This document was created with assistance from ChatGPT.

CodePop Low Level Design Document

Introduction

Welcome to the Low Level Design document for CodePop, a mobile app designed for soda shops. This document provides a working description of the product's system architecture, subsystems, classes, database tables, system performance concerns, potential security risks, user interface prototypes, programming languages, libraries, and frameworks that will be used, a deployment plan, and an explanation of how third-party systems will be integrated. This document is intended to keep developers, company management, and customers all on the same page.

Backlog development plan

Sprint breakdown

- Sprint 1 **(M)** 1 week
    - Must have features and basic functionality
    - Sprint 2 **(S)** 1 week
        - Should have features to enhance the application
        - Finish up remaining Must Have features not completed in previous sprint
    - Sprint 3 **(C)** 2 weeks
        - Could have features
        - Finish up any remaining tasks left unfinished in previous sprints.

Sprint Tasks Outline

Sprint 1:

Front-end Team:

- Create These pages:
    - sign up page
    - account page
    - home page
    - preferences page
    - drink creation page
    - cart page
    - payment page
    - Admin Dashboard
    - Manager Dashboard
- Special focus will be put on creating the nav bar that will persist through all pages for ease of navigation.
- Pages will be created with basic design layout with focus on functionality.

- There will be a focus on Accessibility such as adding tab controlled maneuvering and screen readers for all elements.
  Middle-End Team:
- Sign Up / Create Account:
    - Get user input and send to User datatable
        - Username
        - Password (**Security:** sha256 hash)
        - Email (**Security:** sha256 hash)
        - Role
        - Order history (empty)
        - Redirect to home page after
- Sign In:
    - Get user input and verify user existence
        - Email/username
        - Password
        - Error if user doesn't exist
        - Redirect to home page after
- Saved Drinks / Favorite Drinks
    - Add a drink from favorites to cart
        - Remove a drink from favorites
        - Edit drink object
            - Get original object info
            - Replace with whatever the user inputs
            - Saving or canceling redirects back to the favorites page
- Preferences
- Add flavor preference
- Remove flavor preference
- Drink Creation
- Create a drink object using user input
    - Checkbox to add to favorites, give an error if user exceeds “favorite” limit
- User can search for items (instant state change)
    - Get user input and compare with stock, send back everything that matches
- Automatically add to cart after creating
- Going back should not save the user input and redirects to previous page
- Randomly generated drinks

- When the user presses the randomize button:
    - Force the AI to follow a strict “Soda Object” format in its responses
- If user is logged in, send a message to AI with the user’s preference list
    - Or do a completely random one
- If user is not logged in, AI will create a random drink
- Convert AI output to a soda object to display to the user
- Error message if something went wrong creating the object (will take the place of the randomized soda image)
- Randomize re-clicked = resend the message
- Cart
- Remove drink from cart
- Edit drink in cart
    - Get original object info
        - Replace with whatever the user inputs
        - Option to add drink to favorites
        - Saving or canceling redirects back to the cart
- Checkout redirects to purchase page
- Purchase/Checkout
- Get user’s input payment information
- Send info to stripe, bringing back an error or success message
- If user wants to, save payment information (**Security:** sha256 hash?)
- Redirects to a page with an “I’m on my way!” button. Order starts creation once user clicks.
- Once order is finished being made, user is given a randomly generated code to open the cooler with their order
- Admin Dashboard
    - See list of all active user accounts (instant state change)
    - View user account & info
        - Delete a user account (**Security**: “Are you sure?” pop-up)
- Promote to manager (change role from user -> manager role) (**Security**: “Are you sure?” pop-up)
    - Back button redirects to dashboard
- Manager Dash

- Pull up revenue, inventory counts, and total # of users (instant state change)
- Could use an API to show revenue and inventory counts as graphs **(need to research)**

  Back-End-Team

- Create User Table
- Create Notification Table
- Create Preferences Table
- Create Order Table
- Create Payment Table
- Create Revenue Table
- Create Drink Table
- Create Inventory table
- Create access functions for each data table to pass information from the database to the middleware. (These are outlined in the backend UML)

  Sprint 2:

  Front-End Team:

- Design for all pages will be finalized with added graphics
- Create confirmation page with additional features:
  - Rating availability
- Additional timing options will be added to checkout page
- Accessibility will be fully tested and checked according to WCAG standards
  Middle-End Team:
- Sign Up / Create Account:
- Check user input
  - Valid email (@gmail, @yahoo, @, etc.)
  - Secure password (this length, these symbols, etc.)
  - Username not already in use
  - Send error otherwise
- Confirm user sign up with email (ensure user typed in the correct email)
  - User email -> django send_email() – **(research)**
    - Write up email topic and body
  - Send error if not valid email and redirect to sign up page

