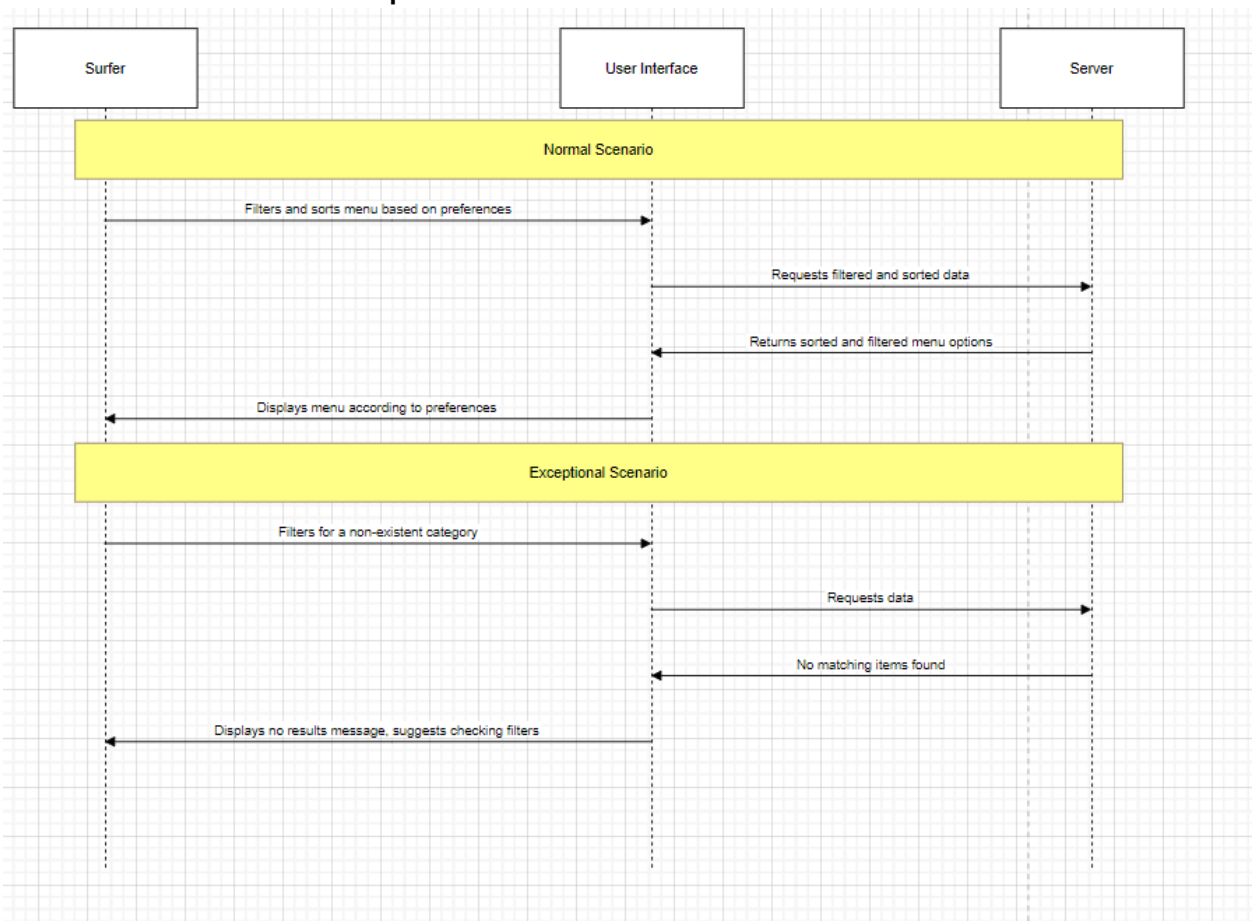
1. Create Account



2. Subscribe to Updates

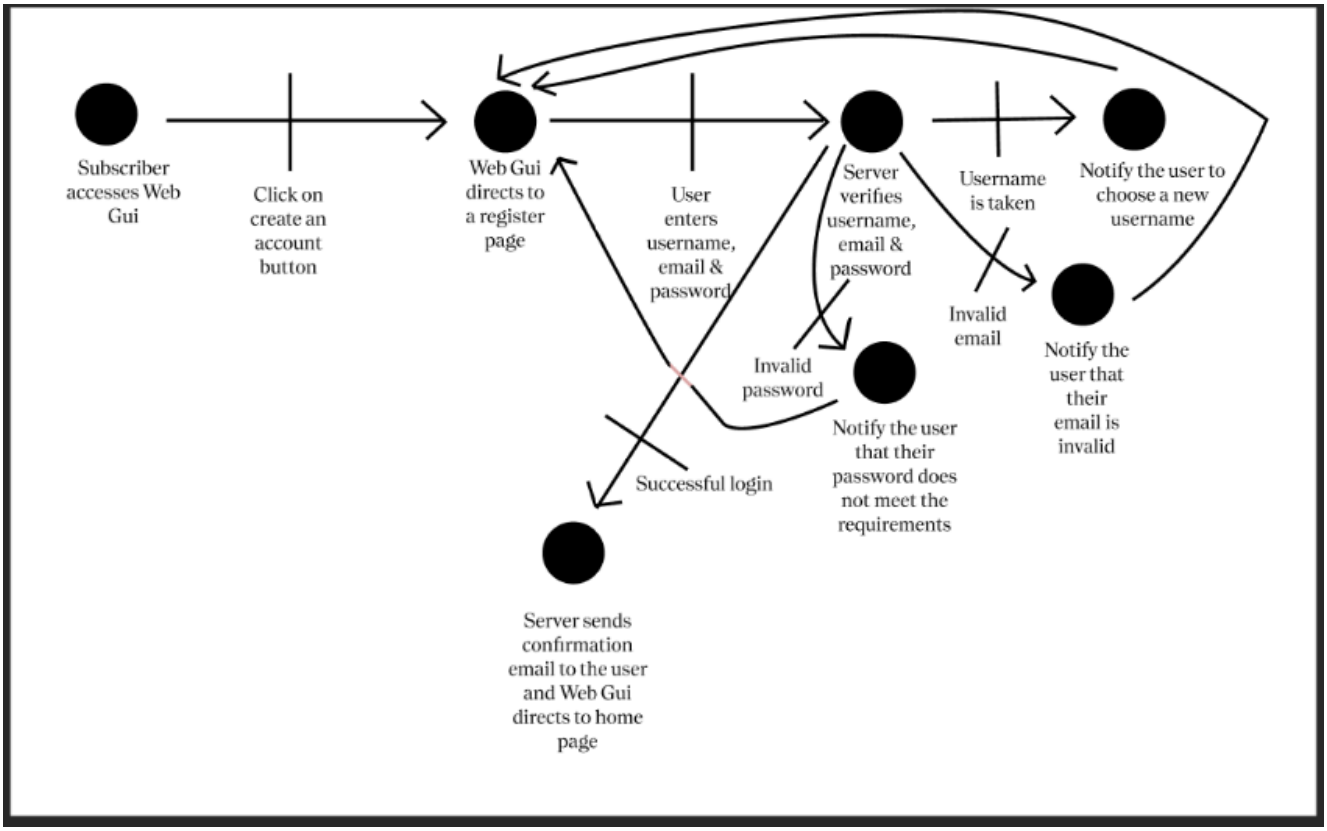


3. Interactive Menu Exploration

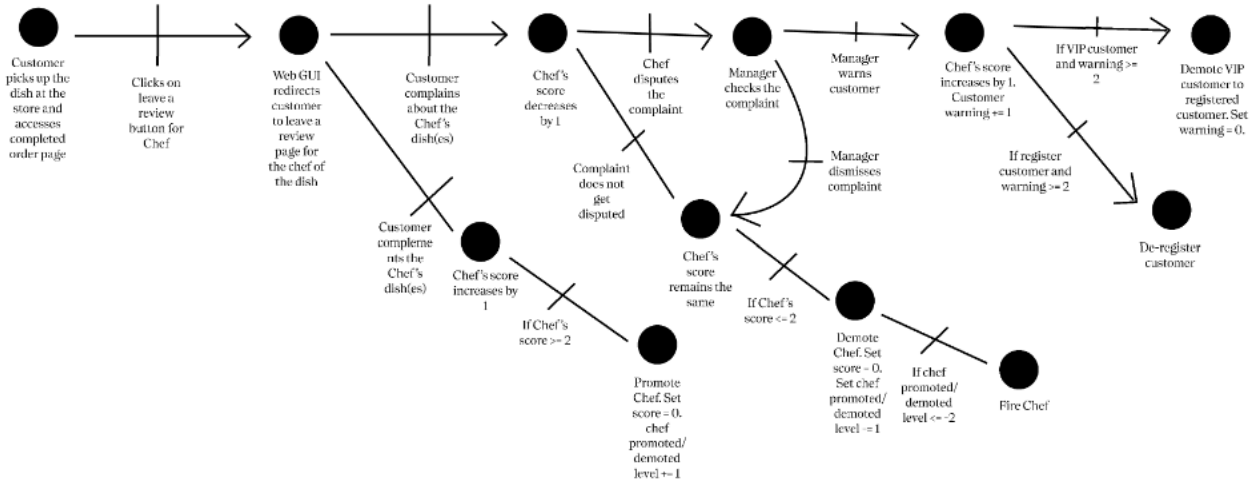


