Dan's Frappuccino Paradise

This is a web app designed to facilitate multiple aspects of a standard coffee shop experience in a digital environment.

Workspace Layout

The web app will be stored in this repository.

Organization

Team Members:

- John Belnap
- Caden Harris
- Trenton Peters
- Kollin Murphy

Version-Control Procedures

Team members should clone the repository in Kollin's account of the project "**dans-frappuccino-paradise**".
Before each meeting, collaborators should submit a pull request so we can monitor progress and discuss issues.
After making any significant change, team members should notify the team for feedback.
During development we will work primarily on feature branches which will merge back into master.

Tool Stack

This project uses Astro as a Server Side Renderer and SolidJS as a front-end framework. Additionally, we utilize a SQLite database and the Sequelize ORM.

- Pros:
  - Cutting-edge technology
  - Super duper fast
  - The majority of the site is pure HTML (with the addition of TailwindCSS and DaisyUI to make it easier)
  - Easy to learn
  - Can use any UI framework (Solid, Vue, React, Preact, Svelte, etc.)
    - Can also mix-and-match even within the project
  - Shows a lot of initiative on a resume
  - The API can be implemented using Astro, which is super intuitive
- Cons:
  - Little experience within the group
    - *Flip-side*: None of us have significant experience with Django or any other framework.

    - Limited documentation & examples
        - *Flip-side*: There is a lot of documentation for the UI framework(s) we decide to use, which is where most of our effort will be spent.

Setup Procedure

1. Make sure you have `node` installed.
2. Clone the repo, `cd` into it, and run `npm run init`. This will initialize a SQLite database with the proper schema and seed it with some initial data, including the users `dan`, `employee`, and `user`, each with the password `password123`.

Build Instructions

Just run `npm run build` and everything will work like a charm! It'll spin up a server on port 8080. View the front-end in the browser at `http://localhost:8080`.

Unit Testing Instructions

The unit tests address several commonly identified uses including those found in the use case diagrams. The unit tests are located in @@@@. All unit test will be run by default, but it will be possible to run specific unit tests by inputting the number corresponding to the same use case. It is possible to test the following cases @@@@.

System Testing Instructions

1. Make sure you have `node` installed: node.
2. Clone the repo, `cd` into it, and run `npm run init`. This will initialize an SQLite database with the proper schema and seed it with some initial data, including the users `manager`, `employee`, and `user` @@@@, each with the password `password123`.
3. Run `npm run build`. This will start up a server on port 8080. View the front-end in the browser at `http://localhost:8080`. The current user types are @@@@, and the testing password for each one is: `password123`.

Project Plan

Project Summary

To create a web application that serves all the core functionality of a coffee shop. This includes user-side functions such as ordering and purchasing coffee, seeing the user wallet, and adding funds to the user wallet, employee-side tasks such as tracking hours worked and marking completed orders, and manager-side tasks such as paying employees and ordering new supplies.

Team Organization

We plan to be an evenly organized team, with rotating scrum masters. Since Kollin has the most familiarity with the tech stack, he will serve as the first scrum master. Code reviews will be implemented so that someone in the group reviews any code that is merged.

Software Development Process

The development will be broken up into five phases. Each phase will be a little like a Sprint in an Agile method and a little like an iteration in a Spiral process. Specifically, each phase will be like a Sprint, in that work to be done will be organized into small tasks, placed into a “backlog”, and prioritized. Then, using on time-box scheduling, the team will decide which tasks the phase (Sprint) will address. The team will use a Scrum Board to keep track of tasks in the backlog, those that will be part of the current Sprint, those in progress, and those that are done.

Each phase will also be a little like an iteration in a Spiral process, in that each phase will include some risk analysis and that any development activity (requirements capture, analysis, design, implementation, etc.) can be done during any phase. Early phases will focus on understanding (requirements capture and analysis) and subsequent phases will focus on design and implementation. Each phase will include a retrospective.

Phase	Iteration
1.	Phase 1 - Requirements Capture
2.	Phase 2 - Analysis, Architectural, UI, and DB Design
3.	Phase 3 - Implementation
4.	Phase 4 - More Implementation and Testing

We will use Unified Modeling Language (UML) to document user goals, structural concepts, component interactions, and behaviors.

Communication Policy

We plan to meet, either physically or virtually, at least once a week. Our primary method of communication will be via discord, where we already have a group set up and have met. When we meet, we will go over what we have completed over the last week and what we plan to accomplish over the next week. This Once a week minimum doesn’t mean that we aren’t able to ask questions and request assistance throughout the week.

Risk analysis

Since we are building what is in essence a proof of concept project that won’t really be deployed, the main risk is that we don’t complete the project to the required specification and our grades suffer as a result. To mitigate this, we will set achievable and timely goals that will allow us to complete the project in time.

In terms of the risk associated with the individual sections of the project:

Database

- The database is essentially the heart of the assignment, so while the likelihood that it will fail is low, we do need to be careful as any consequences here have the potential to break the entire project.