- Pre-set Menu
- Add menu drink to favorites
- Edit pre-set drink object
    - Get original object info
        - Add drink to cart with whatever the user inputs
        - Option to add drink to favorites
        - Adding to cart or canceling redirects back to the menu (user input should not change the pre-set objects)
- Add pre-set drink to cart
- Ratings
- Stars: Each star = a specific %
- Add to the drink object's % for each rating submission
- Keep track of the top 3-5 drinks after each rating submission (if >3-5 have equal ratings base it off of most recent submissions – submitting a rating will put that drink at the top of its "rating group" – ex. If there are 6 5-stars and one of them gets a new submission, it becomes the #1 rated drink)
- User's should be able to edit drink ratings
- Preferences (Geolocation)
    - Geolocation checkbox: enable if checked, otherwise disable
- Payment/Checkout (Geolocation)
    - User has option to pick between the "On My Way" button and Geolocation
    - Send helpful message if user tries to pick geolocation, but it is unchecked in preferences

      Using Geolocation:
    - Track the user up until they are within 500-1000m of a location
    - Once within range, order creation starts (display a timer on screen)
        - Once order is finished being made, user is given a randomly generated code to open the cooler with their order
- Confirmation Page
- Redirected from Payment/Checkout
- Displays the timer
- Option to refund order (subtract revenue)
- Option to rate each drink in the order (see ratings section)

Back-End-Team

- Populate Database with inventory (sodas, syrups, etc…)
- Stress test database to ensure performance with a large amount of data.

Sprint 3:

Front-End Team:

- Create complaints page and add nav button to page from the confirmation page
- Additional pickup time options added to checkout page
- Manual testing to ensure all functionality of navigation and buttons
  Middle-End Team:
- Complaints
- Send user input to AI and display its response (**Security:** remove any suspicious symbols, replace suspicious words with secure synonyms)
- Confirmation page
  - Option that redirects to complaints page
- Purchase/Checkout
  - Option to pick pickup-time (along with “On My Way” button and geolocation)

    Using pickup-time:
  - Calculate timePickUp - timeToMakeOrder = timeToStartOrder
  - Start the timer and show on screen (Confirmation page) when the (device) clock hits timeToStartOrder
    - Once order is finished being made, user is given a randomly generated code to open the cooler with their order

All Tasks Outline

Redirects

- Incorporate loading screen between redirects
- There will always be some “Home” button that can redirect back to the home page

  Home
- Interactable buttons that redirect to their specific page

  Sign Up / Create Account
- Check user input **(S)**

    - Valid email (@gmail, @yahoo, @, etc.)
    - Secure password (this length, these symbols, etc.)
    - Username not already in use
    - Send error if issues

- Confirm user sign up with email (ensure user typed in the correct email) **(S)**

    - User email -> send_email
        - Write up email topic and body
    - Send error if not valid email (redirect back to sign up page)

- Get user input and send to User datatable **(M)**

    - Username
    - Password (**Security:** sha256 hash)
    - Email (**Security:** sha256 hash)
    - Role
    - Order history (empty)

  Sign in

- Get user input and verify user existence **(M)**

    - Email/username
    - Password
    - Error if user doesn't exist

  Sign In / Sign up - Redirect to home page

  Complaints **(C)**

- Send user input to AI and display its response
- **Security:** remove any suspicious symbols, replace suspicious words with secure synonyms
- Maybe have a file with all the words and compare with that? Or an array/dictionary

  Saved Drinks / Favorites **(M)**

- Add drink to cart
- Remove drink from favorites
- Edit drink object (save and back button, back redirects to favorites page)

  Preferences **(M)**

- Add preference (flavors)
- Remove preference (flavors)
- enable/disable geolocation checkbox

  Pre-set Menu **(S)**

- Add to favorites
- Edit drink object (save and back button, back redirects to menu page)
- Add to cart

  Drink Creation **(M)**

- Create a drink object using user input
  - Add to favorites option (error if too many)
- User search pulls up related items (instant input state change)
- Automatically add to cart after saving (should just be an “add to cart” or “back” button
- Going back should not save the user input and redirects to previous page

  Randomly generated **(M)**

- If user log in, send a message to AI with the user’s preference list
  - Or do a completely random one
- If not log in, send a message to AI with ingredients of the top 3-5 rated drinks
  - Or do a completely random one
- Bring back AI output to display (should become a soda object: Add to cart or edit – force the AI to follow a strict format so we can parse it’s response into an object)
- Error message if something went wrong creating the object (will take the place of the randomized soda image)
- Rerandomize clicked = resend the message

  Cart **(M)**

- Remove from cart
- Edit + add to favorites
- Confirm redirects to purchase

  Purchase/Checkout **(M)**

- Enter payment information and send to stripe, bringing back an error or success message
- Pickup options (only one):
  - Finish at time (use pickup time - creation time = start time)
  - Start when button clicked
  - geolocation
- Save payment (**Security:** sha256 hash?)