Petri-net Diagrams

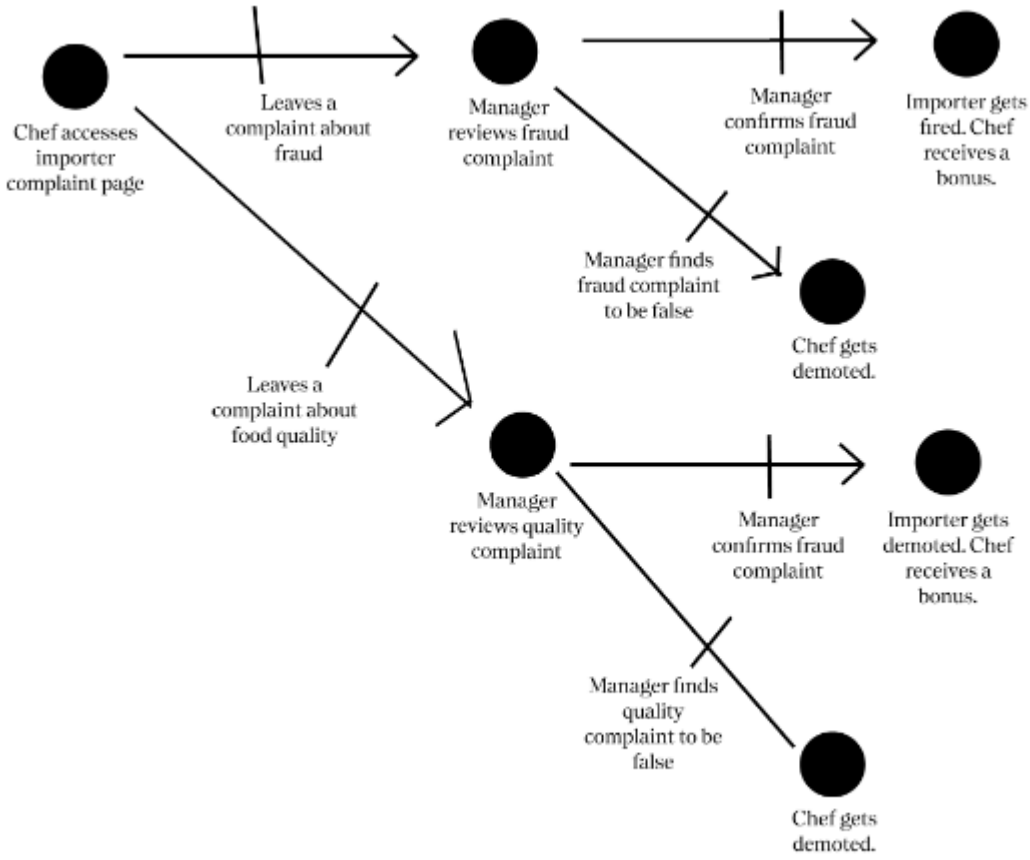
1. Subscriber registers account



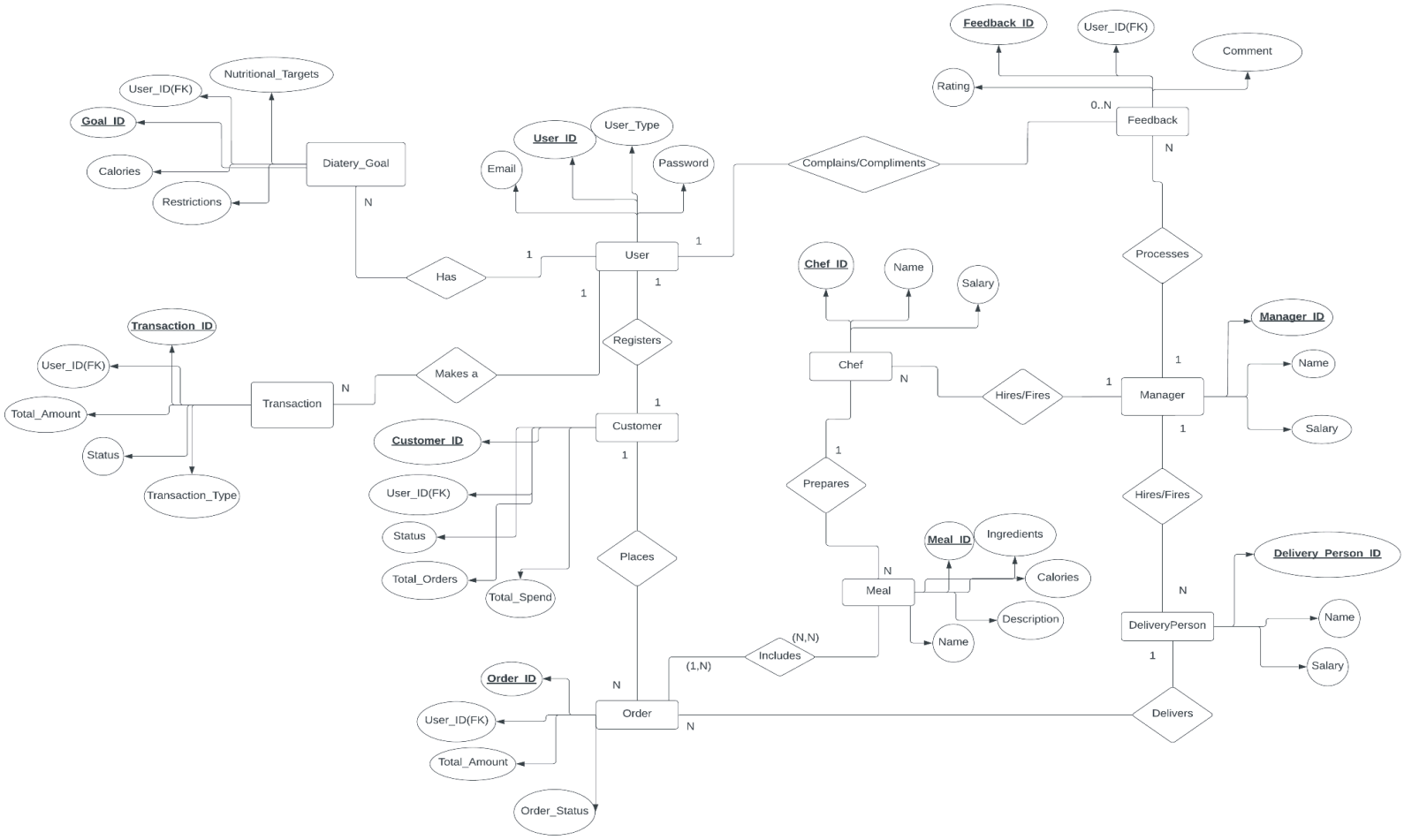
2. Customer leaves a review for Chef after picking up order from store



