This document was created with assistance from ChatGPT.

Code Pop High-Level Design Document

Introduction

- Purpose
- Scope
- Audience
  System Overview
- Problem Statement
- Proposed Solution
- Hardware Platform
    - Mobile
    - Laptop and Desktop

  Architecture Design

- Architecture Overview: Explanation of the overall architecture (monolithic, microservices, etc.)
- Component Diagram: Diagram showing major system components and their relationships
- Technology Stack: Technologies and frameworks used (e.g., languages, databases, servers)
    - React Native
    - Django
    - PostgreSQL
    - AI

  Modules and Components (Internal Interfaces)

- Module Overview: Description of key modules or components, their responsibilities, and interactions
- Data Flow Diagram (DFD): Illustration of how data moves between components
- Component Interaction: Details on how system components will communicate (e.g., APIs, web services)
  Data Design
- Data Model: High-level structure of data, including key entities and relationships
- Database Design: Type of database used (relational, NoSQL, etc.), major tables, and relationships
- Data Access Layer: Overview of how data is accessed, stored, and retrieved (e.g., ORM, SQL)
  Integration Points (External Interfaces)
- External Systems: Description of external systems or services the app will integrate with

- APIs: List of public/external APIs, endpoints, methods, and data contracts
    - Payment System: Stripe
    - Geolocator: MapBox
    - AI Chatbot: DialoGPT
    - Notifications: Firebase Cloud Messaging (FCM)

User Interface (UI) Design Overview

- UI/UX Principles: High-level UI/UX principles (e.g., responsiveness, accessibility)
- Mockups: High-level mockups or wireframes of key screens
- Navigation Flow: Overview of how users will navigate the app
Input and Output (I/O)
- Input
- Output
Security and Privacy
- Authentication and Authorization: Description of user roles and permission management
- Data Encryption: Explanation of how data will be encrypted (at rest and in transit)
- Compliance: Relevant data protection laws (GDPR, HIPAA)
- Privacy
Testing Strategy
- Unit Testing
- Manual Testing
Risks and Mitigations
- Identified Risks: List of known risks (e.g., technology choice, dependencies)
- Mitigation Plans: Strategies for addressing these risks
    - User Geolocation
    - User Input (AI)
    - Payment Information
    - Allergies
    - User (Account) Information
    - Location Revenue Information
    - Legal Issues

---

1. Introduction

Purpose: This document exists to provide a reference for developers while working on the CodePop app to ensure that the development team can work independent of each other and still have code that will work together to form the final project at the end.

Scope: This document has a large scope that encompasses just about every part of development, but it is more focused on the "why" of each design choice than the "how". As such it won't delve too deeply into the specific implementation detail.

Audience: This document is meant for developers and stakeholders of the project to ensure development is going in the right direction and everyone is on the same page.

2. System Overview

Problem Statement: In the world of dirty soda shops, there are too many options and many long lines, resulting in a confusing and overwhelming customer experience.

Proposed Solution: CodePop will provide a simple, AI-powered ordering experience to help eliminate the confusion and pressure typically associated with dirty soda shops.

Hardware Platform:
CodePop is designed to be accessible to a wide variety of users. Our goal is to create software that is easy/quick to use. This section outlines the hardware platforms CodePop could be built on, including priorities and possibilities.

- **Mobile**:
  - **App**:
    CodePop's priority hardware will be a mobile application. Phones are generally very easy to use and carry around. Since users will need to travel to a CodePop location to pick up their drinks, having a device they can easily bring with them is essential. Touchscreens make it easy for users to navigate the app quickly.
    - A mobile app is more prioritized than a website because we believe it best fits the client's needs.
  - **Website**:
    The mobile app can be converted to a mobile-optimized website with the same functionality and layout as the app. To ensure accessibility, the app and website will be designed to work on both Android and iOS devices. However, due to easier testing methods, we will begin by developing the app for Android.
  - **Touchscreen**:
    Since touchscreen functionality is key to accessibility and usability, it will be prioritized in the app's UI. Buttons and sections will be larger in size to make them easier to tap without zooming in. Other actions, such as swiping and holding, will also be considered.
  - **Gestures**:
    Gestures will not be included in the first version of the app. They are less reliable than touchscreen interactions, and our focus will remain on perfecting the core features of the app instead.
  - **Portrait vs. Landscape**:
    The app/website will be optimized for portrait mode to allow easy access to all points of the screen and to enable comfortable use with one hand. Landscape mode may be considered in future versions or when laptop/desktop accessibility is introduced.
- **Laptop/Desktop**:
  - **Website**:
    While phones are the primary use-case for the CodePop app/website, a laptop/desktop UI will

not be a high priority initially. A desktop-friendly UI may be added after the mobile functionality is complete, provided it doesn't divert resources from more critical features. Laptop/desktop access will be limited to the website only, not an app, to avoid over-scoping.