Geolocation / Start Order Creation

- If Geolocation (MapBox) **(S)**
    - Track the user up until they are within 500-1000m (500 = TSC to Engineering - 1000 = bottom of Old Main hill to FAV stoplight) of a location
    - Start the timer for the order and show on screen (order is being created)
    - Timer ends, say order is ready (give cooler-code)
- If specific time set **(C)**
    - Calculate timePickUp - timeToMakeOrder = timeToStartOrder
    - Start the timer and show on screen when the (device) clock hits timeToStartOrder (order is being created)
    - Timer ends, say order is ready (give cooler-code)
- If start button **(M)**
    - Note: Button should say “Start My Order!” or “On My Way”, not “I’m here”
    - User should be given info on how long their order will take
    - When button gets clicked start the timer and show it on screen (order is being created)
    - Timer ends, say order is ready (give cooler-code)

Confirmation **(S)**

- Timer (time remaining for order creation)
- Option to redirect to complaints page
- Option to refund
- Rate each drink in order

Ratings **(S)**

- Stars: Each star = a specific %
- add to the drink’s % for each rating submission
- Keep track of the top 3-5 drinks after each rating submission (if >3-5 have equal ratings base it off of most recent submissions – submitting a rating will put that drink at the top of its “rating group” – ex. If there are 6 5-stars and one of them gets a new submission, it becomes the #1 rated drink)

Admin Dash **(M)**

- Delete a user account (**Security**: have an “Are you sure?” pop-up)
- Promote (change role from user -> manager role) (**Security**: have an “Are you sure?” pop-up)
- View user account & info (back button takes back to admin dash)
- (this may all be instant state change)

Manager Dash **(M)**

- Pull up revenue, inventory counts, and total users (instant state change?)
- Use API to show graphs?