3. Chef leaves a complaint for the importer



3. Entity-Relation Diagram



4. Detailed Design

Enhanced Use Cases for Customers

1. Customize Orders

```
Function CustomizeOrder(dishId, customizations)
    Input: dishId (identifier for the selected dish), customizations (list of changes like add or remove
ingredients)
    Output: Success with updated dish details or Error with a message

    Begin
        if dish exists(dishId) then
            updatedDish = applyCustomizations(dishId, customizations)
            if updatedDish is valid then
                save updatedDish to database
                return Success, updatedDish details
            else
                return Error, "Customization cannot be applied"
            end if
        else
            return Error, "Dish not found"
        end if
    End

Exception Scenario:
    Function applyCustomizations(dishId, customizations)
        - Check if customizations are allowable for the dish
        - If a requested ingredient removal is part of a premade mix, return invalid
        - Otherwise, apply changes and return updated dish
```

2. Repeat Last Order

```
Function RepeatLastOrder(customerId)
    Input: customerId (unique identifier for the customer)
    Output: Success with order details or Error with a message

    Begin
        lastOrder = getLastOrder(customerId)
        if lastOrder exists then
            if items in lastOrder are available then
                placeOrder(lastOrder)
                return Success, lastOrder details
            else
                return Error, "One or more items in your last order are no longer available"
            end if
        else
            return Error, "No last order found"
        end if
    End

Exception Scenario:
    - Handle cases where the item is discontinued or out of stock
    - Suggest alternative items if possible
```

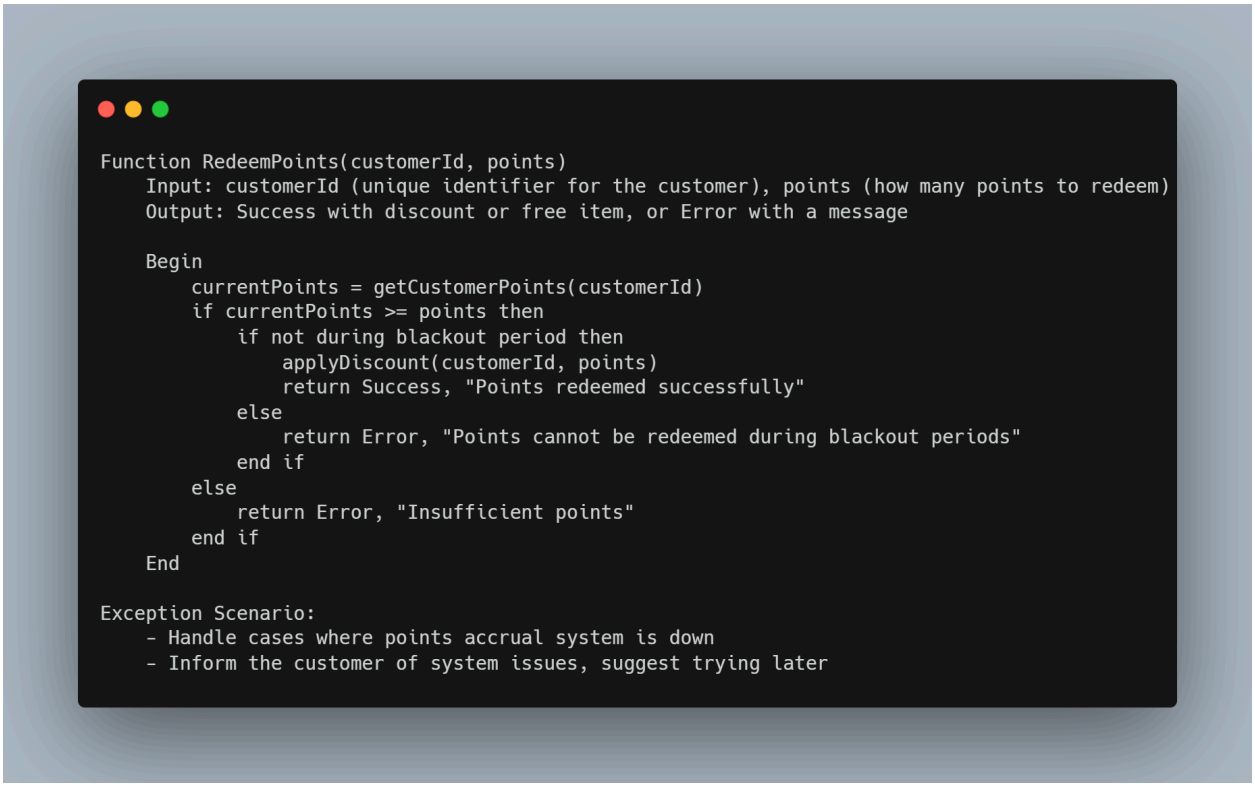
3. Schedule Orders

```
Function ScheduleOrder(orderDetails, dateTime)
    Input: orderDetails (details of the order), dateTime (scheduled time and date for the order)
    Output: Success with confirmation or Error with a message

    Begin
        if dateTime is during operating hours then
            if date is available for scheduling then
                schedule order in database for dateTime
                return Success, "Order scheduled for " + dateTime
            else
                return Error, "Selected date/time is not available"
            end if
        else
            return Error, "Restaurant is closed at the selected time"
        end if
    End

Exception Scenario:
    - Handle cases where scheduling conflicts with closed hours or full capacity
    - Provide alternative times or immediate feedback on next available slot
```