- **Touchscreen Laptops**:
  Although touchscreen laptops exist, their dimensions differ significantly from mobile devices, and they will not be prioritized in the initial development phase. Their prioritization will remain with every other laptop device.

3. Architecture Design

- **Architecture Overview**:
  - This will be a client-server architecture where the app run on the phone will be a client that will communicate and get information from our server and then display that information for the user.
- **Component Diagram**: Diagram showing major system components and their relationships.
- **Technology Stack**:
  Technologies and frameworks used (e.g., languages, databases, servers).
  - **React Native (Frontend):**
    - React native is a very popular front end development framework that is widely supported by most API's and backend frameworks.
    - React is based on javascript which offers the best integration and flexibility for a lightweight client-server integration.
    - React's hot reload feature offers much more flexibility to front end developers to see changes in real time, which leads to faster, more flexible front end development.
  - **Django(Backend):**
    - Django comes preloaded with user authentication and security built in which should help us save time in not having to develop those parts of the software.
    - Django also is widely supported by most tools because of its popularity.
  - **PostgreSQL(Database):**
    - PostgreSQL is well suited for databases that have complex relationships.
    - It is one of the most popular databases to be used with Django meaning there is a lot of community support and help for PostgreSQL.
    - PostgreSQL is very scalable meaning it will function well with both large and small amounts of data
  - **Artificial Intelligence (AI)**:
    - **Scikit-Learn** will be the AI library we use for both of the AI Models listed below (Content-Based Model & Item-Based Collaborative Filtering Model). We have chosen this library for the following reasons:
      - Is free
      - Is protected under the BSD license, which allows for free usage in personal and commercial projects
      - Easy to implement
      - Works well for small datasets
      - Is widely used in academia and industry, providing strong documentation for any of our future needs
      - Uses Python
    - **Content-Based AI Filtering Model by Scikit-Learn** will be used for the personalized drink suggestion for account users. We have decided this model for the following reasons:

        - **Does not require large amounts of user data**:
          As a startup, we will not have significant user data to rely on initially. This model allows for an enhanced customer experience from the start without the need for extensive data.
        - **Allows for cold-starts for new users**:
          Since the recommendations are item-based, the AI does not depend on other users for suggestions. This provides personalized recommendations to new users even before we gather large-scale user interaction.
        - **Is highly personal and customizable**:
          This model operates solely on user A's data, not on data from users B or C. As a result, the recommendations for each user will be entirely different, offering a deeply personalized and unique experience.

- The downside of this model is that it does not capture popularity trends.
    - **Item-Based Collaborative Filtering Model (Optional) by Scikit-Learn** for the random drink suggestion.
        - **Leverages User Behavior:** Identifies relationship between items and clicks, ratings, purchases, etc.
            - **Diverse Recommendations:** Unlike the content-based model, this model suggests items that the users might not have considered on their own
            - **Captures popularity trends**
- The downsides of this model is that new items can't be recommended to users until there has been enough interaction. For this reason, we decided to use the content-based filtering model for more personal suggestions (especially because our drink preference data will be very scattered at the beginning)
    - **Gemini (AI Images)**
        - AI images will be used for smaller items like Loading Screens, Icons, Logos, etc.
            - Although AI is the future, we want to give our front-end team the opportunity to learn how to create a variety of custom images, animations, and backgrounds. AI Images will be a smaller part of the front-end team's bigger creation.
        -

- 
AI
- 

4. Modules and Components (Internal Interfaces)

- **User Management Module:** Manages customer profiles, authentication, and user interactions.
  - Responsibilities
    - User registration and login
      - Email confirmation (Django)
    - Profile management
    - Preferences and order history
  - Components
    - User service: Handles user data storage and retrieval
    - Authentication service: Manages login, sessions, and secure password management
    - Recommendation service: Manages preferences and order history
- **Soda Catalog Module:** Manages the inventory of soda products and custom drink options.
  - Responsibilities
    - Product listing and categorization
    - Custom drink creation
    - Inventory management
  - Components
    - Product service: Manages operations for soda products
    - Customization service: Allows users to create custom sodas