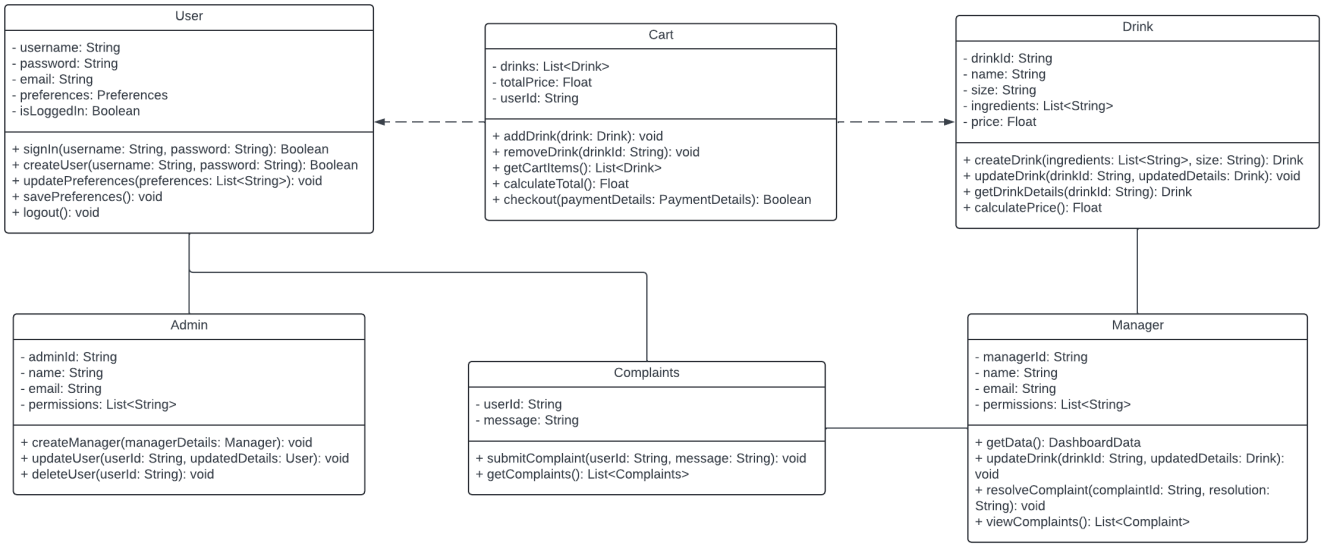
System architecture

Client-server architecture

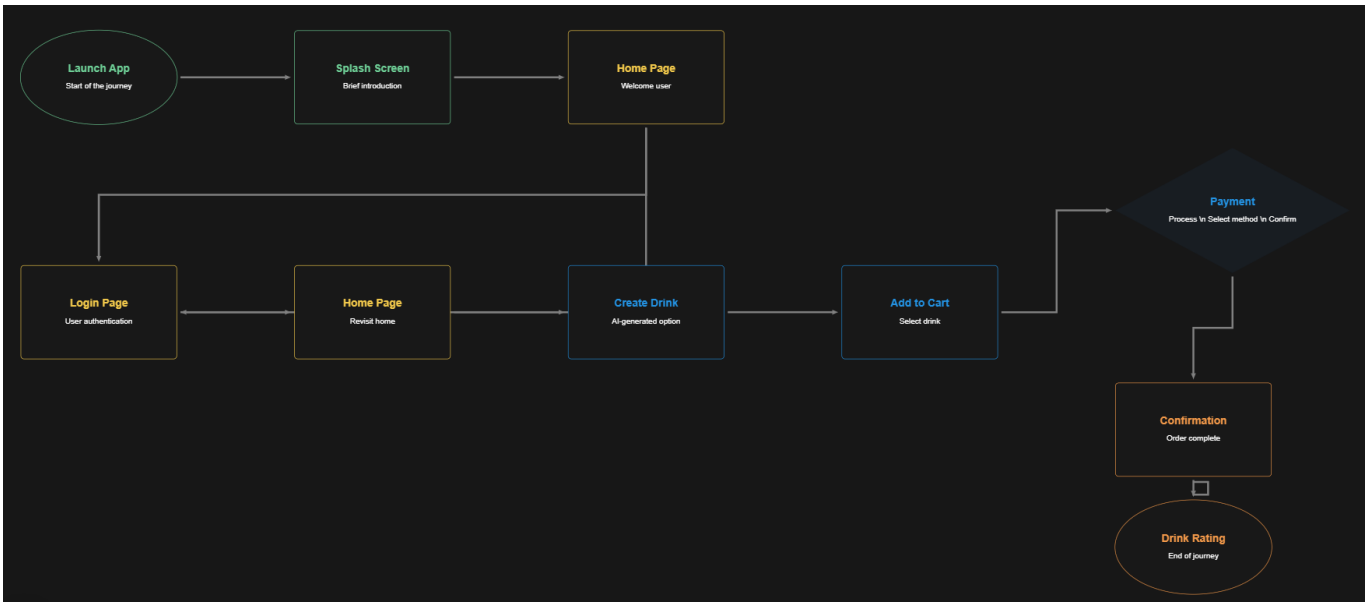
The CodePop system architecture follows a three-tier model, consisting of the front-end, middle-end, and back-end layers. The front-end, built with React Native, handles user interaction and interfaces, communicating with the middle-end via API requests. The middle-end is powered by Django, acting as the central framework that manages business logic, authentication, and data flow. It interacts with the PostgreSQL database on the back-end, which stores user data, drink configurations, and order information. APIs like Stripe (for payments), MapBox (for geolocation), and AI-based models (for drink generation and complaints chatbot) are integrated to enhance functionality and user experience, ensuring seamless interaction between the client and server.

Subsystems and UML Class Diagrams

- App objects:



- Main user flow:



User interfaces:

Accessibility

The CodePop interface is carefully designed to support both usability and accessibility, adhering to the Web Content Accessibility Guidelines (WCAG). A color palette has been selected based on an analysis of common forms of color blindness, ensuring that problematic combinations like teal and purple are not placed next to each other for better readability. Each page is fully compatible with screen readers, offering audio feedback for users with visual impairments. Additionally, tab-controlled navigation is integrated, allowing keyboard users to easily move through the interface without relying on a mouse, ensuring a more inclusive user experience.

Flow and design for page layout

Home Page

- Nav Bar: Links to Drink Design, Account User Home, Cart, Complaints.
- Seasonal Drinks Menu: Displayed as a carousel.
- Generate Random Drink Button (AI-powered).
- Set Drink Text Box (input for AI API-generated drink).
- Create Account Button (for non-account users).
- **Primary Flow**: User either navigates through nav bar or signs in/creates an account.

Sign-In Page

- Username/Password text boxes.
- Login Button: Authenticates user via `signIn()`, redirects to Home Page upon success.
- Error Message displayed automatically if login fails.
- Link to Sign-Up Page for new users.

Sign-Up Page

- Fields for account creation.
- Button to create an account via `createUser()`, redirect to Sign-In after successful creation.

Account User Home Page

- Displays saved drinks and current user preferences.
- Buttons to modify preferences (`updatePreferences()`) and save changes (`savePreferences()`).
- Option to enable/disable geolocation.

Complaints Page

- Text entry box for complaints, integrated with AI API to manage complaints.

Drink Design Page

- Drink creation UI with add-ins displayed as graphics.
- Interactive drink customization (soda, size, ice, syrups, etc.).
- Search bar for quick ingredient lookup.
- Drink object is created (`createDrink()`) and saved to the cart.

Cart Page

- Displays a list of drinks with options to remove, edit, or proceed to checkout.
- Each drink object shows price and ingredients.
- The Checkout button redirects to the Payment Page.
- User can edit existing drinks using `updateDrink()`

Payment Page

- Stripe API for secure payment processing.
- User can choose between geolocation-based tracking or a scheduled pickup time.
- Confirmation message displayed after payment submission.

Confirmation Page

- Displays drink details with a timer countdown.
- Option to rate each drink and file a complaint if needed.
- Refund button if necessary.

Manager Dashboard

- Data visualization (charts) of store data like revenue and expenses.
- Managers can access this dashboard to view key metrics via `getData()`.

Admin Dashboard

- Admin controls for managing user accounts (update, delete, create manager accounts).
- Data visualization for user and store statistics.

