Kitchen Organizer

By:

Austin Bredeken

Iman Hassan

Quin Perkins

Esikotan Orekan

Introduction

- Esikotan is presenting slides 3-7.
- Iman is presenting slides 8-10 and 19-23.
- Quin is presenting slides 11-18.
- Austin is doing the live demo.

Problem statement

Addresses the issue of managing kitchen inventory and reducing food waste by tracking what's about to expire. Manually checking if the kitchen has a certain food item that may be tedious. It may also be difficult to keep track of items the kitchen is running out of

Project purposes

Address the problem by creating a user-friendly application that enables users to manage their kitchen inventory seamlessly.

Provide features such as inventory tracking, expiration date monitoring…

The application aims to simplify the kitchen management process.

Functional requirements

Expiration Date Tracking: The application should track the expiration dates of food items and notify users when items are about to expire.

Inventory Management: Users should be able to add, remove, and update items in their inventory.

User Authentication: Implement user authentication to ensure data privacy and security.

Notification System: Provide a notification system to alert users about expiring items and low inventory levels.

Search Functionality: Include a search feature to allow users to quickly find specific items in their inventory.

Multiple kitchen: Enhance user experience and streamline kitchen management.

Nonfunctional requirements

Data Integrity: Enter days to expiration; the app calculates the actual date. No incorrect entries allowed.

Volume Testing: Manages thousands of items smoothly. Shows a practical number of items for best performance.

Usability: Clear labels and error help.

Portability: Works on Windows, MacOS, and Linux without changes.

Use cases and actor

Use cases

User Updates Inventory: Allow users to add, remove, or update items in their inventory.

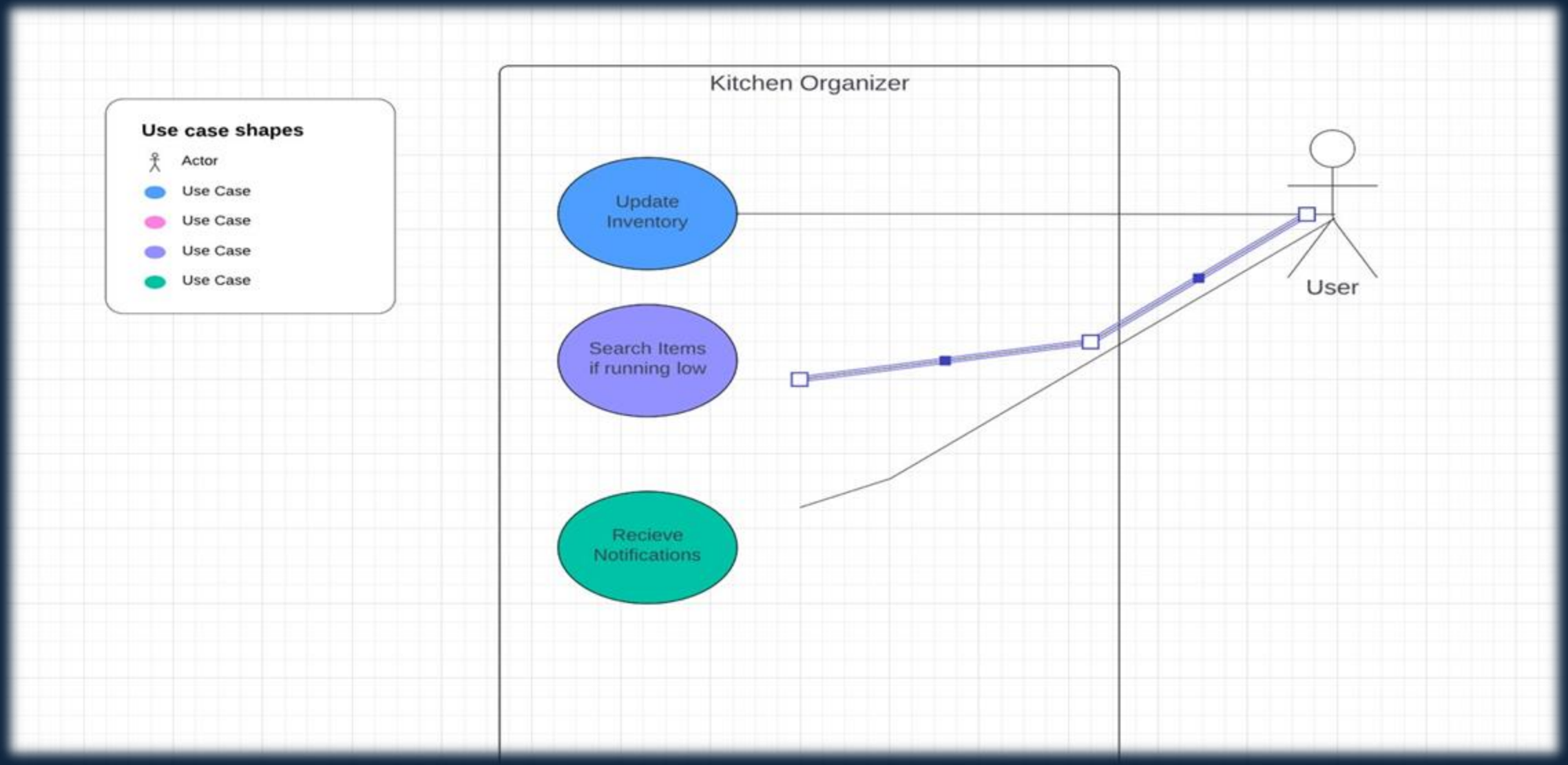
User Searches for Items: Enable users to search for specific items in their inventory.

User Receives Notifications: Notify users about expiring items and low inventory levels.

Actor

User

UML use case diagram



Project classes

FoodCollection: Represents a collection of food items.

InventoryItem (Abstract Class): Abstract class representing an inventory item.

Food: Represents a food item.

User: Represents a user of the application.

Notification: Represents a notification system for notifying users about low inventory or expiring items.

Notify (Interface): Interface defining the structure for notification classes to implement

Relations between classes

FoodCollection:	• Aggregates instances of the Food class.
InventoryItem (Abstract Class):	• Implemented by subclasses such as Food.
Food:	• Extends the InventoryItem class and includes additional attributes specific to food items, such as expiration date. • Implements methods for setting and retrieving food-specific attributes.
User:	• Aggregates instances of the FoodCollection class.
Notification:	• Associated with a specific user and their inventory. • Implements the Notify interface for sending notifications.
Notify (Interface):	• realization relationship between the Notification class and the Notify interface