4. Participate in Loyalty Programs

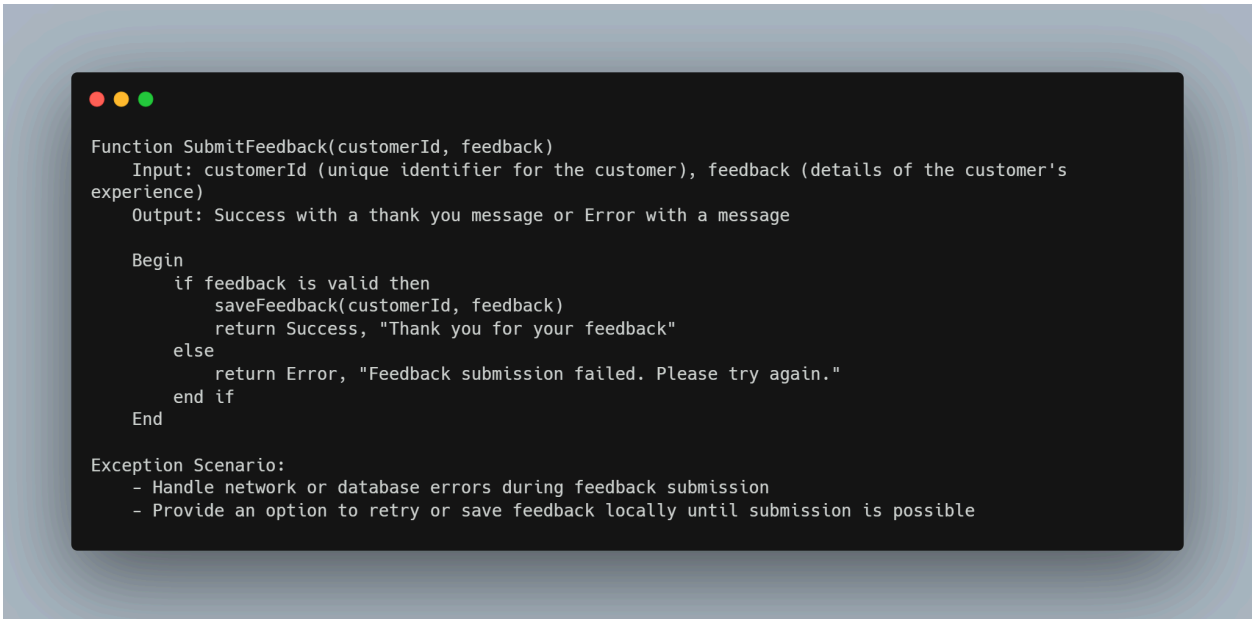


```
Function RedeemPoints(customerId, points)
    Input: customerId (unique identifier for the customer), points (how many points to redeem)
    Output: Success with discount or free item, or Error with a message

    Begin
        currentPoints = getCustomerPoints(customerId)
        if currentPoints >= points then
            if not during blackout period then
                applyDiscount(customerId, points)
                return Success, "Points redeemed successfully"
            else
                return Error, "Points cannot be redeemed during blackout periods"
            end if
        else
            return Error, "Insufficient points"
        end if
    End

Exception Scenario:
    - Handle cases where points accrual system is down
    - Inform the customer of system issues, suggest trying later
```

5. Provide Feedback



```
Function SubmitFeedback(customerId, feedback)
    Input: customerId (unique identifier for the customer), feedback (details of the customer's
experience)
    Output: Success with a thank you message or Error with a message

    Begin
        if feedback is valid then
            saveFeedback(customerId, feedback)
            return Success, "Thank you for your feedback"
        else
            return Error, "Feedback submission failed. Please try again."
        end if
    End

Exception Scenario:
    - Handle network or database errors during feedback submission
    - Provide an option to retry or save feedback locally until submission is possible
```

New Use Cases for Stores

1. Update Menu Items

```
Function UpdateMenuItem(menuItemId, newDetails)
    Input: menuItemId (identifier for the menu item), newDetails (object containing updated information
such as price, description, availability)
    Output: Success message or Error message

    Begin
        if menuItem exists(menuItemId) then
            update menuItem in database with newDetails
            return Success, "Menu item updated successfully"
        else
            return Error, "Menu item not found"
        end if
    End

Exception Scenario:
    Function UpdateMenuItemExceptionHandling(menuItemId)
        - Check database connection before attempting update
        - If database is temporarily down, log the update attempt and retry later
        - Return Error, "Temporary database issue. Please try again later."
```

2. Handle Special Requests

```
Function HandleSpecialRequest(orderId, specialRequestDetails)
    Input: orderId (identifier for the order), specialRequestDetails (details about the customer's
special dietary needs or preferences)
    Output: Confirmation of receipt or Error message

    Begin
        if order exists(orderId) and is modifiable then
            update order with specialRequestDetails
            notify kitchen to prepare order as per the new specifications
            return Success, "Special request handled successfully"
        else
            return Error, "Order not found or cannot be modified"
        end if
    End

Exception Scenario:
    - If the request includes an allergen-free preparation in a kitchen not equipped for such, return
Error, "Unable to fulfill allergen-free conditions"
```

3. Manage Inventory

```
Function UpdateInventory(item, quantity)
    Input: item (identifier for the inventory item), quantity (number to add or subtract)
    Output: Updated inventory details or Error message

    Begin
        if inventory item exists(item) then
            adjust inventory levels of item by quantity
            return Success, "Inventory updated successfully"
        else
            return Error, "Item not found in inventory"
        end if
    End

Exception Scenario:
    - If the database cannot be accessed due to a network issue, return Error, "Cannot update inventory
at this time. Network issue."
```