Login

- Login is also important, as the access to any individual page is determined solely by the user’s account. Some of us have experience, so the likelihood of error is lower, but again we need to make sure that this system is solid.

UI

- All of us have experience in HTML, which is a large portion of how the website will be built. Because of that, risk is relatively low with consequences that are fairly easy to resolve.

Reference to the README.md

Refer to the README.md for more info

Requirements Definition

Introduction and Context

Dan has opened up a frappuccino restaurant and is in need of software to serve as a multi-purpose management and sales system for all sales and store activity to go through. This project aims to create a system to allow users of all roles to efficiently do what they need to do at the restaurant.

Customers and Employees will be able to make orders from an assortment of customizable drinks. When purchasing, customers will be able to pick them up with ease. Similarly for employees, creating orders will add them to the queue for baristas to prepare. They also have the ability to log their work hours within the same system.

It will allow the manager/store owner to order inventory and keep track of items in stock, pricing, employees, store balance, and work hours. They have the ability to create, edit, and remove employees that can be paid for their work time based on their flat rate of $15/hour.

By giving all users an easy way to get what they need out of Dan's Frappuccino Paradise, the multi-user software will help customers, employees, and managers alike in achieving a good frappuccino experience.

Functional Requirements

1. **Authentication**
    1. Available views are determined by the Account Role
        1. Visitors to the site must have an Account prior to accessing any pages other than sign in or create account
    2. Authentication state is maintained across site visits using a cookie to save an authorization token
    3. Account Usernames are not case sensitive
    4. Account Passwords are encrypted
    5. Account Passwords have a minimum character count of 8
    6. There are three distinct Account Roles: User, Employee, and Manager
2. **Placing an Order**
    1. An authenticated Account ("the Account") is presented with a list of all Products, each with an image and a price
    2. When the Account selects a Product, they are taken to a Product page which lists the Visible ProductIngredients, each with a quantity, and a size selector, from which they can choose Mini (8 oz), Medium (16 oz), Massive (24 oz)
        1. Changing the Size will modify the count of each ProductIngredient by a scale of 1x (Small), 2x (Medium), or 3x (Large), which will in turn affect the item price

      2. The Account cannot remove default ProductIngredients from the item
      3. The Account can add new ProductIngredients to the item, and remove those items which they have added
      4. The Account can click an Add To Order button which will do the following:
         1. Create an Order if there is no unpaid order for the associated Account
         2. Add an OrderProduct to the Order
         3. Add any applicable OrderAddOns to the OrderProducts
   3. The Account has a Cart, which displays the Products and their associated AddOns from the Account's most recent unpaid Order
      1. Products can be removed from the Cart
   4. The Cart has a Place Order button
      1. Following a successful purchase, the Account is directed to an Order page which displays the current fulfillment status of the Order
3. **User Home**
   1. The User Home is accessible by an Account of any Role
   2. This page presents the user with a list of their orders, sorted by createdAt descending
      1. OrderProducts are listed below each order
      2. There is a dropdown menu next to each OrderProduct, which presents some options to:
         1. Favorite it, which will automatically add it the favorite orders on the same page
         2. Cancel the order, which will refund the money to their account and mark the order as cancelled
   3. This page also presents a numeric entry from which the user can add funds to their Balance
   4. This page also has a list of favorite drinks, which can be easily reordered
      1. When an Order is favorited / unfavorited, this list should dynamically update
4. **Employee Home**
   1. The Employee Home is accessible by an Employee or a Manager
   2. Can make an Order on behalf of an Account by searching their username
   3. Marks Orders as fulfilled
   4. Able to log their Hours
5. **Manager Home**
   1. The Manager Home is only accessible by a Manager
   2. Can add money to Store Balance with a simple numeric entry, which creates a Transaction
   3. Can distribute money to Employees based on their hours entered, which deducts from the Store Balance and creates a Transaction
      1. Employees are paid a flat rate of $15 / hr
   4. Can order inventory items, which increases the QuantityOnHand for the ProductIngredient, deducts from the Store Balance, and creates a Transaction
   5. Can change the percentPriceModifier, smallBasePrice, mediumBasePrice, largeBasePrice

    6. Has access to a table that displays all Products, each with a bulleted list of its ingredients, the cost of each, a total calculated price, and a delete button
        1. Deleting a Product adds an isDeleted flag to the model to prevent destroying current and past orders
    7. Can create new Products with a form to input the Product Name, Image Url, a list of all ProductIngredients with checkboxes to make them toggle-able, a dynamically calculated price, and a submit button which creates the new Product and its associated ProductIngredients
    8. Can create Employees by assigning them a username and a password
    9. Can delete Employees or demote them to User