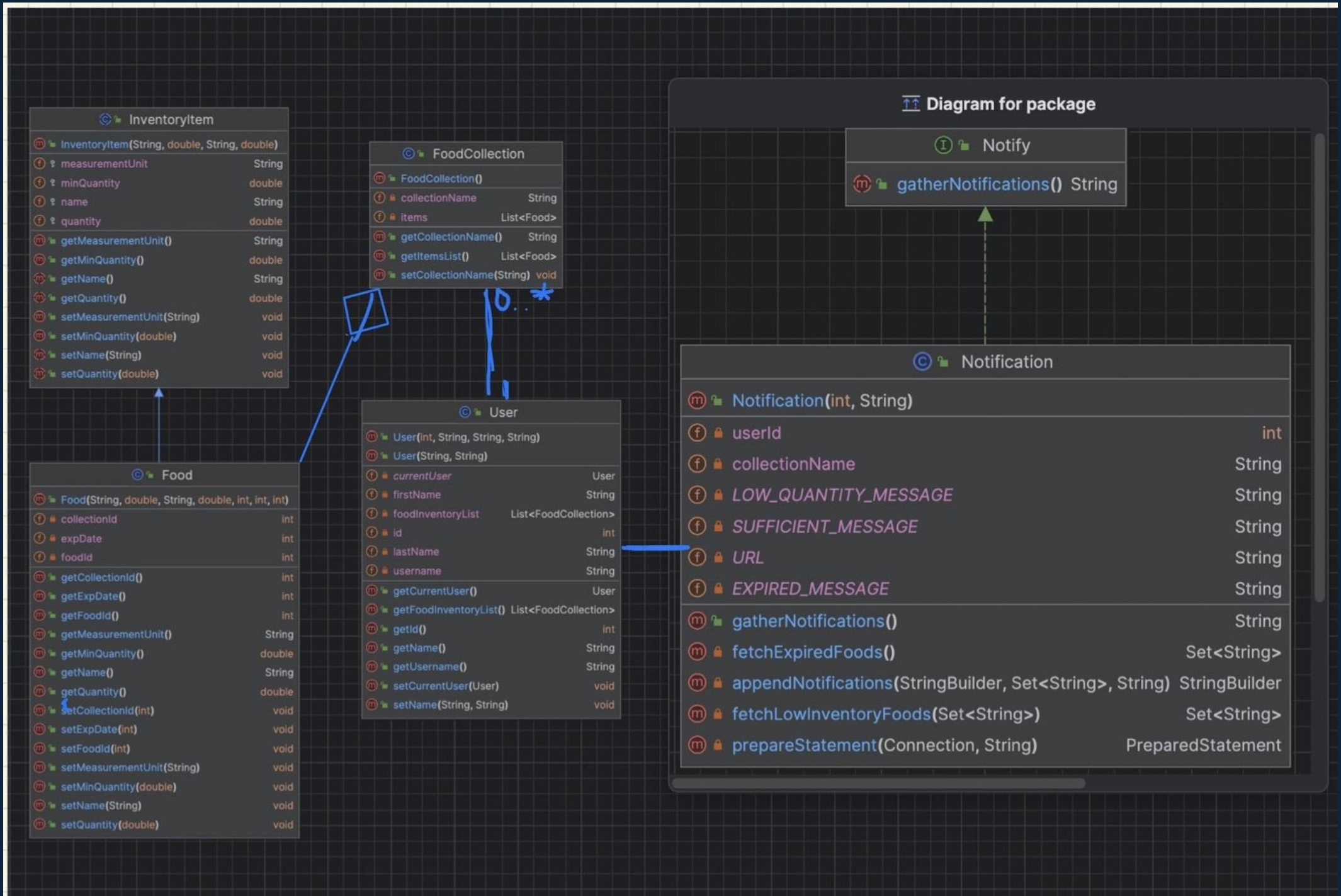
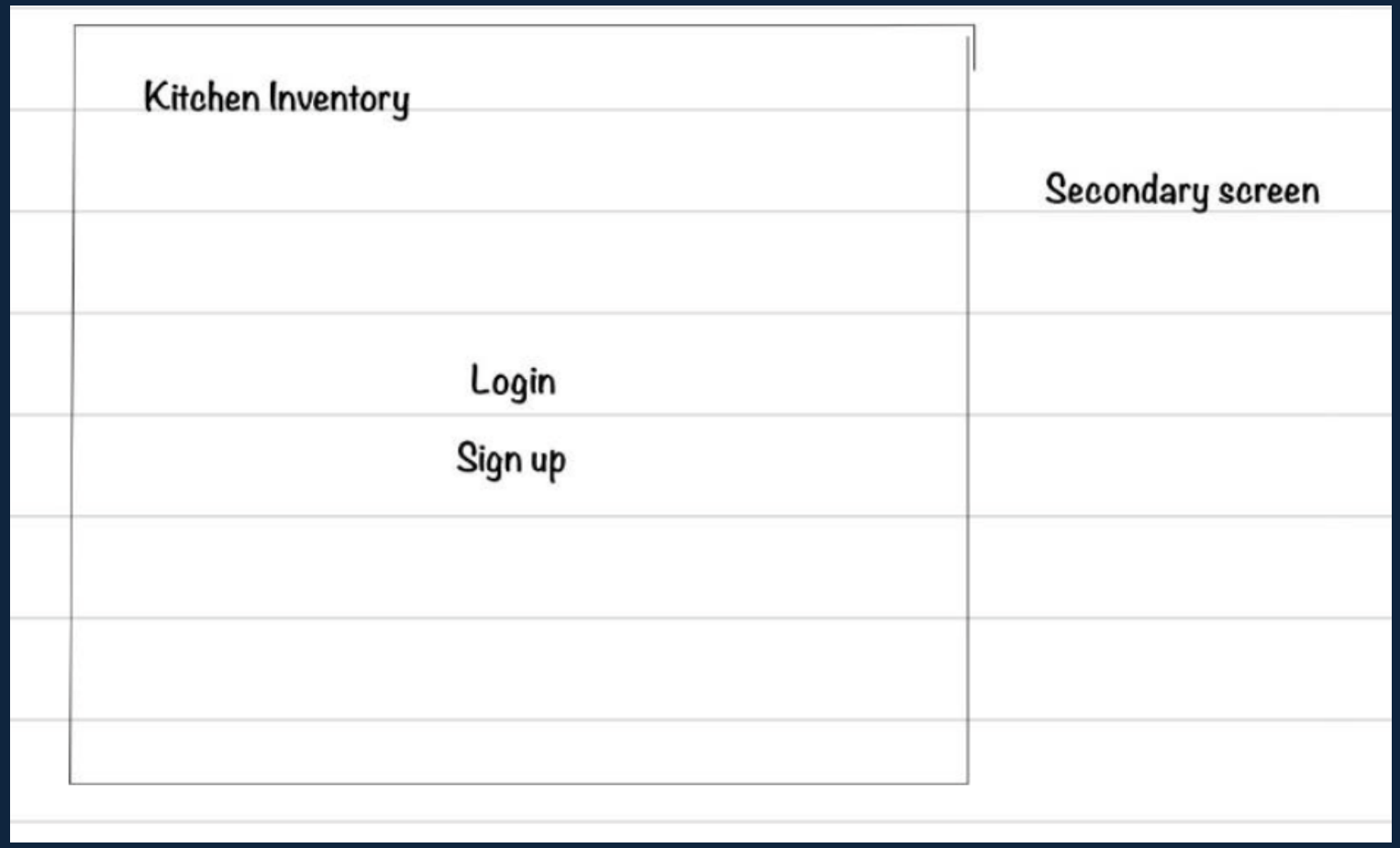