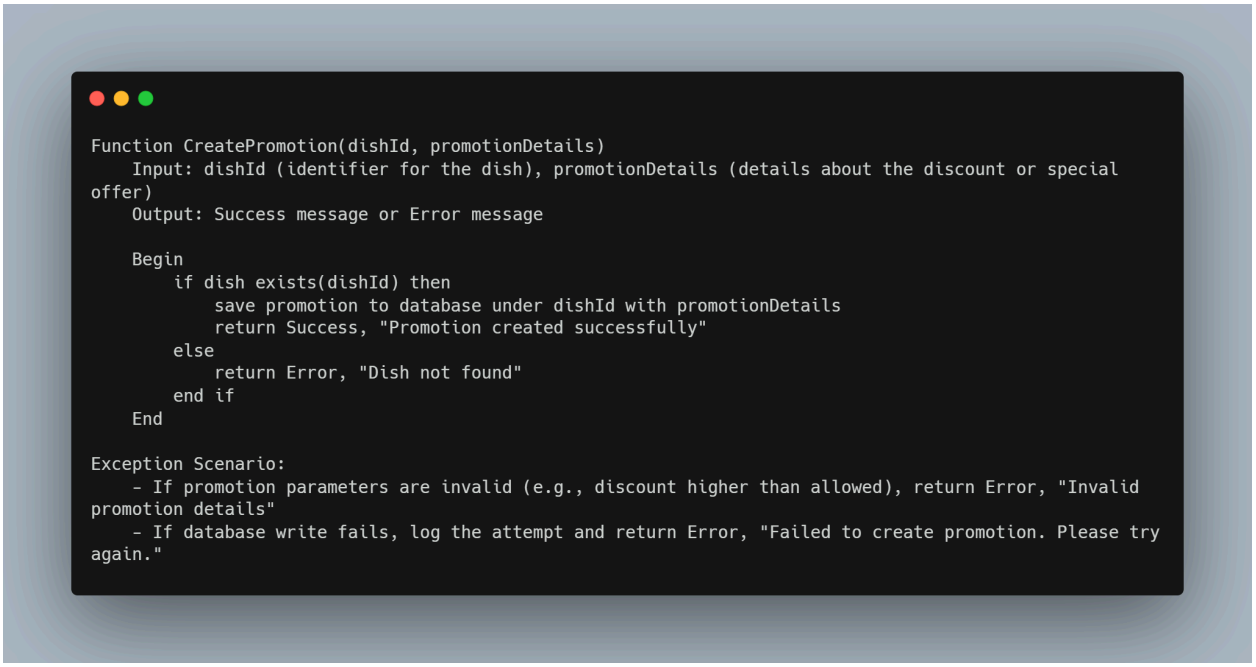
4. Analyze Sales Data

```
Function GetSalesData(dateRange)
    Input: dateRange (start date to end date for which sales data is needed)
    Output: Sales data or Error message

    Begin
        salesData = fetchSalesData(dateRange)
        if salesData is not empty then
            return Success, salesData
        else
            return Error, "No sales data available for the specified range"
        end if
    End

Exception Scenario:
    - If data retrieval is interrupted or corrupted, log the error and retry if possible
    - Return Error, "Failed to retrieve sales data due to a system error."
```

5. Promote Specials



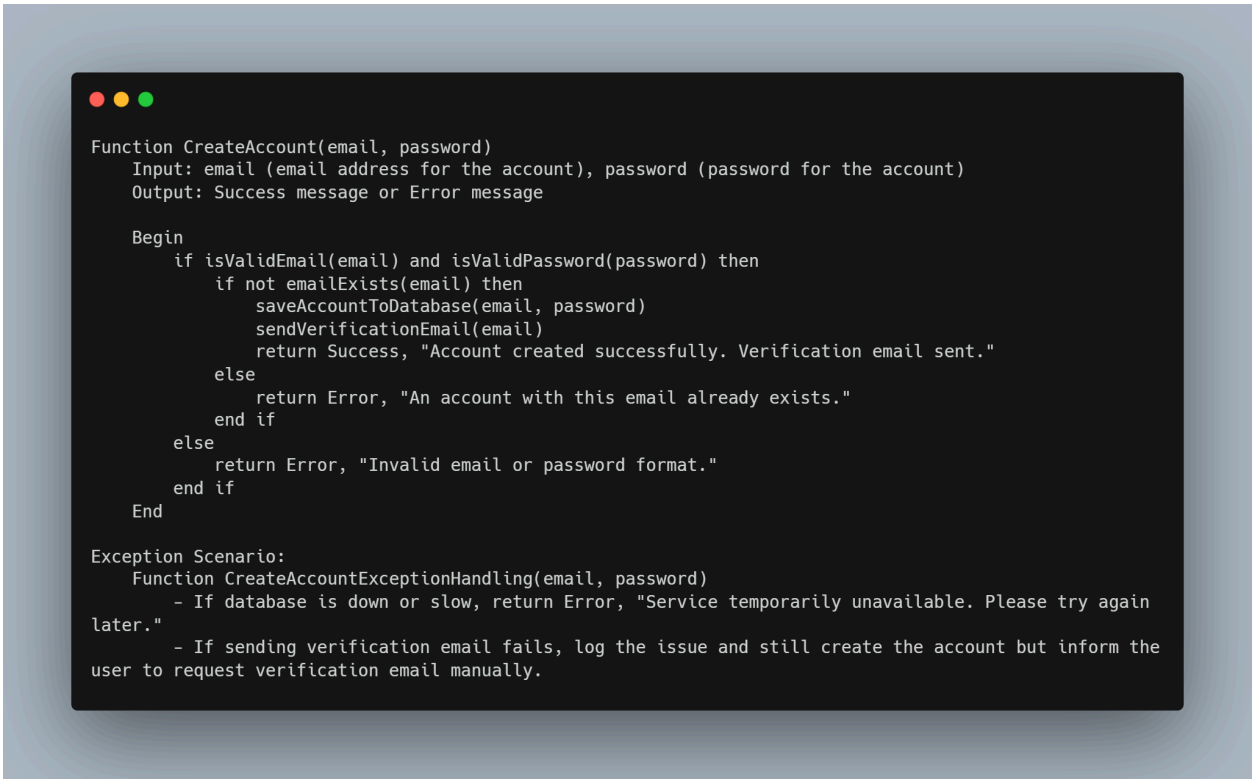
```
Function CreatePromotion(dishId, promotionDetails)
    Input: dishId (identifier for the dish), promotionDetails (details about the discount or special
offer)
    Output: Success message or Error message

    Begin
        if dish exists(dishId) then
            save promotion to database under dishId with promotionDetails
            return Success, "Promotion created successfully"
        else
            return Error, "Dish not found"
        end if
    End

Exception Scenario:
    - If promotion parameters are invalid (e.g., discount higher than allowed), return Error, "Invalid
promotion details"
    - If database write fails, log the attempt and return Error, "Failed to create promotion. Please try
again."
```

Expanded Use Cases for Surfers

1. Create Account



```
Function CreateAccount(email, password)
    Input: email (email address for the account), password (password for the account)
    Output: Success message or Error message

    Begin
        if isValidEmail(email) and isValidPassword(password) then
            if not emailExists(email) then
                saveAccountToDatabase(email, password)
                sendVerificationEmail(email)
                return Success, "Account created successfully. Verification email sent."
            else
                return Error, "An account with this email already exists."
            end if
        else
            return Error, "Invalid email or password format."
        end if
    End

Exception Scenario:
    Function CreateAccountExceptionHandling(email, password)
        - If database is down or slow, return Error, "Service temporarily unavailable. Please try again
later."
        - If sending verification email fails, log the issue and still create the account but inform the
user to request verification email manually.
```

2. Subscribe to Updates

```
Function SubscribeToUpdates(email, restaurantId)
    Input: email (email address for subscription), restaurantId (identifier for the restaurant the user
wants updates from)
    Output: Success message or Error message

    Begin
        if isValidEmail(email) then
            if not isSubscribed(email, restaurantId) then
                addSubscriptionToDatabase(email, restaurantId)
                return Success, "Subscribed successfully to updates."
            else
                return Error, "Already subscribed to updates for this restaurant."
            end if
        else
            return Error, "Invalid email format."
        end if
    End

Exception Scenario:
    Function SubscribeToUpdatesExceptionHandling(email, restaurantId)
        - If database connection fails, return Error, "Unable to process subscription at this time."
        - If email service to confirm subscription fails, log the error and notify the user to confirm
subscription manually.
```

3. Interactive Menu Exploration

```
Function ExploreMenu(filterOptions)
    Input: filterOptions (object containing filtering criteria such as dietary restrictions, popularity,
etc.)
    Output: Filtered menu items or Error message

    Begin
        menuItems = getMenuItems()
        filteredItems = applyFilters(menuItems, filterOptions)
        if not isEmpty(filteredItems) then
            return Success, filteredItems
        else
            return Error, "No items match your criteria."
        end if
    End

Exception Scenario:
    Function ExploreMenuExceptionHandling(filterOptions)
        - If the menu data retrieval fails, return Error, "Failed to retrieve menu. Please try again."
        - If filtering logic errors (e.g., invalid filters), log the error and prompt the user to adjust
the filters.
```

5. System screens:

Here, we demonstrate major GUI screens of the system and showcase a prototype of the functionality of Eat Smart.