Database tables

User Table

Field Data	Data Type	Constraints

Field Data	Data Type	Constraints
UserID	int	Primary Key
Username	String	
Password	String	
Email	String (Email)	
UserRole	String	Default “Customer”
FavoriteDrinks	List of Drink IDs	Default “Empty List”

Preference Table

Field Data	Data Type	Constraints
PreferenceID	int	Primary Key
UserID	int	Foreign Key References User(UserID)
Preference	bool	Default “no preferences checked”

Order Table

Field Data	Data Type	Constraints
OrderID	int	Primary Key
UserID	int	Foreign Key References User(UserID)
DrinkDetails	List drink objects	
OrderStatus	String	
PaymentStatus	String	
PickupTime	Timestamp	
CreationTime	Timestamp	

Drink Table

Field Data	Data Type	Constraints
DrinkID	int	Primary Key
SyrupsUsed	List[Strings]	
SodaUsed	List[Strings]	
AddIns	List[Strings]	
Rating	double	Default NULL
Price	double	

Inventory Table

Field Data	Data Type	Constraints
InventoryID	int	Primary Key
ItemName	String	
ItemType	String	Must be “Soda”, “Syrup”, “Add In” or “Physical”(for cups lids straws etc…)
Quantity	int	
ThresholdLevel	int	
LastUpdated	Timestamp	

Payment Table

Field Data	Data Type	Constraints
PaymentID	int	Primary Key
OrderID	int	Foreign Key References Order(OrderID)
UserID	Int	Foreign Key References User(UserID)
Amount	double	
PaymentMethod	String	
PaymentStatus	String	
RefundStatus	String	

Notification Table

Field Data	Data Type	Constraints
NotificationID	int	Primary Key
UserID	int	Foreign Key References User(UserID)
Message	String	
Timestamp	timestamp	
Type	String	

Code Table

Field Data	Data Type	Constraints
OrderID	int	Foreign Key References User(UserID)
ExpirationTime	Timestamp	

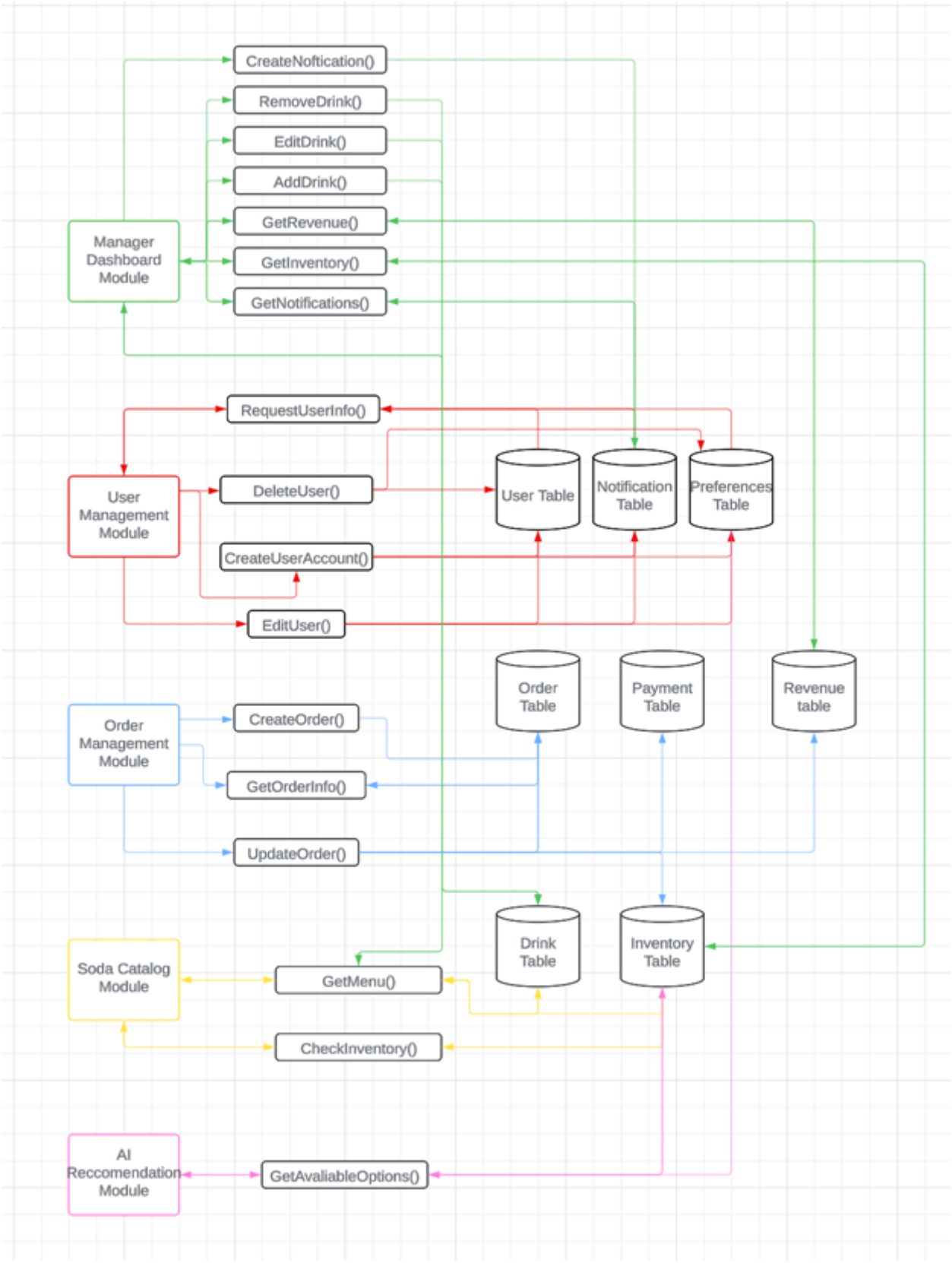
Revenue Table

Field Data	Data Type	Constraints
RevenueID	int	Primary Key
TotalAmount	double	
Date	date	
UserID	int	Foreign Key References User(UserID)