Diagram for package
InventoryItem
InventoryItem(String, double, String, double)
measurementUnit String
minQuantity double
name String
quantity double
getMeasurementUnit() String
getMinQuantity() double
getName() String
getQuantity() double
setMeasurementUnit(String) void
setMinQuantity(double) void
setName(String) void
setQuantity(double) void
FoodCollection
FoodCollection()
collectionName String
items List<Food>
getCollectionName() String
getItemsList() List<Food>
setCollectionName(String) void
Notify
gatherNotifications() String
User
User(int, String, String, String)
User(String, String)
currentUser User
firstName String
foodInventoryList List<FoodCollection>
id int
lastName String
username String
getCurrentUser() User
getFoodInventoryList() List<FoodCollection>
getId() int
getName() String
getUsername() String
setCurrentUser(User) void
setName(String, String) void
Food
Food(String, double, String, double, int, int, int)
collectionId int
expDate int
foodId int
getCollectionId() int
getExpDate() int
getFoodId() int
getMeasurementUnit() String
getMinQuantity() double
getName() String
getQuantity() double
setCollectionId(int) void
setExpDate(int) void
setFoodId(int) void
setMeasurementUnit(String) void
setMinQuantity(double) void
setName(String) void
setQuantity(double) void
Notification
Notification(int, String)
userId int
collectionName String
LOW_QUANTITY_MESSAGE String
SUFFICIENT_MESSAGE String
URL String
EXPIRED_MESSAGE String
gatherNotifications() String
fetchExpiredFoods() Set<String>
appendNotifications(StringBuilder, Set<String>, String) StringBuilder
fetchLowInventoryFoods(Set<String>) Set<String>
prepareStatement(Connection, String) PreparedStatement