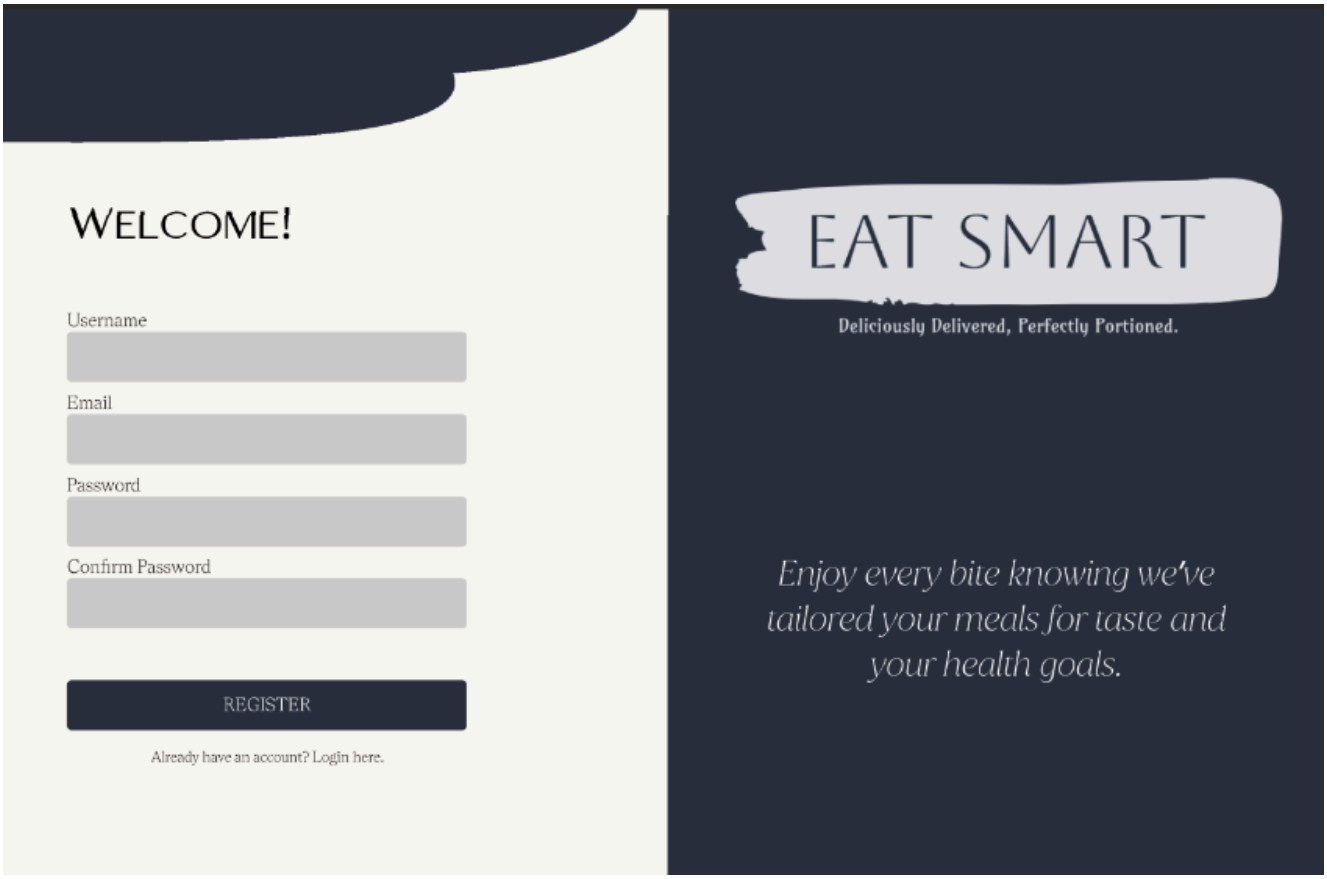


Figure 1: RegisterPage

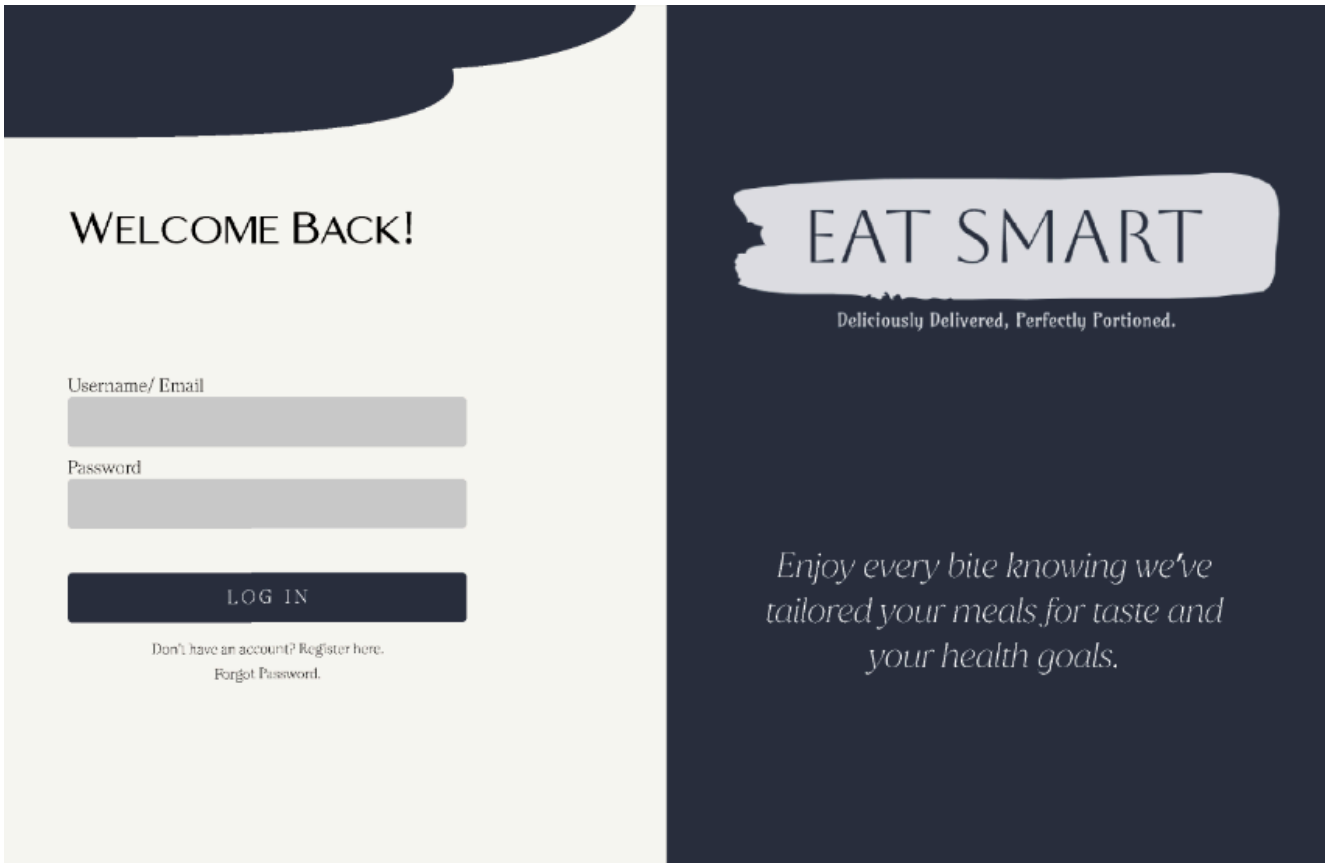


Figure 2: Login Page