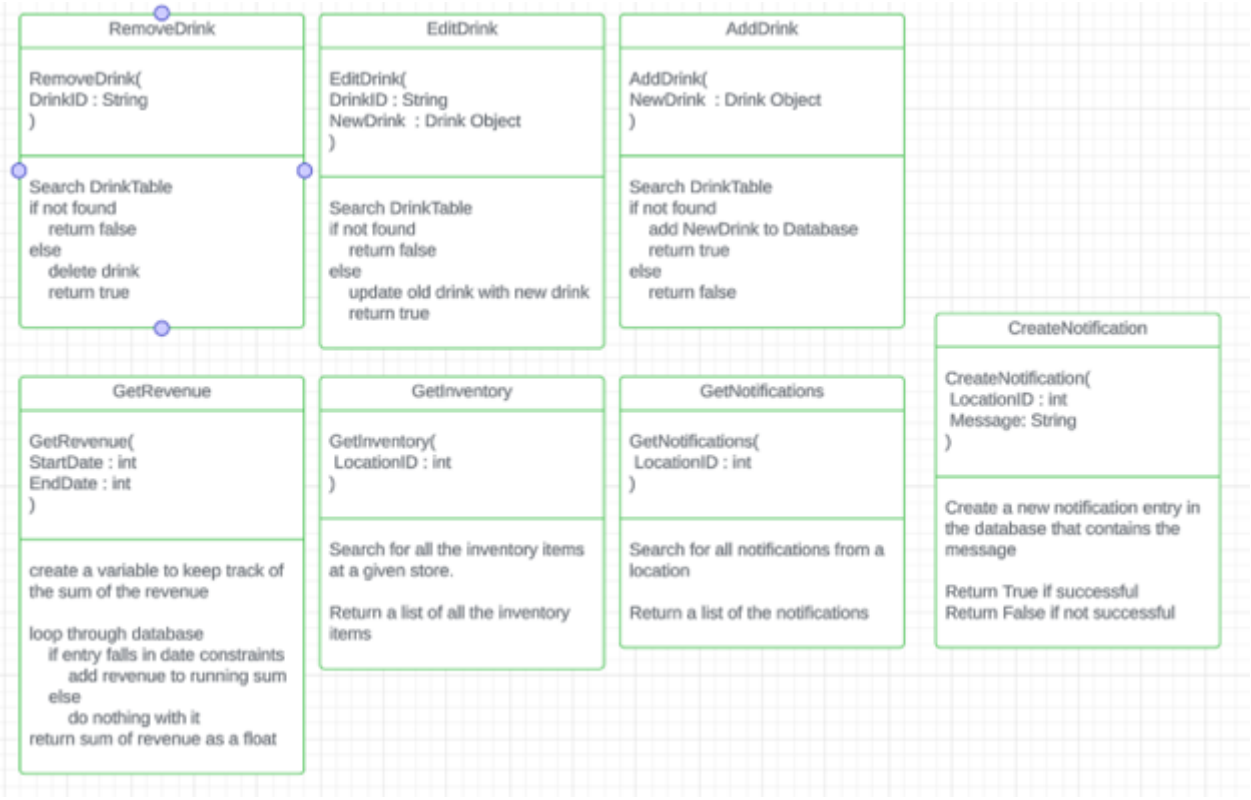
Inventory(Items that must be in the database)

Sodas	Syrups	Add ins
Mtn. Dew Diet Mtn. Dew Dr. Pepper Diet Dr. Pepper Dr. Pepper Zero Dr Pepper Cream Soda Sprite Sprite Zero Coke Diet Coke Coke Zero Pepsi Diet Pepsi Rootbeer Fanta Big Red Powerade Lemonade Light Lemonade	Coconut Pineapple Strawberry Raspberry Blackberry Blue Curacao Passion Fruit Vanilla Pomegranate Peach Grapefruit Green Apple Pear Cherry Cupcake Orange Blood Orange Mango Cranberry Blue Raspberry Grape Sour Kiwi Chocolate Milano Huckleberry Sweetened Lime Mojito Lemon Lime Cinnamon Watermelon Guava Banana Lavender Cucumber Salted Caramel Choc Chip Cookie Dough Brown Sugar Cinnamon Hazelnut Pumpkin Spice Peppermint Irish Cream Gingerbread White Chocolate Butterscotch Bubble Gum Cotton Candy Butterbrew Mix	Cream Coconut Cream Whip Lemon Wedge Lime Wedge French Vanilla Creamer Candy Sprinkles Strawberry Puree Peach Puree Mango Puree Raspberry Puree Ice??????