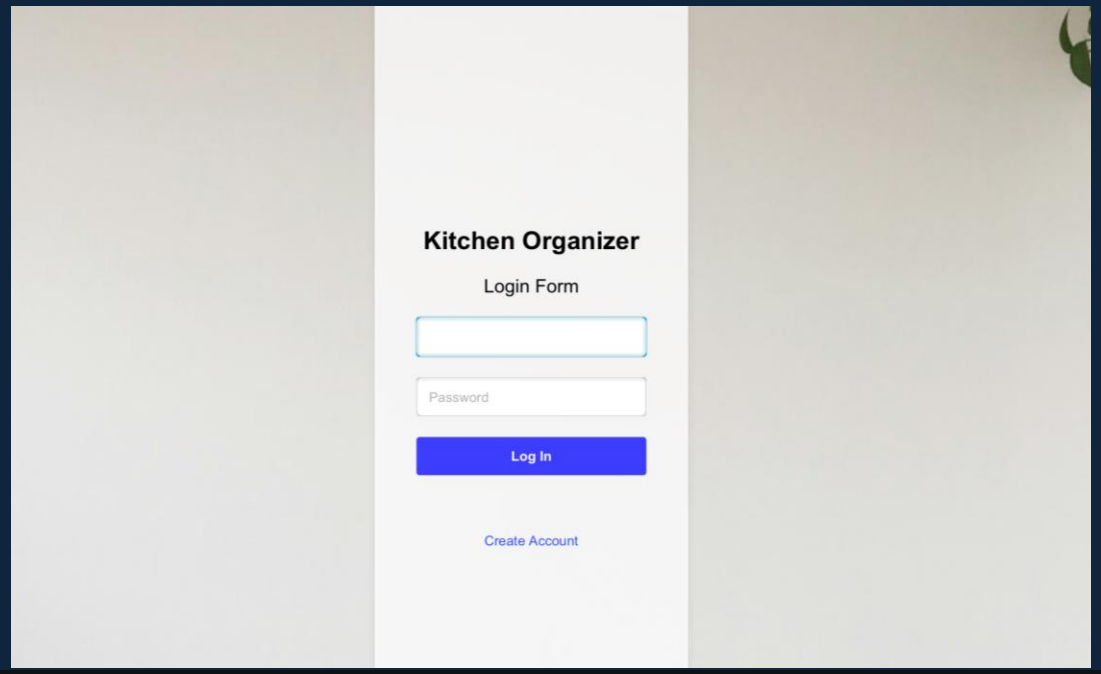
Database diagram

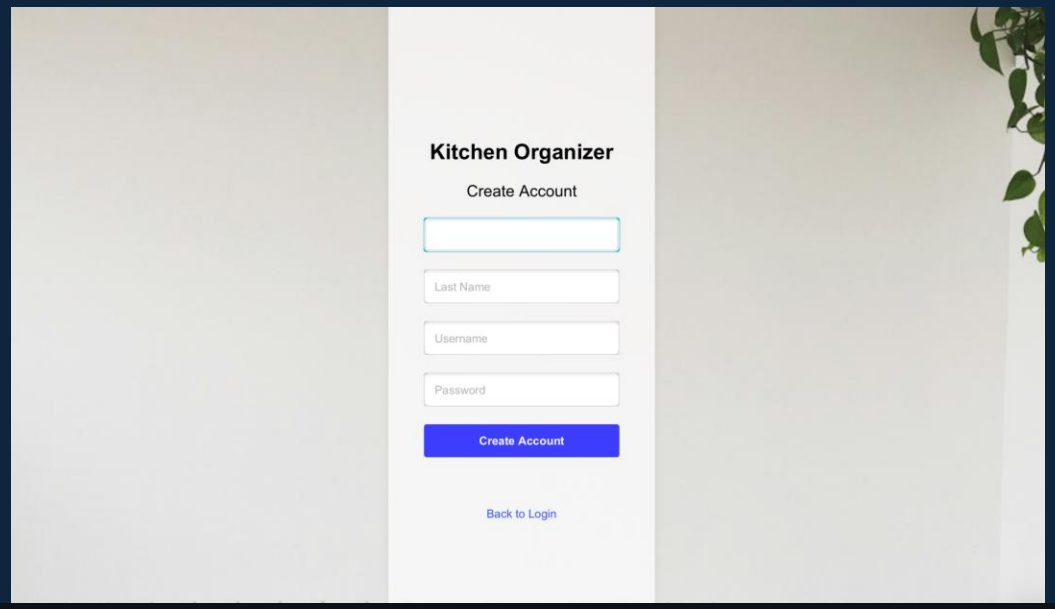
Login screen Interface



Original User Interface Wireframe

Login screen





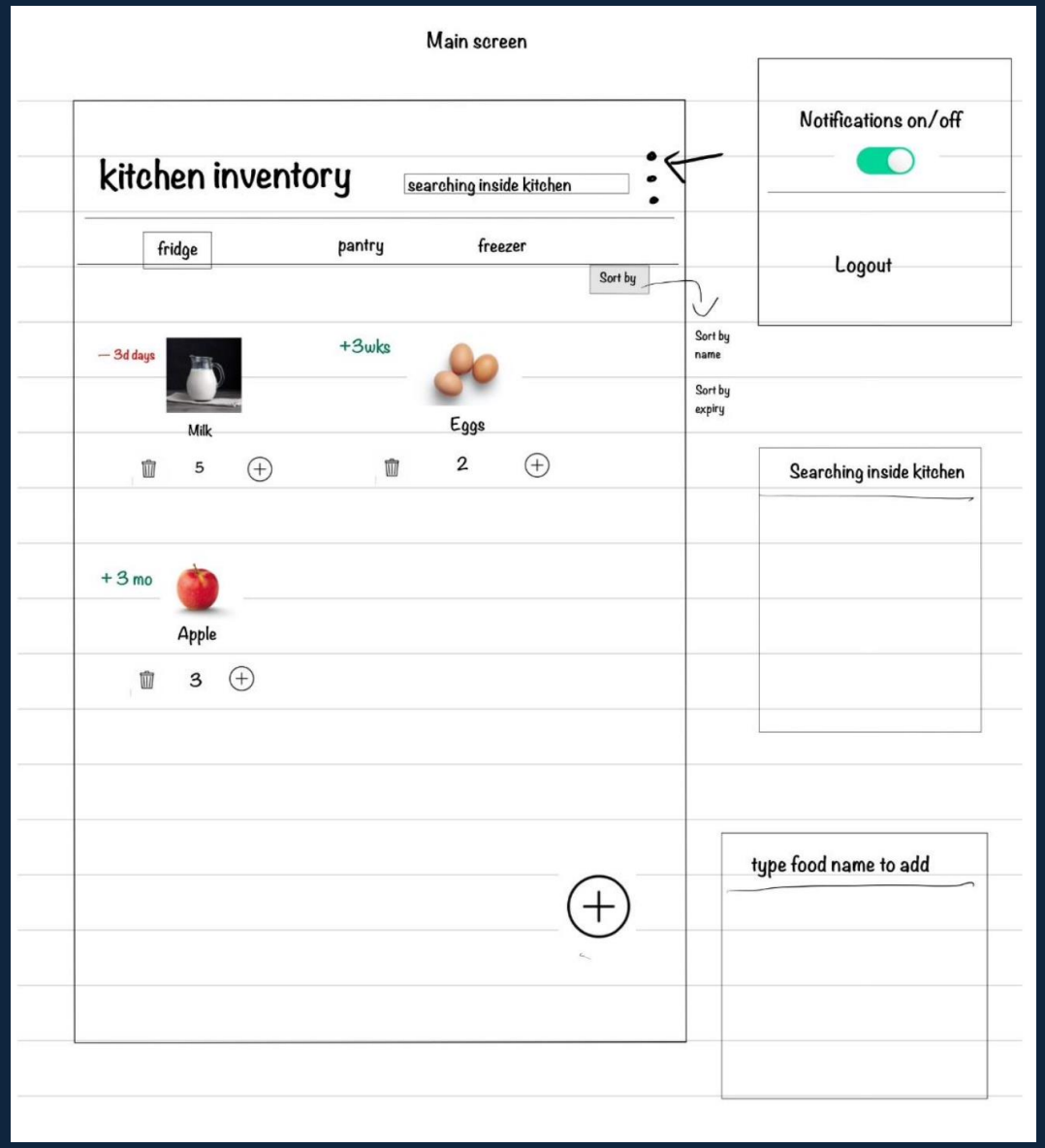

```
// Insert new user into the database
String sql = "INSERT INTO Users (username, password, firstName, lastName) VALUES (?, ?, ?, ?)";
```

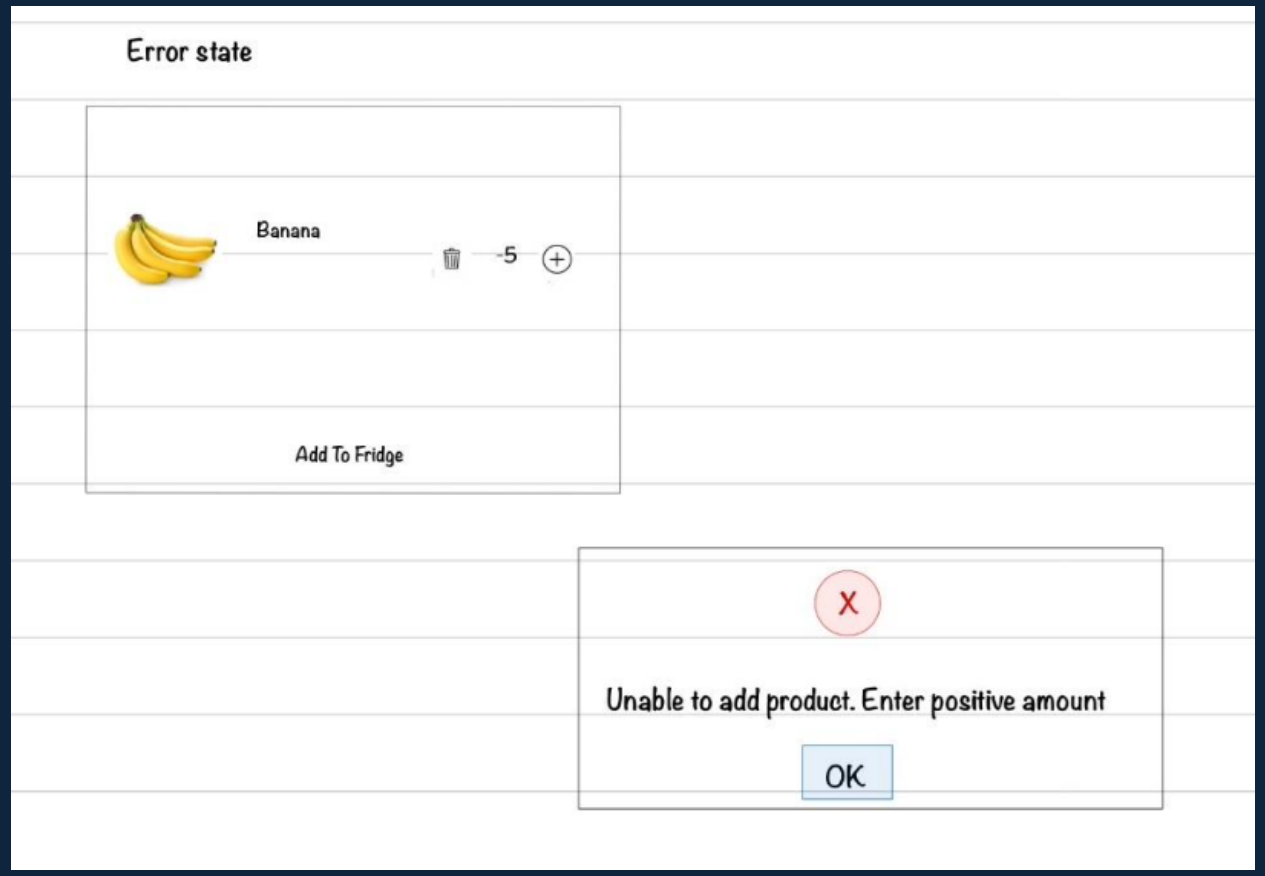
```
private boolean checkCredentials(String username, String password) {
    String sql = "SELECT id FROM Users WHERE username = ? AND password = ?";
```

```
if (checkCredentials(enteredUsername, enteredPassword)) {
```

```
private User fetchUserFromDatabase(String username) {
    User user = null;
    String sql = "SELECT * FROM Users WHERE username = ?";
```