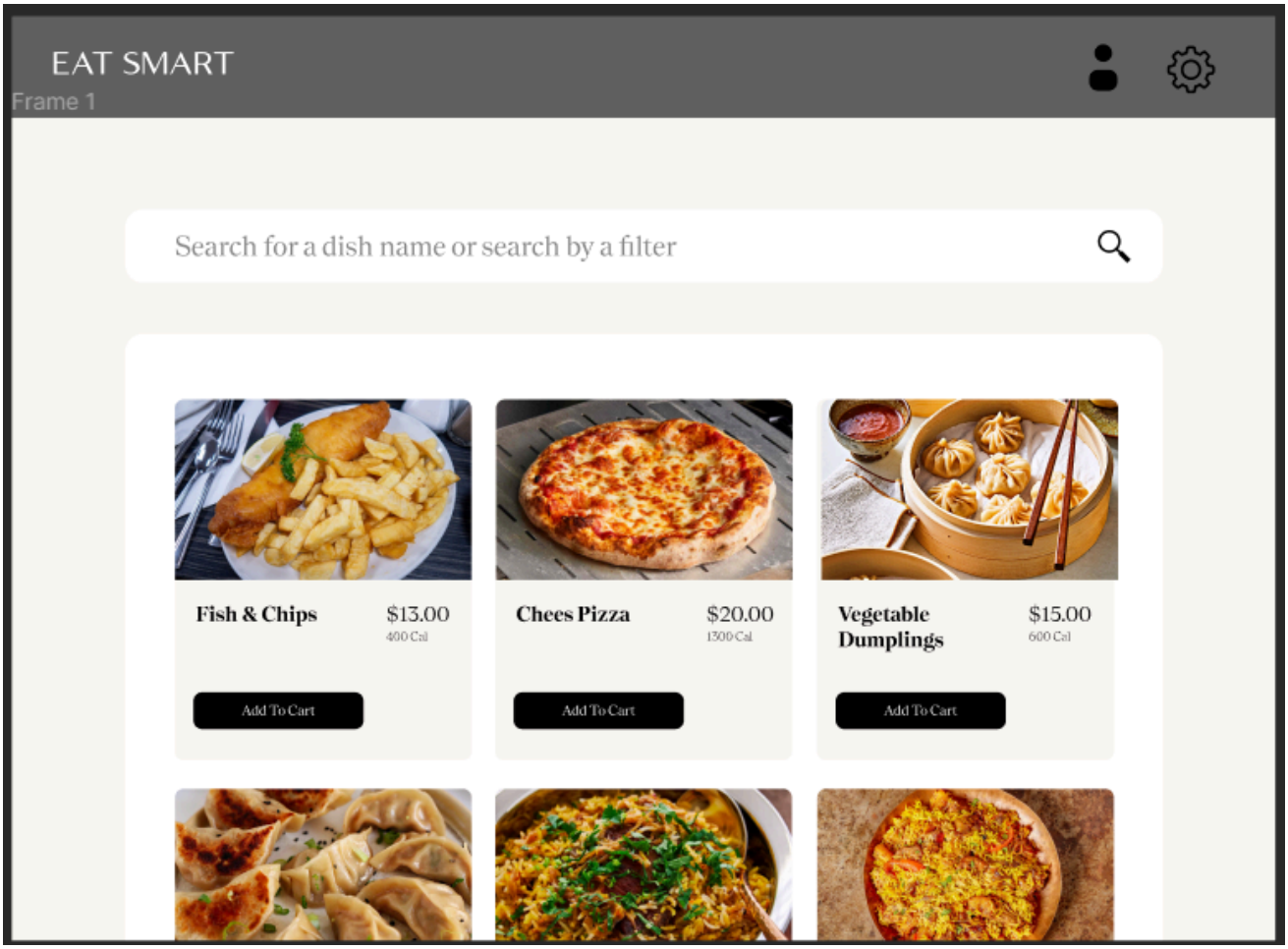


Figure 3: Page 1 of Website Prototype (Dish selecting page for customer)