Non-functional Requirements

6. **Orders**
    1. The price of an OrderProduct is calculated as follows:where *size* is 1 for small, 2 for medium, and 3 for large; *modifier* is percentPriceModifier; *base* is smallBasePrice, mediumBasePrice, or largeBasePrice
    2. The price of an Order is the sum of all of its OrderProducts
    3. Failure conditions of an order
        1. Whenever an Order fails to be paid, it should not be deleted or modified
        2. When the Account's Balance is less than the calculated Order total, an error message is displayed informing the user that they have insufficient funds
    4. Actions to take upon successful payment
        1. A Transaction is created
        2. The Account's Balance is deducted by the Order Price
        3. The Order is marked as Paid
        4. The QuantityOnHand is decremented for each ProductIngredient and each OrderAddOn
            1. The quantity to deduct is scaled by the Product size: 1 for small, 2 for medium, 3 for large
7. **The project will employ a database with the following entities:**
    1. StoreConfig
        1. Key (string)
        2. Value (float)
    2. Product
        1. Slug (string)
        2. Name (string)
        3. ImageUrl (string)
        4. IsDeleted (boolean default false)
    3. ProductIngredient
        1. Slug (string)
        2. Name (string)
        3. Price (float)
        4. QuantityOnHand (integer)
    4. Account

        1. Role (User | Employee | Manager)
        2. Username (string)
        3. Password (hashed string)
        4. Balance (float)
        5. isDeleted (boolean default false)
    5. AccountFavorite
        1. AccountId
        2. OrderProductId
        3. Name (nullable string)
    6. Transaction
        1. StoreId
        2. AccountId (nullable integer)
        3. Type (FundsAdded | OrderPaid | InventoryPurchased)
        4. OrderId (nullable integer)
        5. Price (float)
    7. Order
        1. AccountId
        2. Paid (boolean default false)
        3. Status (Created | Cancelled | Fulfilled)
    8. OrderProduct
        1. OrderId
        2. ProductId
        3. Size (Small | Medium | Large)
    9. OrderAddOn
        1. OrderProductId
        2. ProductIngredientId
    10. Hours
        1. AccountId
        2. HoursWorked
        3. Paid (boolean default false)
8. **The project will contain an initialization setup script which seeds the database with the following:**
    1. StoreConfig (Key: ‘balance’, Value: 10000)
    2. StoreConfig (Key: ‘percentPriceModifier’, Value: 1.50)
    3. StoreConfig (Key: ‘smallBasePrice’, Value: 2.00)
    4. StoreConfig (Key: ‘mediumBasePrice’, Value: 4.00)
    5. StoreConfig (Key: ‘largeBasePrice’, Value: 5.00)
    6. 1 Manager (Username: ‘dan’, Password: ‘password123’, Balance: $0)
    7. 1 Employee (Username: ‘employee’, Password: ‘password123’, Balance: $0)
    8. 1 User (Username: ‘user’, Password: ‘password123’, Balance: $10)

Future Features

1. Deploy to Netlify to allow anyone around the globe to order our delicious fraps.
2. Allow purchasing a drink as a guest (without a login).
3. Implementing an integration with Stripe to use actual currency.

Glossary

This section explains important terms and definitions.

Customer - a user who wishes to purchase drinks with the available customization options

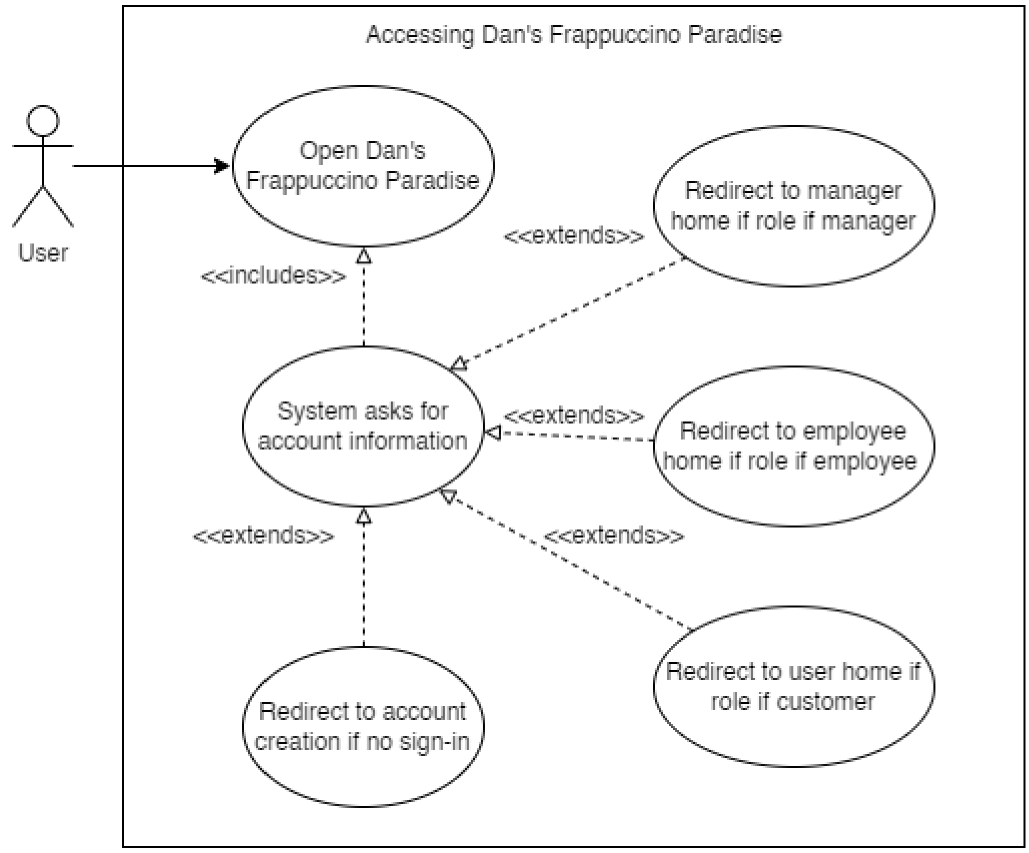
Employee - a user that receives money for work hours and can make orders on behalf of customers