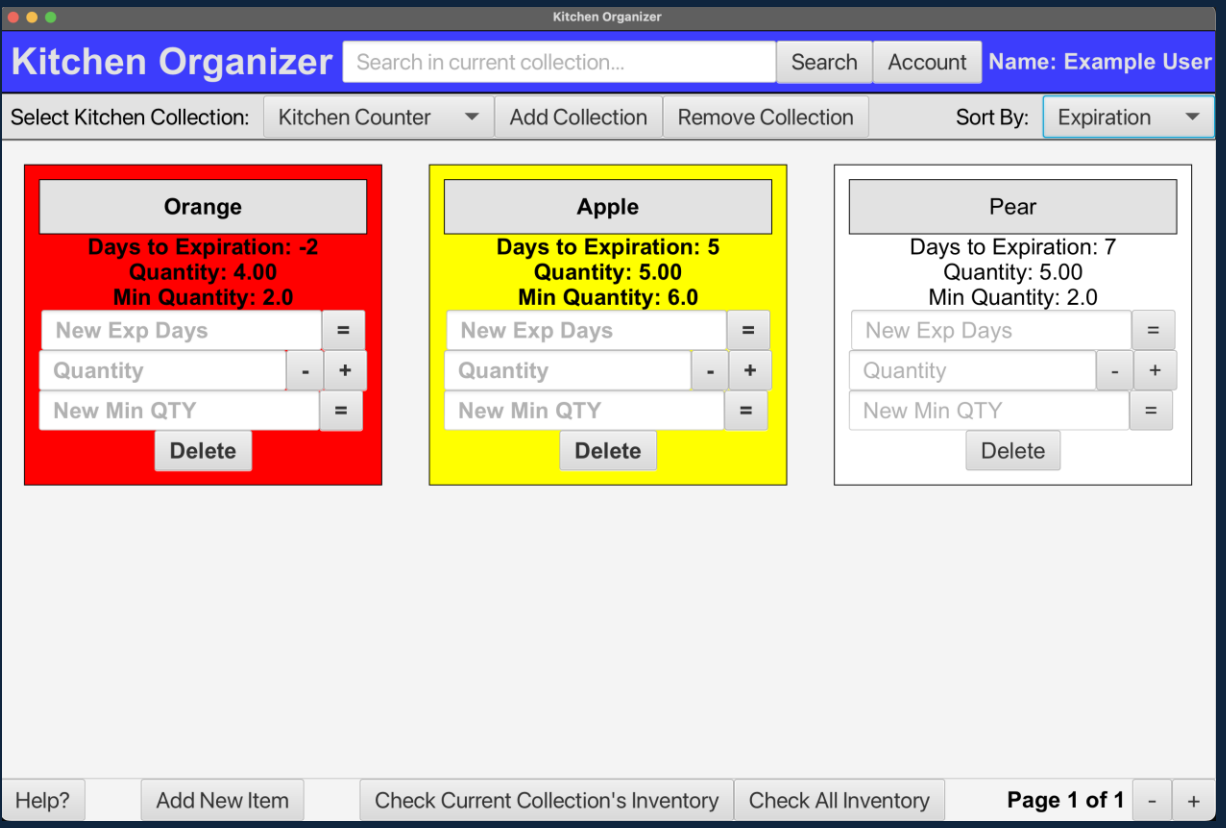
Main screen Interface

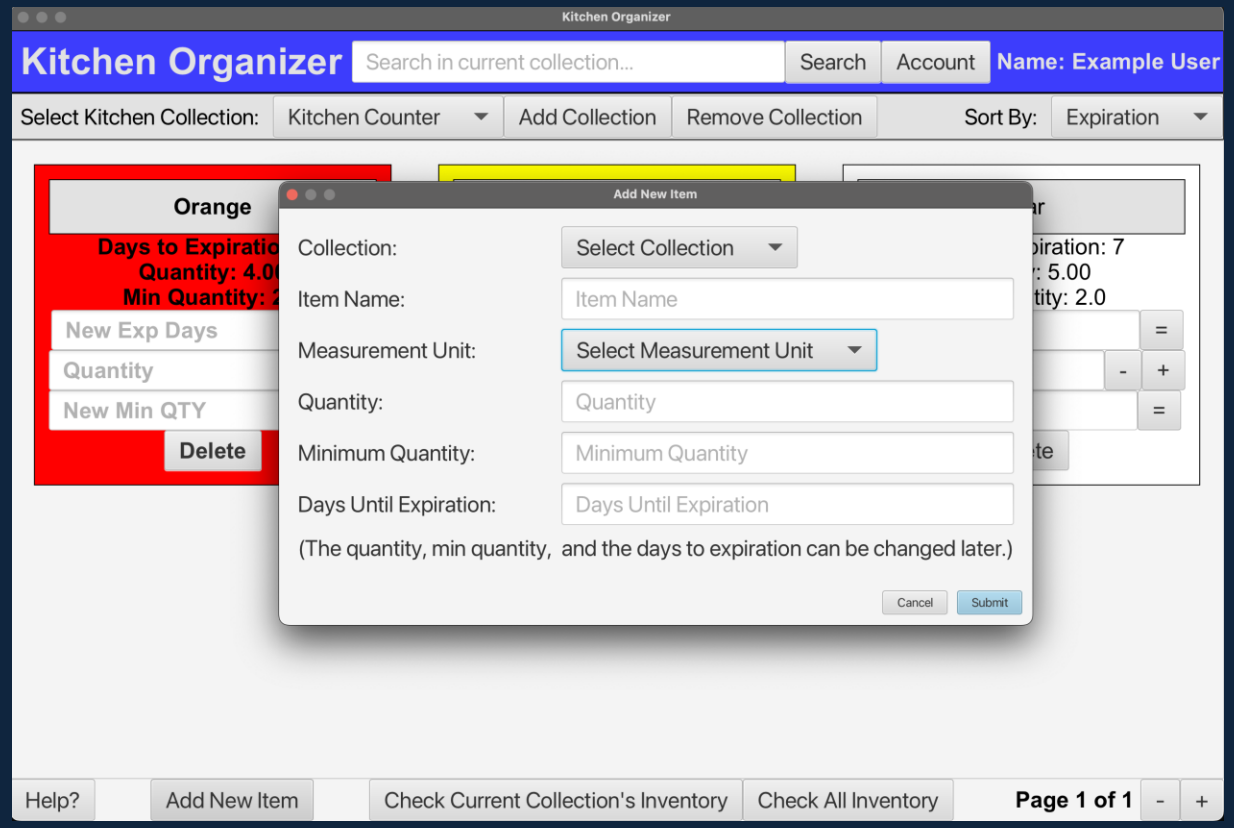




Original User Interface Wireframe

Main screen





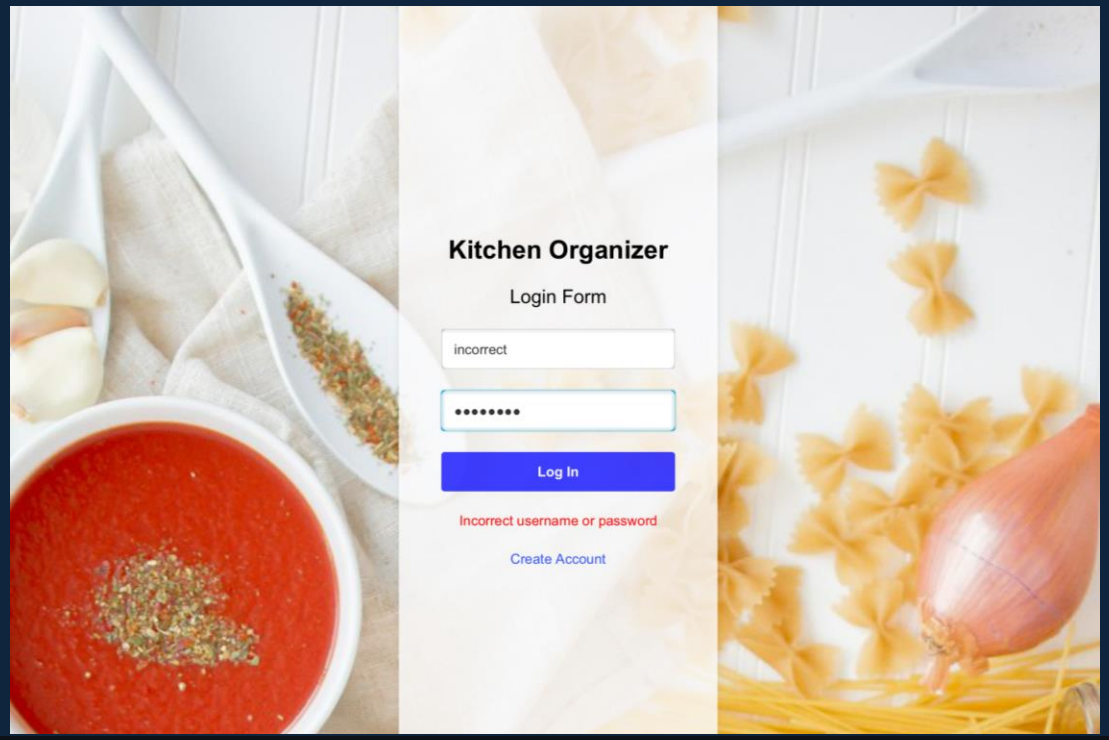

```
public static void addFoodToCollection(String collectionName, String name, double quantity, String measurementUnit, double minQuantity, int expDateDays) {
```

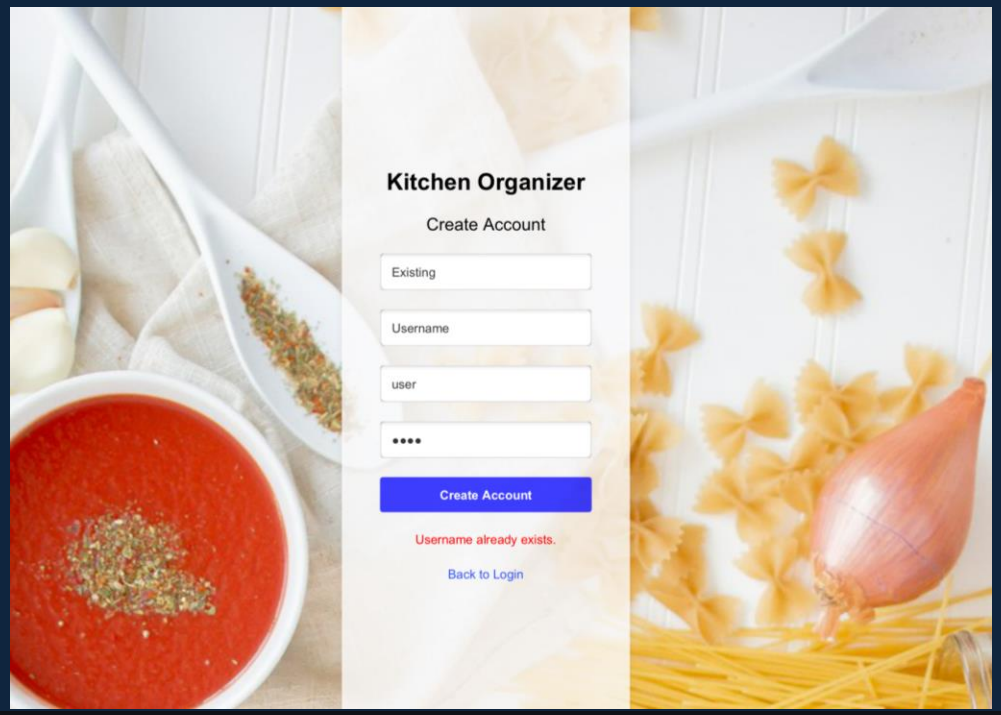
```
// Calculate the expiration date by adding expDateDays to the current date
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DAY_OF_YEAR, expDateDays);
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
String expDateStr = sdf.format(calendar.getTime());
```



```
String sql = "INSERT INTO Foods (collectionId, name, quantity, measurementUnit, minQuantity, expDate) VALUES (?, ?, ?, ?, ?, ?)";
```