Backend UML:



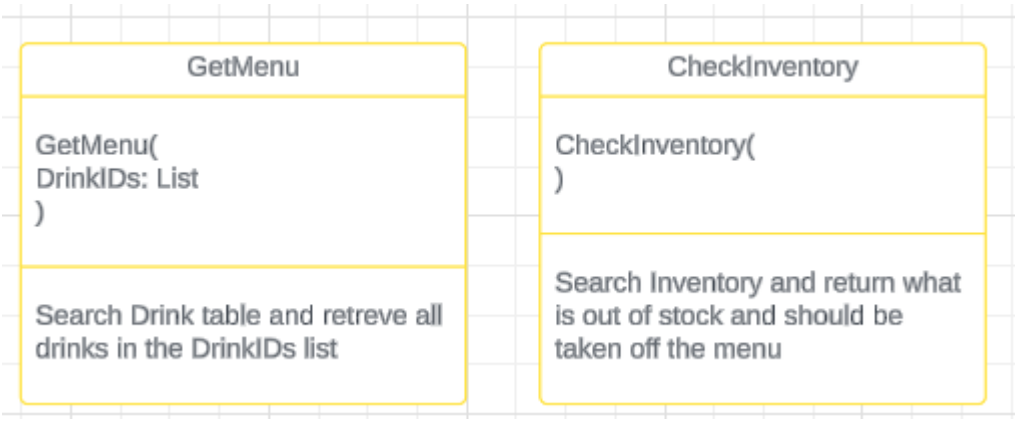
Manager Dashboard Module Functions:



User Management Module Functions:

Order Management Module Functions:

Soda Catalog Module Functions:



AI Recommendation Module Functions:

System performance

To address system performance, potential bottlenecks such as high traffic during peak hours, concurrent payment processing, and large database queries are considered. The system is designed with scalability in mind, using load balancing techniques to distribute traffic across multiple server instances. The use of a CDN for static assets ensures faster content delivery to users across different regions. For the backend, Django's ORM optimizes database queries, and indexing will be employed for frequently accessed data like user preferences and drink details to improve response times. PostgreSQL will handle database connections efficiently, but if demand grows, database read replicas will be deployed to reduce the load on the primary database. Additionally, autoscaling for cloud infrastructure ensures that more server instances can be automatically provisioned during traffic spikes. This architecture allows the system to handle increases in load while maintaining optimal performance.

Security risks

The CodePop app incorporates a robust security model based on Django's built-in authentication and authorization system. This system manages user accounts, groups, permissions, and cookie-based sessions, ensuring secure access across different user roles. Admins have the authority to manage user accounts, add or remove users, and create manager accounts, while managers have access to store-specific data like revenue and expense reports. To enhance security, features such as password strength checking can be implemented. The app separates the client and server, utilizing token-based authentication for secure communication. Django's security features include query parameterization for injection protection and Cross-Site Request Forgery (CSRF) protection to prevent unauthorized actions. Additionally, sensitive data, including user payment information, email addresses, and store revenue reports, will be encrypted using SHA-256 both at rest and in transit. CodePop complies with relevant data protection laws such as GDPR and takes measures to address the OWASP Top 10 security risks. Users will be given the option to opt into features that handle personal data, ensuring transparency and privacy.

- **Authentication and Authorization**: Description of user roles and permission management.
    - Explanation of admin and manager access and roles:
        - Admins have access to user account information as well as permissions to add/remove general user accounts and create manager accounts.
        - Managers have access to store data such as revenue and expense reports.
    - Django comes with a built in user authentication system that handles user accounts, groups, permissions and cookie-based user sessions
        - This system can be expanded and customized to add things like
        - password strength checking to add more security.
    - To secure the application, the client and server will be separated.
        - The client and server will talk to each other through token authentication which is already included with Django.
    - Django security features: https://docs.djangoproject.com/en/5.1/topics/security/
        - Includes injection protection because queries are constructed using query parameterization
        - Includes Cross site request forgery (CSRF) protection which prevents attacks that perform actions using other people's credentials.
- **Data Encryption**: Explanation of how data will be encrypted (at rest and in transit).
    - Django user data encryption
    - Sha 256 encryption
- **Compliance**: Relevant data protection laws (GDPR, HIPAA).
    - Pay attention to the OWASP top 10: https://owasp.org/www-project-top-ten/
- **Sensitive data**
    - User data:
        - Payment information
        - Email
        - geolocation
    - Store data:
        - Revenue reports
- **Privacy**
    - We will make sure that the user has the option to opt into any of the features that handle personal data (geolocation, drink preferences, emails) to ensure that they are able to make an

informed choice about their data.