Manager - the superuser with total access that can order inventory, pay and edit employees, and manage the store balance

User - any of the four types of users of Dan's system

Users and their Goals

Figure 1 - User attempts access of the application


Accessing Dan's Frappuccino Paradise
User
Open Dan's Frappuccino Paradise
<<includes>>
<<extends>>
Redirect to manager home if role if manager
System asks for account information
<<extends>>
Redirect to employee home if role if employee
<<extends>>
<<extends>>
Redirect to account creation if no sign-in
Redirect to user home if role if customer

Participating Actor: User

Entry Condition:

- User wants to access Dan’s Frappuccino Paradise (To order, view the menu, do employee tasks, manager tasks, or edit account information.)

Exit condition:

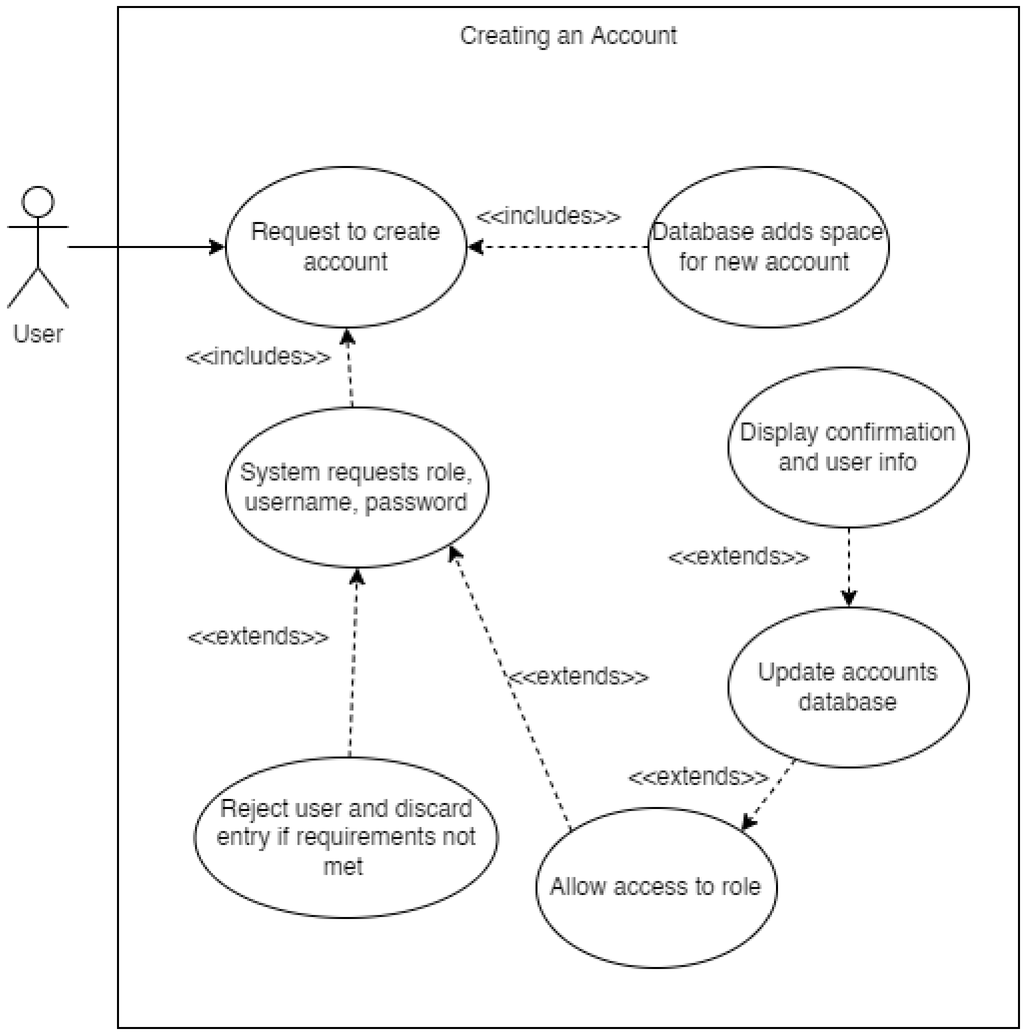
- User has gained access
- User is rejected
- User no longer wants to use the app

Event flow:

1. User opens the application
2. System requires a sign in
    1. Create an account

    2. Sign in
3. Display role-specific pages
    1. User home
    2. Employee home
    3. Manager home

Figure 2 - Customer creates account


Creating an Account
User
Request to create account
<<includes>>
Database adds space for new account
<<includes>>
System requests role, username, password
Display confirmation and user info
<<extends>>
<<extends>>
<<extends>>
Update accounts database
<<extends>>
Reject user and discard entry if requirements not met
Allow access to role

Participating actor: User

Entry Conditions:

- Customer wants to create an account

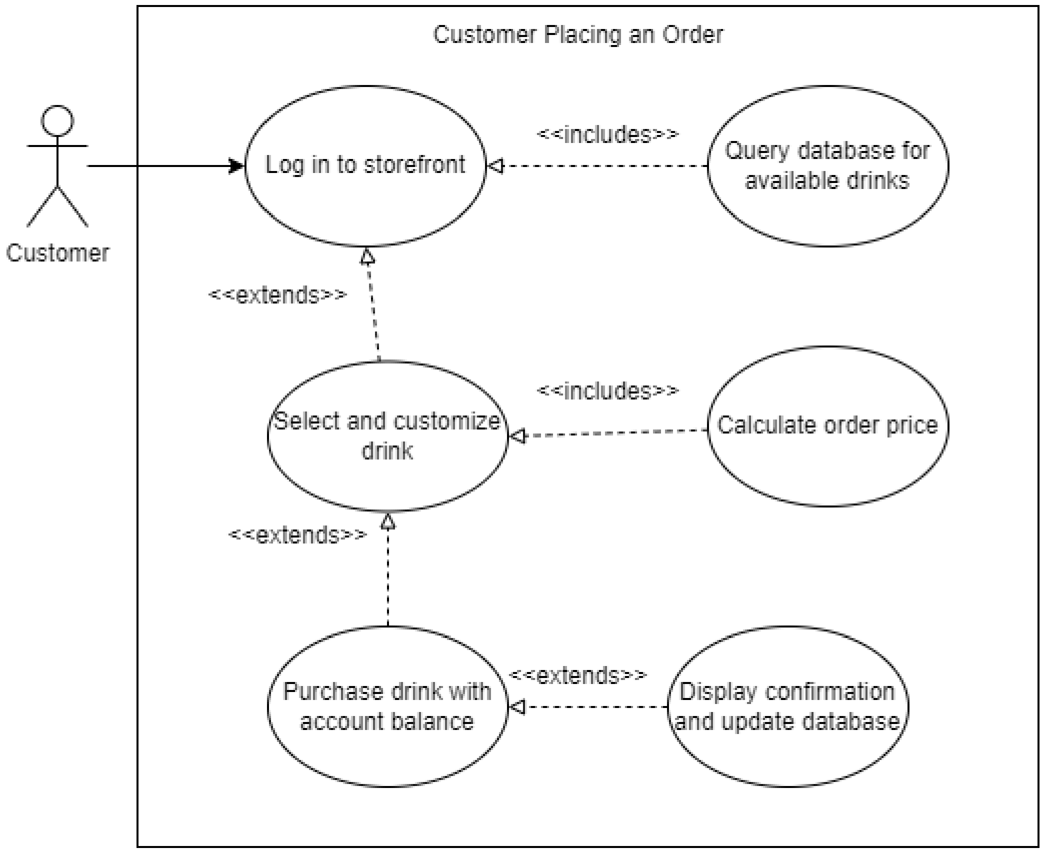
Exit Conditions:

- Successful account creation
- Customer decides to not make an account

Event Flow:

- User makes creation request
- System inquiries user information (username, password, role)
- Allow user into application with role-based permissions

Figure 3 - Customer places an order


Customer Placing an Order
Customer
Log in to storefront
<<includes>>
Query database for available drinks
<<extends>>
Select and customize drink
<<includes>>
Calculate order price
<<extends>>
Purchase drink with account balance
<<extends>>
Display confirmation and update database

Participating actor: Customer

Entry Conditions:

- Customer wants to order a frappuccino

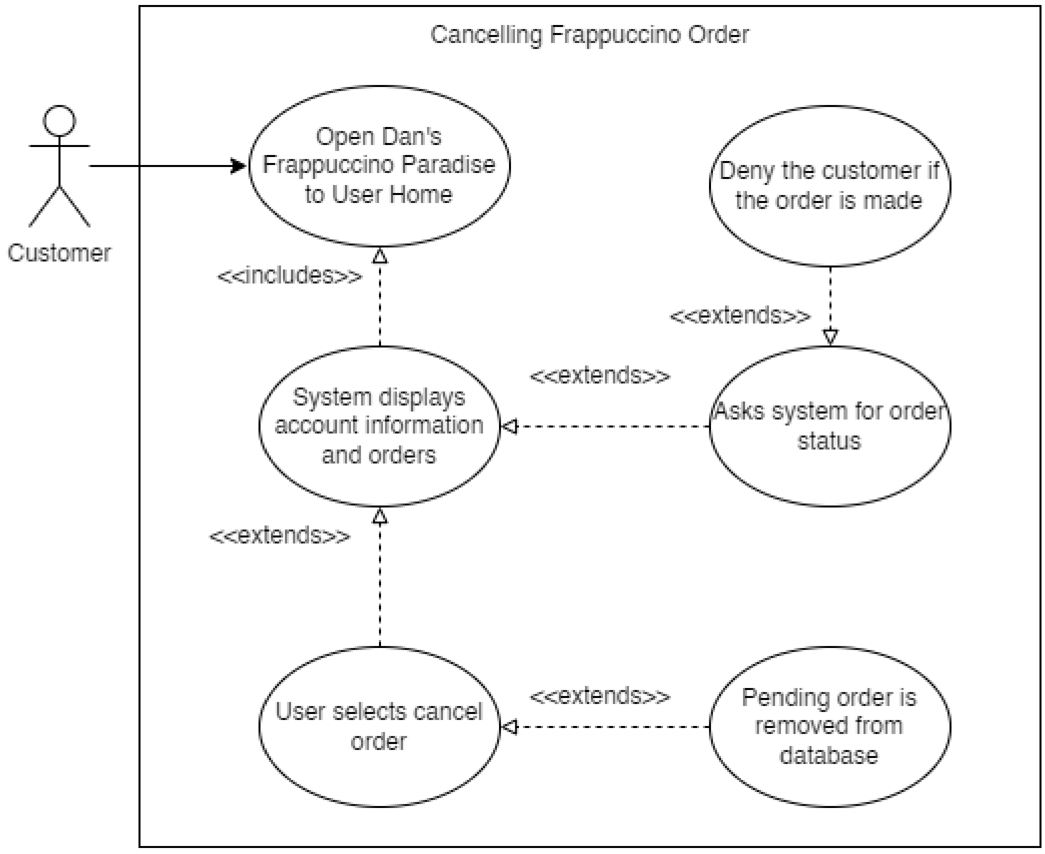
Exit Conditions:

- Successful order

Event Flow:

- User logs in
- System displays drink options for them to customize
- Customer purchases a drink, depleting their balance and receiving confirmation

Figure 4 - Customer cancels order


Cancelling Frappuccino Order
Customer
Open Dan's Frappuccino Paradise to User Home
Deny the customer if the order is made
<<includes>>
<<extends>>
<<extends>>
System displays account information and orders
Asks system for order status
<<extends>>
User selects cancel order
<<extends>>
Pending order is removed from database

Participating actor: Customer

Entry Conditions:

- Customer wants to cancel an order

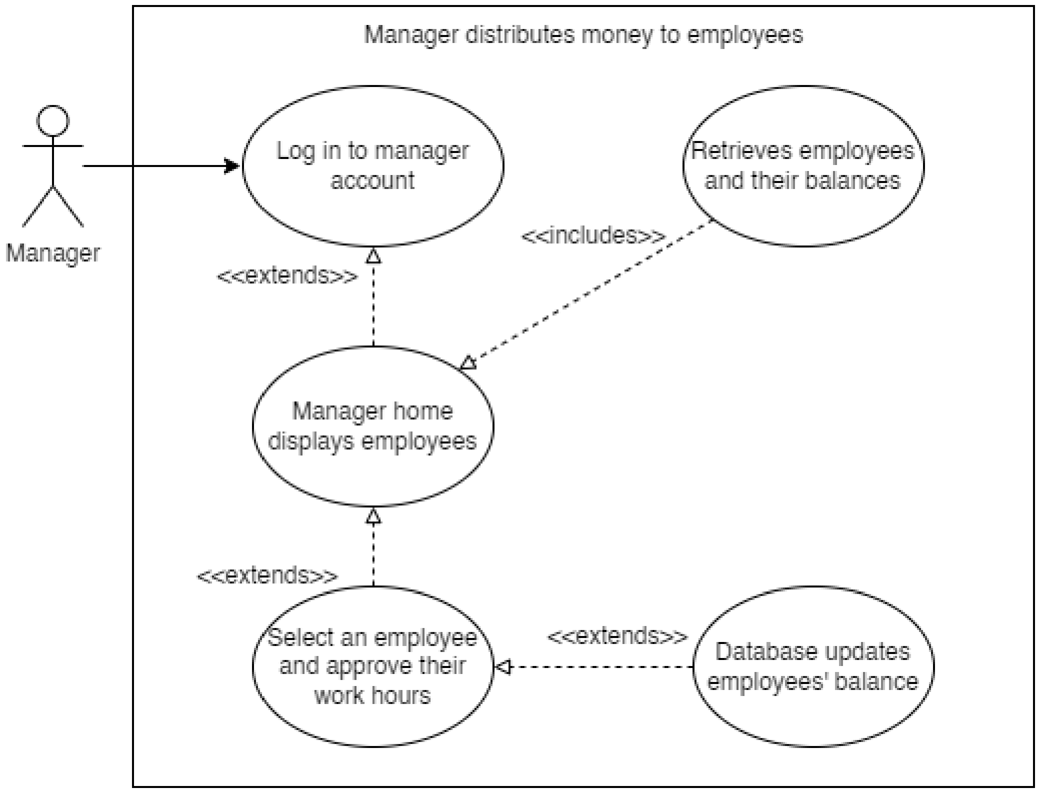
Exit Conditions:

- Successful cancellation
- Denied cancellation

Event Flow:

- User logs in, accesses user home
- They select an order to cancel
- If it isn't already finished, the order is removed from the system

Figure 5 - Manager distributes money


Manager distributes money to employees
Manager
Log in to manager account
Retrieves employees and their balances
<<includes>>
<<extends>>
Manager home displays employees
<<extends>>
Select an employee and approve their work hours
<<extends>>
Database updates employees' balance

Participating actor: Manager

Entry Conditions:

- Manager wants to pay employees for work hours

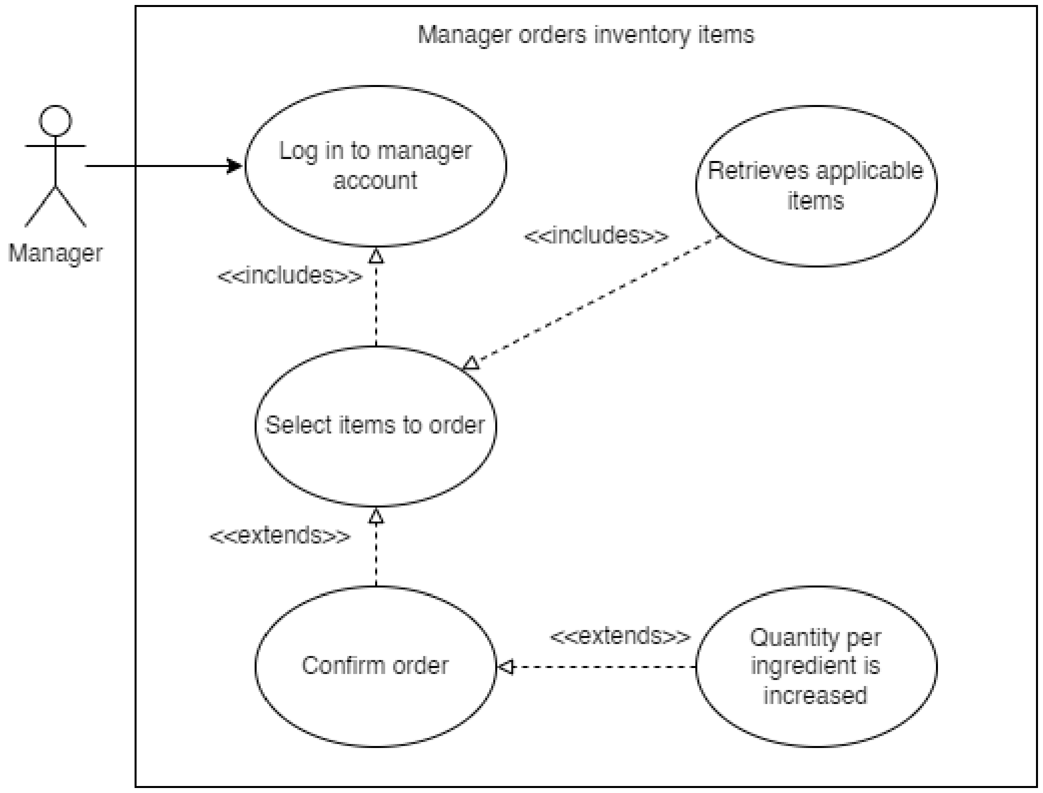
Exit Conditions:

- Employee balance is increased

Event Flow:

- Manger logs in
- They select an employee
- Approve work hours and they are paid based on their wage

Figure 6 - Manager orders items for inventory


Manager orders inventory items
Manager
Log in to manager account
Retrieves applicable items
<<includes>>
<<includes>>
Select items to order
<<extends>>
Confirm order
<<extends>>
Quantity per ingredient is increased

Participating actor: Manager

Entry Conditions:

- Manager needs to order items for store inventory

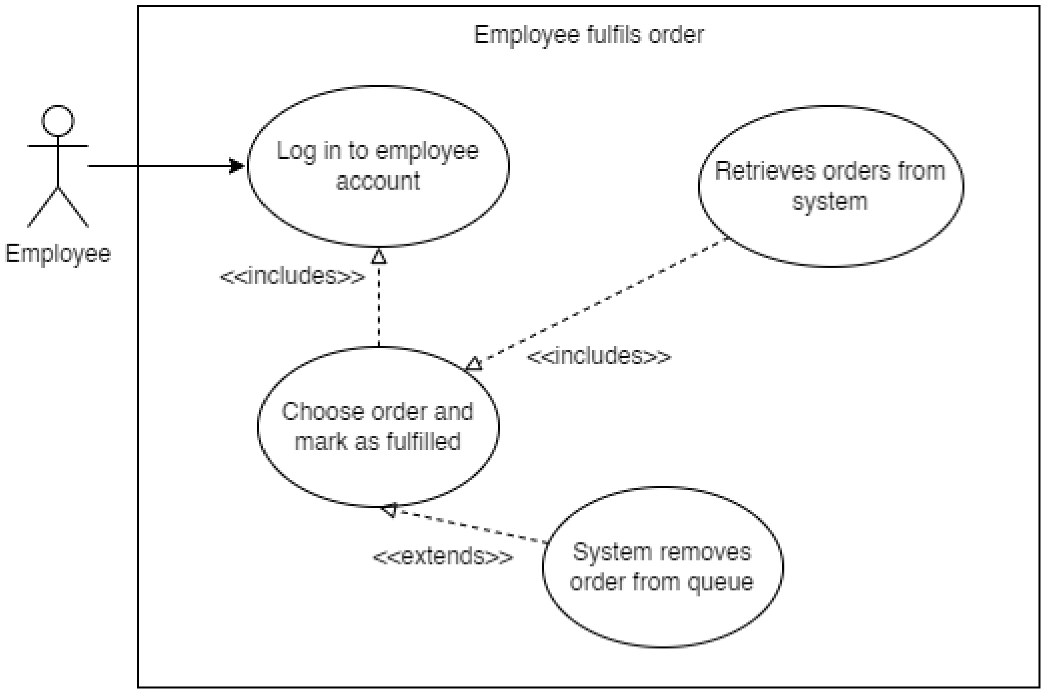
Exit Conditions:

- Items are successfully “ordered” (increase quantity in database)

Event Flow:

- Manger logs in
- They select quantity of items
- Confirm order, values of QuantityOnHand per ProductIngredient increase.

Figure 7 - Employee marks order as fulfilled


Employee fulfils order
Employee
Log in to employee account
Retrieves orders from system
<<includes>>
<<includes>>
Choose order and mark as fulfilled
System removes order from queue
<<extends>>

Participating actor: Employee

Entry Conditions:

- Employee needs to tell the system an order is complete

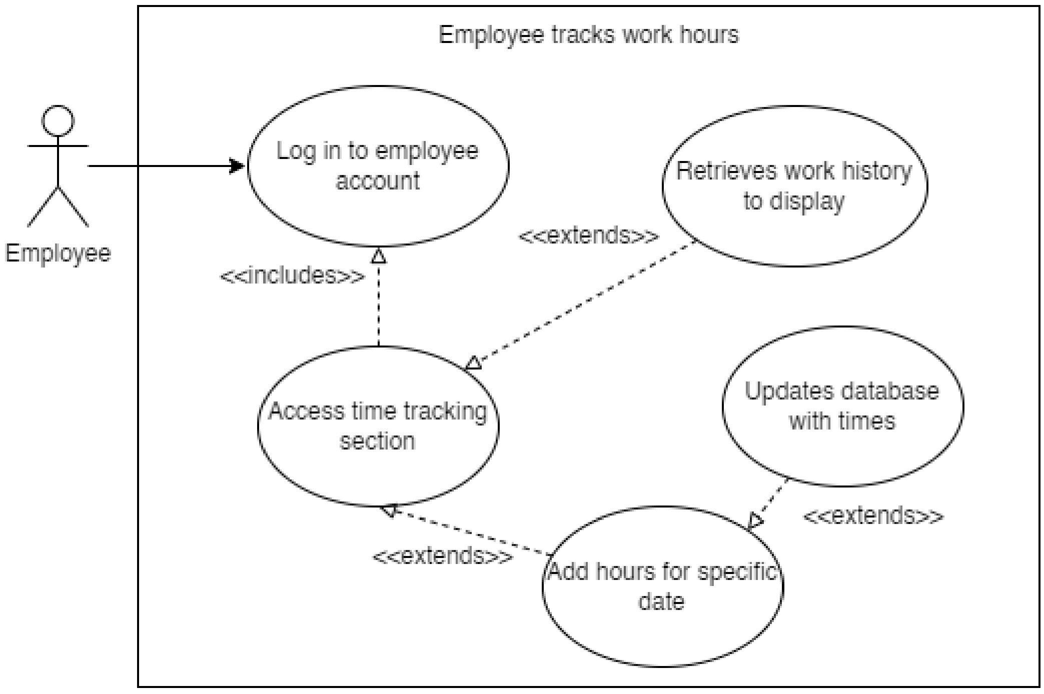
Exit Conditions:

- Order is marked as Fulfilled = true

Event Flow:

- Employee logs in
- They locate and select the completed order
- Mark it as fulfilled, database changes to reflect

Figure 8 - Employee tracks work hours


Employee tracks work hours
Employee
Log in to employee account
Retrieves work history to display
<<extends>>
<<includes>>
Access time tracking section
Updates database with times
<<extends>>
<<extends>>
Add hours for specific date

Participating actor: Employee

Entry Conditions:

- Employee needs to record their labor hours

Exit Conditions:

- Hours are added to system

Event Flow:

- Employee logs in
- They add times for a date
- Increase HoursWorked in database