Error States





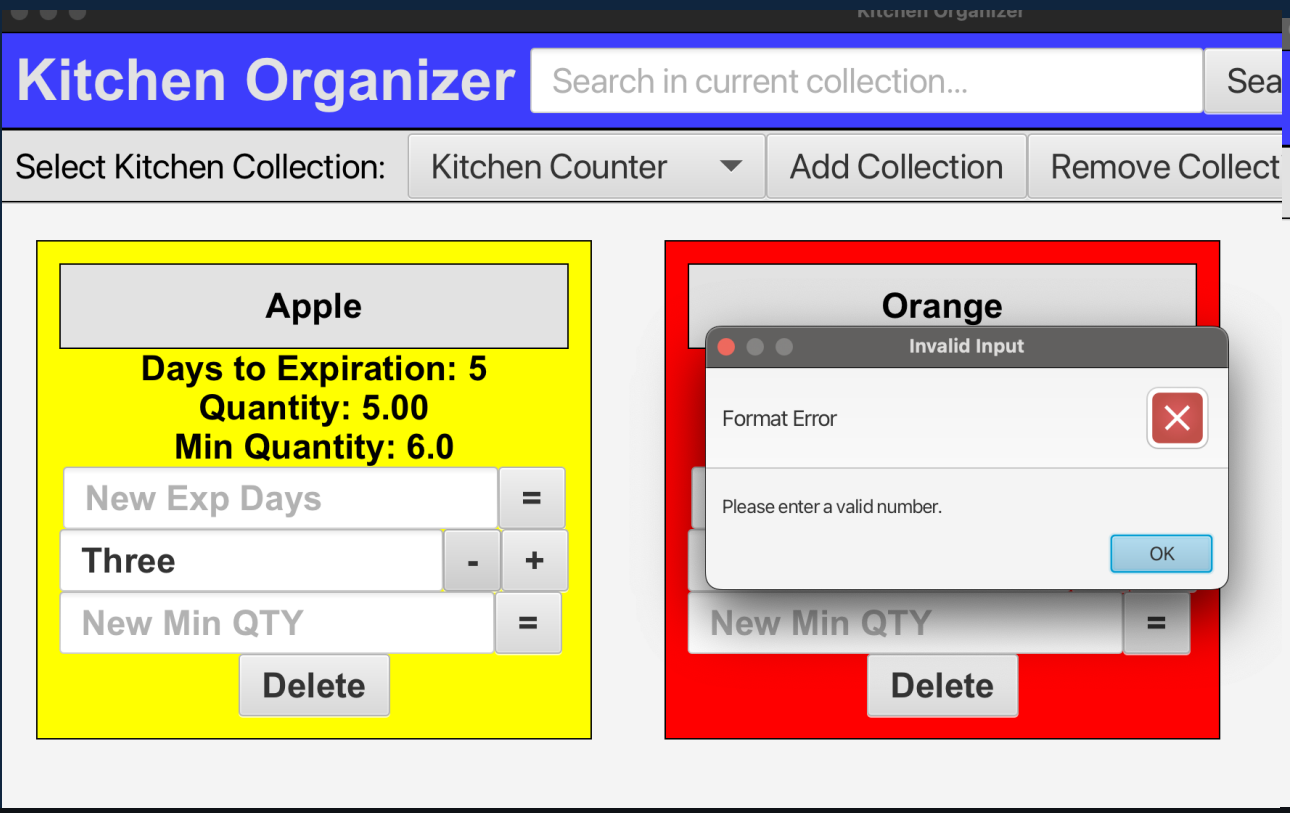

```
private boolean checkCredentials(String username, String password) {
    String sql = "SELECT id FROM Users WHERE username = ? AND password = ?";
```

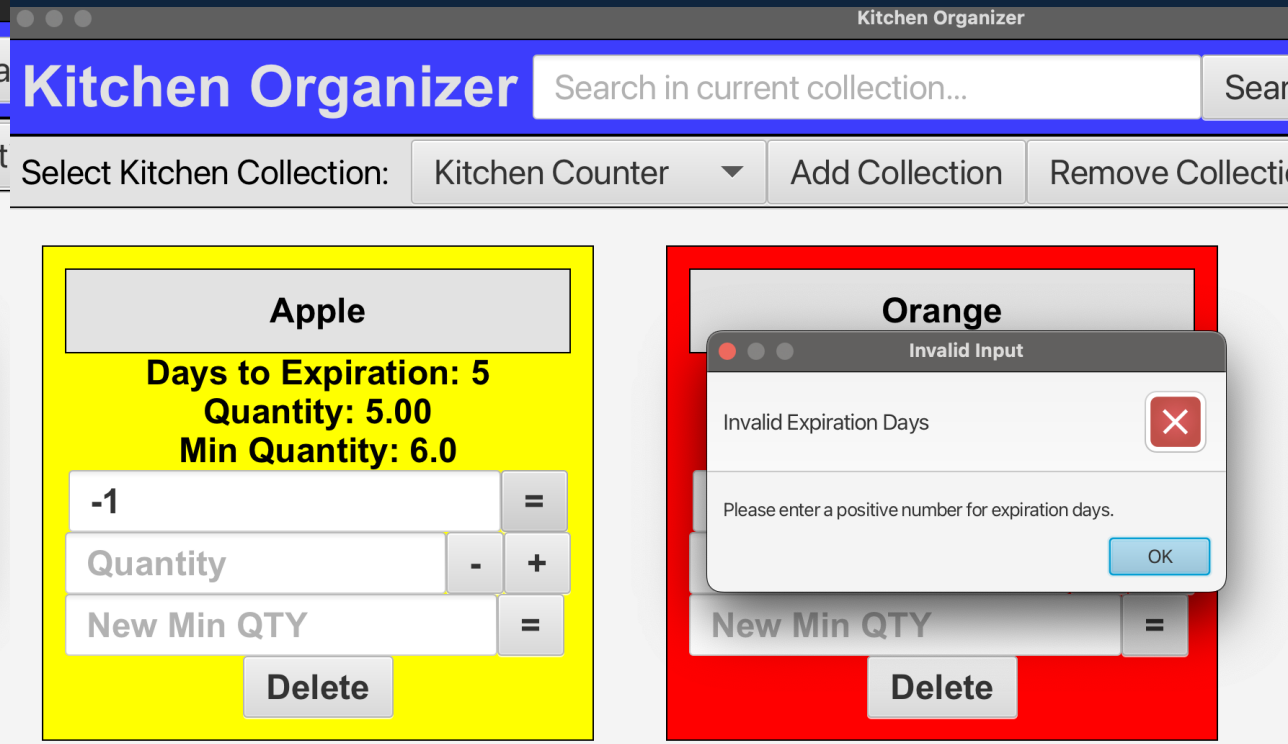
```
private boolean usernameExists(String username) {
    String sql = "SELECT COUNT(*) FROM Users WHERE username = ?";
```

```
if (checkCredentials(enteredUsername, enteredPassword)) {
```

```
} else {
    loginMessageLabel.setTextFill(Color.RED);
    loginMessageLabel.setText("Incorrect username or password");
}
```