Programming languages, libraries, frameworks, and third party systems

Front-End

- Framework: React Native
- Languages: JavaScript, HTML, CSS

Middle-End

- Framework: Django
- Languages: Python

Back-End

- Framework: Django
- Database: PostgreSQL
- Languages: Python, SQL

APIs & External Services

- Payment System: Stripe API
    - Set up stripe account and API keys
    - Integrate Stripe's Mobile SDK for the frontend (checkout screen)
    - Create payment intents on backend
    - Handle payment confirmation on the frontend
        - Pass the `client_secret` to frontend
        - Use the Stripe SDK to confirm the payment by calling `confirmPayment`
        - Set Up Webhooks for Payment Status Tracking (notifies backend if payment was successful, failed, etc.)
- Geolocation: MapBox API
    - Create mapBox account and obtain access token
    - Integrate MapBox SDK into frontend
    - Use the *Geolocation API* or Mapbox's *GeolocateControl* to get the user's current coordinates.
    - Set up proximity alerts
- Random Drink Generation: Content-Based AI Filtering Model (Scikit-Learn)
    - Use a pandas DataFrame to prep data
    - Convert features into numerical values
    - Define a similarity metric for the drinks (probably `cosine_similarity`)
    - Build recommendation function
    - Integrate model into backend

- Complaints Chatbot: DialoGPT (Hugging Face)
    - Set up an environment and load DialoGPT model
    - Create a function to generate responses
    - Set up an API endpoint (so that the app's frontend can call it)
    - Integrate into frontend
- Loading Screens: Gemini AI Images
    - Preload 1-2 images we like
    - Embed the preloaded images into the app

Deployment plan

1. Development Environment Setup

- **Version Control**: Use Git for managing source code. Establish a remote repository (e.g., GitHub, GitLab) for collaboration.
- **Local Development**: Set up local environments for developers with tools like Docker for containerization to ensure consistency.
- **Technology Stack**:
    - Frontend: HTML, CSS, JavaScript, and Reach Native framework
    - Backend: Django for server-side logic and database management.
    - Database: PostgreSQL for managing user data, drinks, and store information.
- **Dependencies**: Use `pip` for Python dependencies and `npm` for any frontend packages.

2. Staging Environment

- **Staging Server Setup**: Deploy a staging environment to test all new features before going live.
- **Testing**:
    - **Unit Testing**: For each individual function like `createUser()`, `getDrink()`, etc.
    - **Integration Testing**: Ensure all components (frontend, backend, database) work together.
    - **Accessibility Testing**: Verify compliance with WCAG, ensuring color contrast, screen reader compatibility, and tab navigation.
    - **Load Testing**: Use tools like Apache JMeter to simulate heavy traffic and ensure scalability.
- **Pre-Deployment Validation**: Test payment flows with the Stripe API, and ensure all security measures (token authentication, CSRF protection) are in place.

3. User Acceptance Testing (UAT)

- Invite key stakeholders (store owners, staff) to use the staging environment and provide feedback.
- Ensure the interface is intuitive, accessible, and aligns with business needs.
- Adjust any features or fix bugs found during UAT before final production deployment.

4. Production Environment Setup

- **Hosting**:
    - **Frontend Hosting**: Deploy frontend assets to a Content Delivery Network (CDN) for fast global access (e.g., AWS S3 or Netlify).

    - **Backend Hosting**: Use cloud-based services like AWS EC2 or DigitalOcean for running the Django app.
- **Database Setup**: Deploy the PostgreSQL database using cloud-based solutions (e.g., AWS RDS or Heroku Postgres) for scalability and backup.
- **Security Setup**:
    - **SSL/TLS Certificates**: Ensure HTTPS is enabled for secure communication.
    - **Environment Variables**: Store sensitive data like API keys and database credentials securely (e.g., AWS Secrets Manager).
    - **Firewall & Access Control**: Set up firewalls and limit access to production servers using SSH keys.
    - **Backup Plan**: Regular backups of databases and critical assets to prevent data loss.

5. Production Deployment

- **Code Freeze**: Set a code freeze date before deployment to ensure no last-minute changes.
- **Final Deployment Steps**:
    - Deploy the frontend and backend to their respective hosting services.
    - Migrate any production databases and ensure all data from staging is migrated.
    - Ensure that token-based authentication and secure data encryption (e.g., SHA-256) are fully operational.
- **Post-Deployment Validation**:
    - Test all critical functions (sign-in, drink creation, cart, payment).
    - Check that the admin and manager dashboards are accessible with the correct permissions.

6. Post-Deployment Monitoring & Maintenance

- **Ongoing Monitoring**: Ensure performance metrics are actively tracked, especially during high traffic periods.
- **Bug Fixes**: Implement a process for quickly addressing user-reported issues or security vulnerabilities.
- **Regular Updates**: Schedule periodic security patches, updates to libraries, and infrastructure improvements.

7. Scaling & Future Enhancements

- **Auto-Scaling**: Set up auto-scaling for cloud-based infrastructure to handle growing user demand.
- **Feature Rollouts**: Plan for iterative feature deployment, such as enhanced AI drink recommendations or more detailed analytics for managers and admins.
- **Performance Optimization**: Regularly review the system's performance and optimize database queries, asset loading times, and server response times.