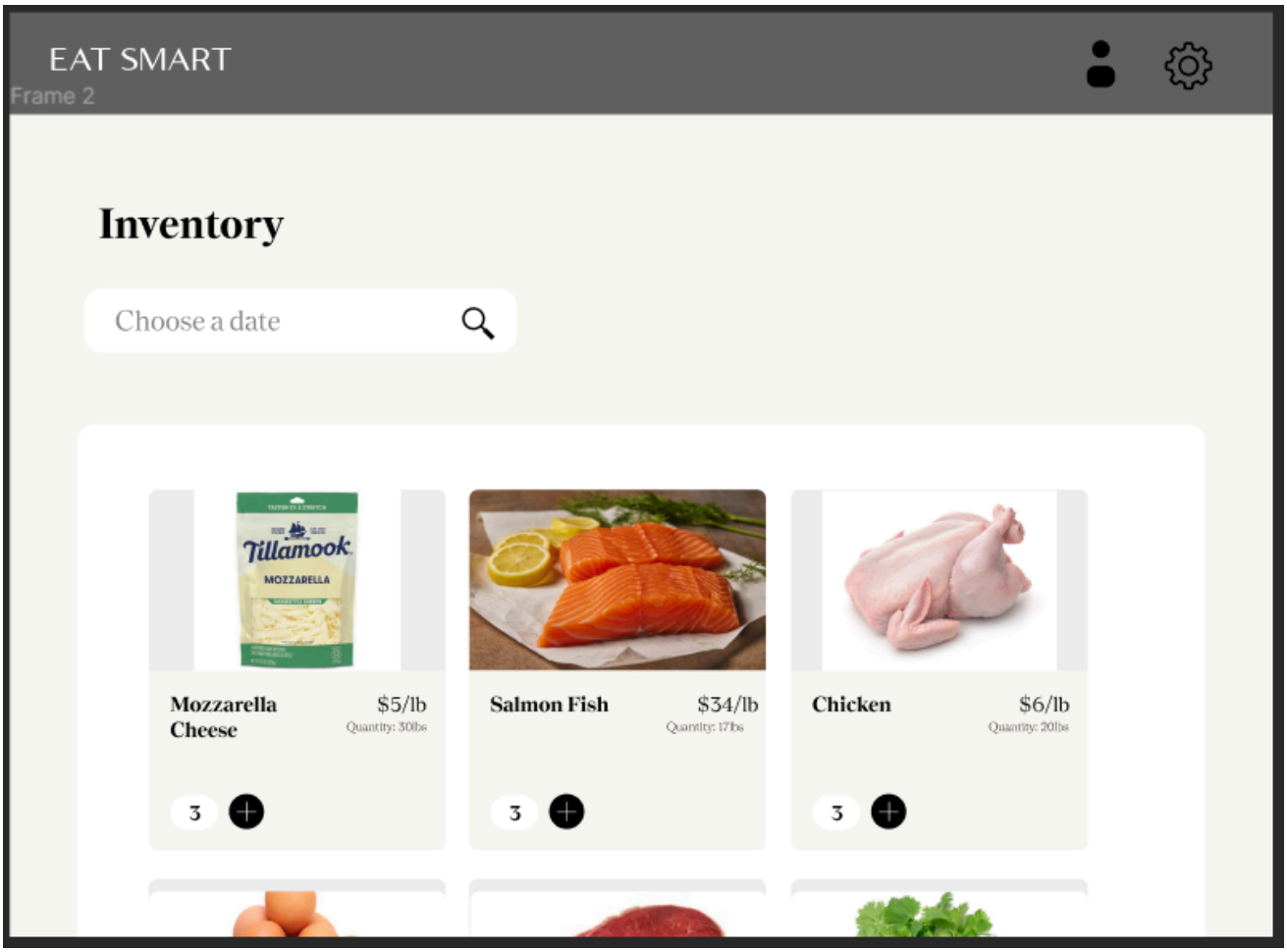


Figure 4: Page 2 of Website Prototype (Stock ordering page for store users)

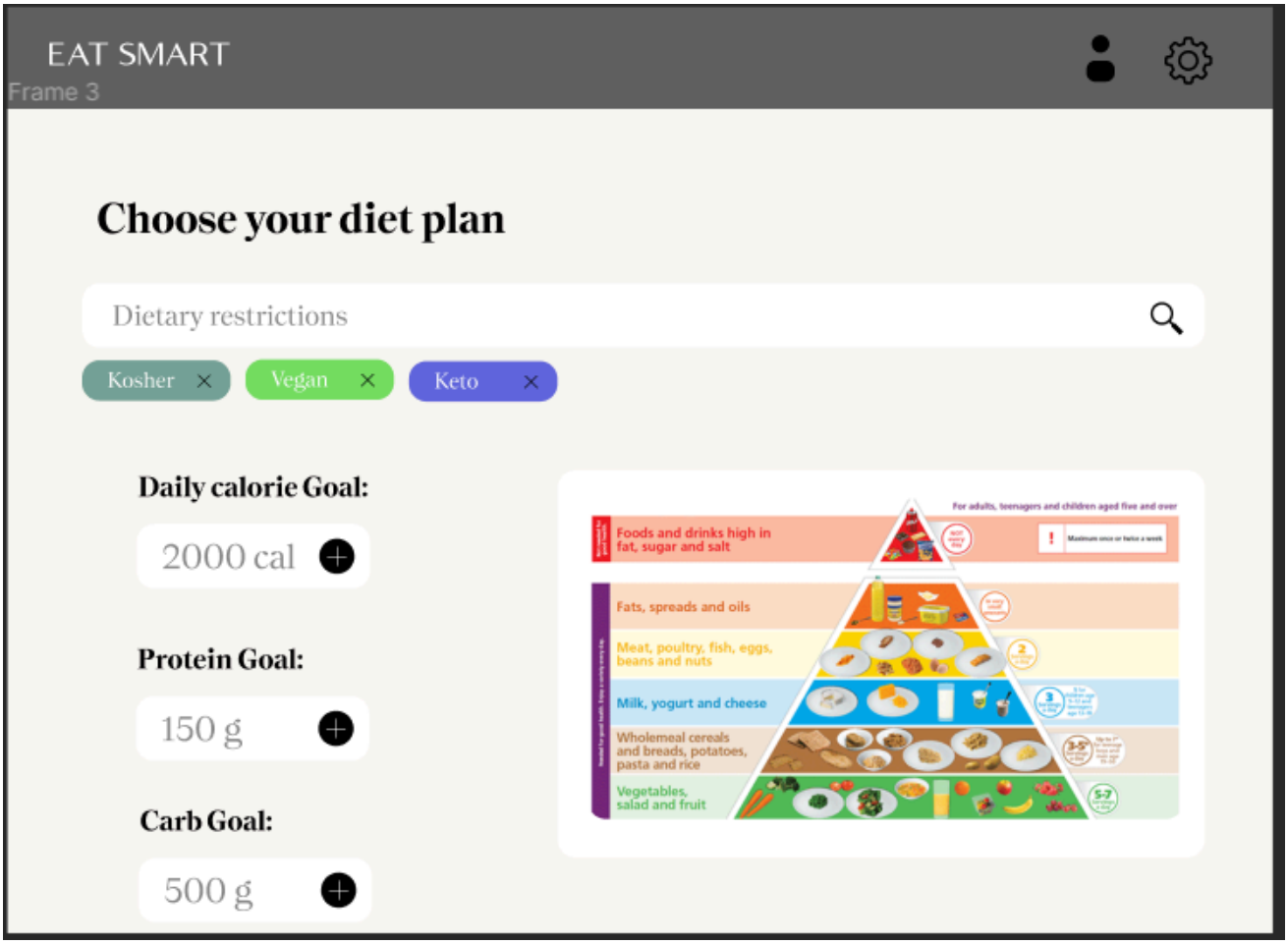


Figure 5: Page 3 of Website Prototype (Diet planning page for customer user)

6. Some memos of the group meetings

4/11/24

- Discuss who's doing what, splitting up the work for each diagram
- Reviewing how to do diagrams such as ER, use cases and collaboration class

4/12/24

- Define use cases, and how we target and develop each of these use cases
- Assign tasks to get started on working on diagrams

4/14/24

- Started writing pseudocode and thinking about the functionalities to use
- Use-cases started to become developed
    - Functionalities as well
- Collaboration class diagram was developed first to get an overview of everything else we had to do
    - Helped visualize and demonstrate the flow of data

4/15/14

- Reviewed petri-nets and how they work in order to incorporate them
    - Drew petri-nets for Eat Smart
- Reviewed and started developing the ER diagram

4/17/14

- Tried to develop a simple prototype for the systems screens
    - Users Login page
    - Dish selecting page for customer
    - Stock ordering page
    - Diet planning page

4/19/14

- Add finishing touches as a group to each step

7. GitHub Repo

Url: https://github.com/mrahman4782/Eat-Smart.git

- Reports are in the report branch