Error States



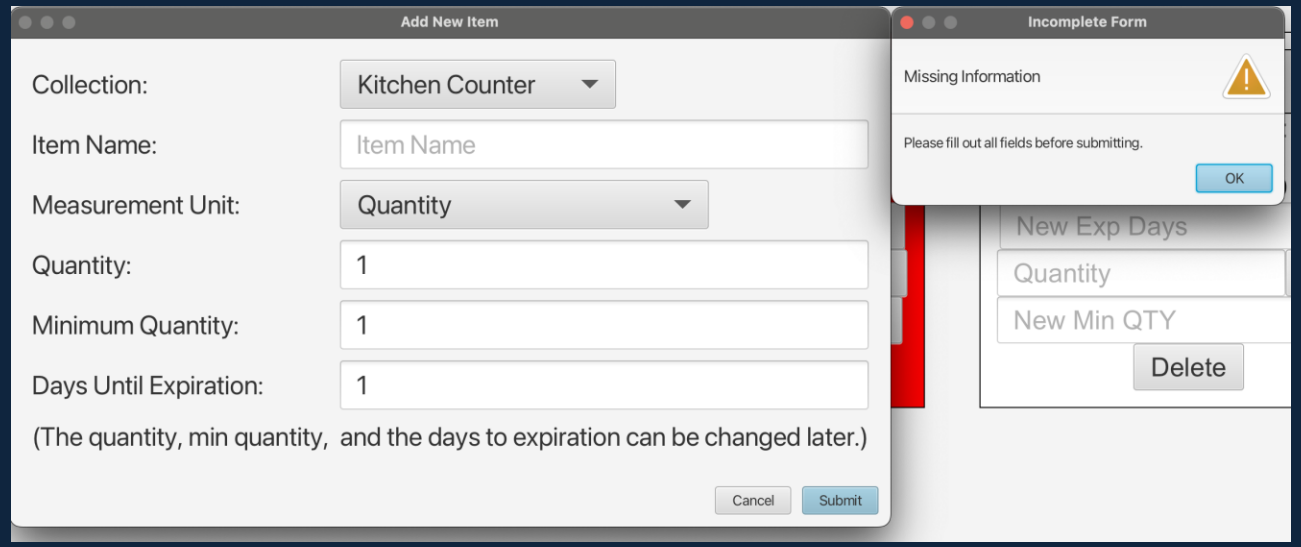



```java
try {
    double quantityChange = Double.parseDouble(usedQuantity.getText());
```

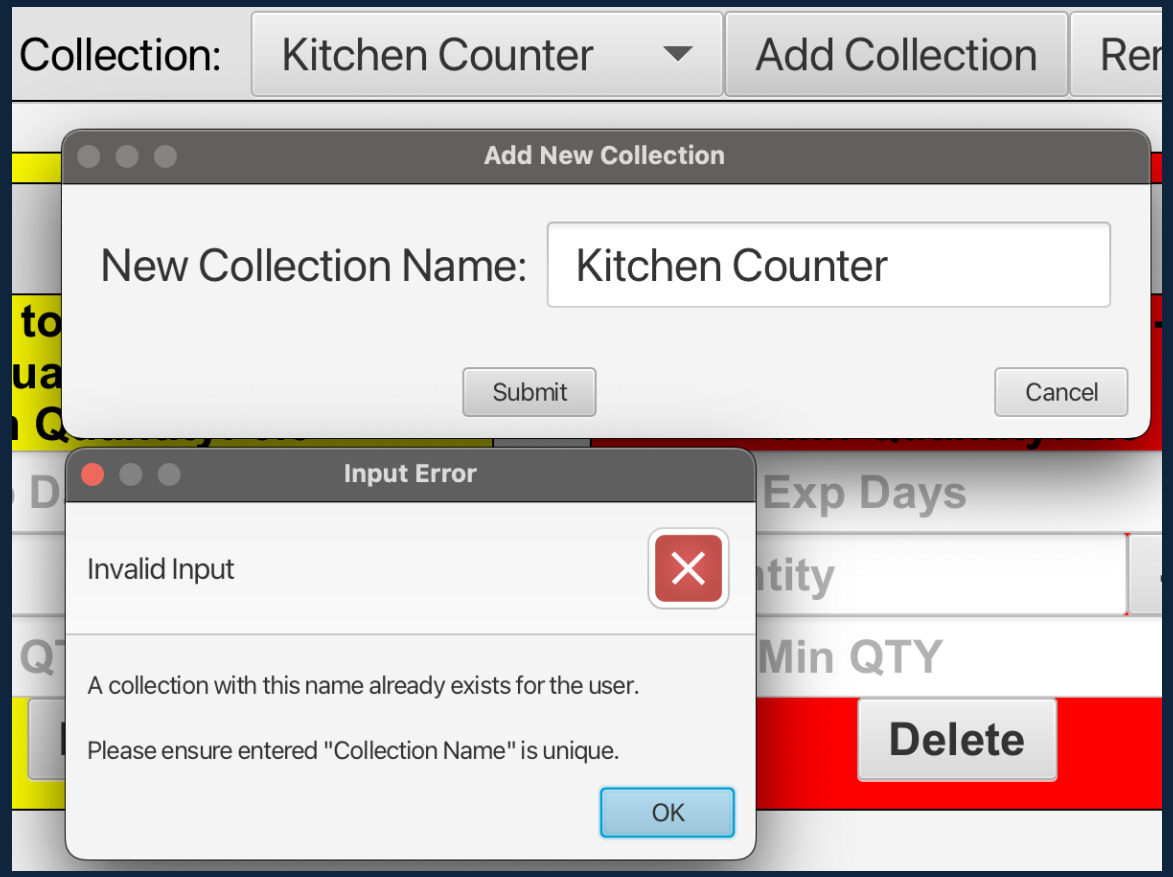
```java
} catch (NumberFormatException e) {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Invalid Input");
    alert.setHeaderText("Format Error");
    alert.setContentText("Please enter a valid number.");
    alert.showAndWait();
```

```java
if (quantityChange < 0) {
    // Alert the user about the negative input
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Invalid Input");
    alert.setHeaderText("Negative Quantity");
    alert.setContentText("Please enter a positive number to add.");
    alert.showAndWait();
    return; // Do not proceed with the operation
```

Error States




```
// Checks if all fields are filled
if (collections.getValue() == null ||
        nameField.getText().trim().isEmpty() ||
        measurementUnitDropdown.getValue() == null ||
        quantityField.getText().trim().isEmpty() ||
        minQuantityField.getText().trim().isEmpty() ||
        expDateField.getText().trim().isEmpty()) {
```




```
// Check if the collection name already exists for the user
String checkSql = "SELECT id FROM FoodCollections WHERE name = ? AND userId = ?";
String insertSql = "INSERT INTO FoodCollections (name, userId) VALUES (?, ?)";
```

Development process

- Organized Teamwork
- Code sharing (GitHub)
- Feature brainstorming
- Testing
- Divide and conquer
- Feedback

What went well?

Successful implementation of core features according to the functional requirements.

We were able to fulfill the requirements for the milestones on time.

We didn’t have many disagreements or conflicts in our group.

What didn’t go well?

Initial challenges in understanding and implementing certain features.

Time management could have been better since we had to rush to get certain things done by the deadline.

Awareness of tasks could have been better since we did not discuss all things that needed to get done in our meetings, and this affected our productivity.

What we could do differently?

Do most of the work a week or two before the deadline in case anything unexpected happens so we don't have to rush at the end.

Ensure that all tasks are discussed in meetings so we can collaborate effectively in accomplishing these tasks.

What we wish we knew

The importance of thorough planning and requirement analysis at the beginning of the project.

The significance of regular communication and collaboration to avoid misunderstandings and conflicts.


Slack bot
Floating emojis
File upload chat
New onboarding
SSO
Reporting
Library upgrades
API documentation 3.0
Auto-mute
App marketplace

Live Demo

Questions?




Thanks