        - Inventory service: Tracks stock levels and alerts for low inventory
- **Order Management Module:** Handles the order lifecycle from creation to delivery.
    - Responsibilities
        - Order placement and tracking
        - Payment processing
        - Order completion scheduling
    - Components
        - Order service: Manages order creation, updates, and status
        - Payment service: Handles payment transactions
        - Order completion service: Manages scheduling of orders and geolocation tracking
- **AI Recommendation Module:** Provides personalized and randomized soda recommendations using AI.
    - Responsibilities
        - Analyze user preferences and behavior
        - Generate product suggestions based on preferences
        - Improve recommendations over time
        - Generate random product suggestions
    - Components
        - Data Analysis Service: Analyzes user data and preferences for insights
        - AI Model: Generates recommendations based on past behavior and trends

5. Data Design

Data Model:

- **Key Entities**
    - **User:** Represents a person who uses the application, whether as a general user or an account user. The User entity stores all the necessary information about the user, such as login credentials and profile details.
        - **UserID:** (Primary Key) A unique identifier for each user.
        - **Username:** The user's chosen name for logging in and identification within the app.
        - **Password:** The user's password, stored securely
            - Encryption: **Yes**
        - **Email:** The user’s email address
            - Encryption: **Yes**
        - **UserRole:** Defines the role of the user (e.g., admin, manager, account user)
        - **OrderHistory:** A reference to all orders made by the user.
    - **Preference**: Stores individual drink preferences for each user. This table adheres to First Normal Form (1NF), meaning that each preference is stored as an atomic value rather than as a comma-separated list.
        - **PreferenceID**: (Primary Key) A unique identifier for each preference entry.
        - **UserID**: (Foreign Key) Links the preference to the user who selected it.
        - **Preference**: A string representing the individual drink preference (e.g., "Strawberry", "Vanilla").

- Example:

id	user_id	preference
1	1	Strawberry
2	1	Vanilla
3	1	Coke
4	2	Lemon
5	2	Root Beer
6	3	Vanilla

- **Order:** Represents a purchase made by a user, containing details about the drinks ordered, their status, and payment information.
  - **OrderID:** (Primary Key) A unique identifier for each order.
  - **UserID:** (Foreign Key) Links the order to the user who placed it.
  - **DrinkDetails:** Information about the drinks included in the order.
  - **OrderStatus:** The current status of the order (e.g., pending, completed, canceled).
  - **PaymentStatus:** Indicates whether payment has been completed, pending, or refunded.
  - **PickupTime:** The scheduled or expected time for the order to be picked up.
  - **CreationTime:** The time when the order was placed.
- **Drink:** Represents the various drink combinations that can be ordered by users. It includes information about the ingredients used and the drink's price.
  - **DrinkID:** (Primary Key) A unique identifier for each drink.
  - **SyrupsUsed:** A list of syrups included in the drink.
  - **SodaUsed:** The type of soda used in the drink.
  - **AddIns:** Any additional ingredients added to the drink (e.g., ice, fruit).
  - **Rating:** Optional, allows users to rate the drink.
  - **Price:** The cost of the drink.
- **Inventory:** Represents the items available in the store's inventory, including syrups, sodas, and add-ins. Tracks the quantity and threshold levels for restocking.
  - **InventoryID:** (Primary Key) A unique identifier for each inventory item.
  - **ItemName:** The name of the inventory item (e.g., a specific syrup or soda brand).
  - **Quantity:** The current amount of the item in stock.
  - **Threshold Level:** The minimum quantity before a restock notification is triggered.
  - **LastUpdated:** The date and time when the inventory was last updated.
- **Payment:** Represents payment transactions associated with orders. It stores information about the payment process and its status.
  - **PaymentID:** (Primary Key) A unique identifier for each payment transaction.
  - **OrderID:** (Foreign Key) Links the payment to the specific order.
  - **UserID:** (Foreign Key) Links the payment to the user who made it.
  - **Amount:** The total amount paid.
  - **PaymentMethod:** The method of payment used (e.g., credit card, PayPal)
  - **PaymentStatus:** Indicates whether the payment was successful, pending, or failed.
  - **RefundStatus:** Indicates if a refund has been issued
    - Encryption: **Yes** (Sensitive information such as payment details should be encrypted).

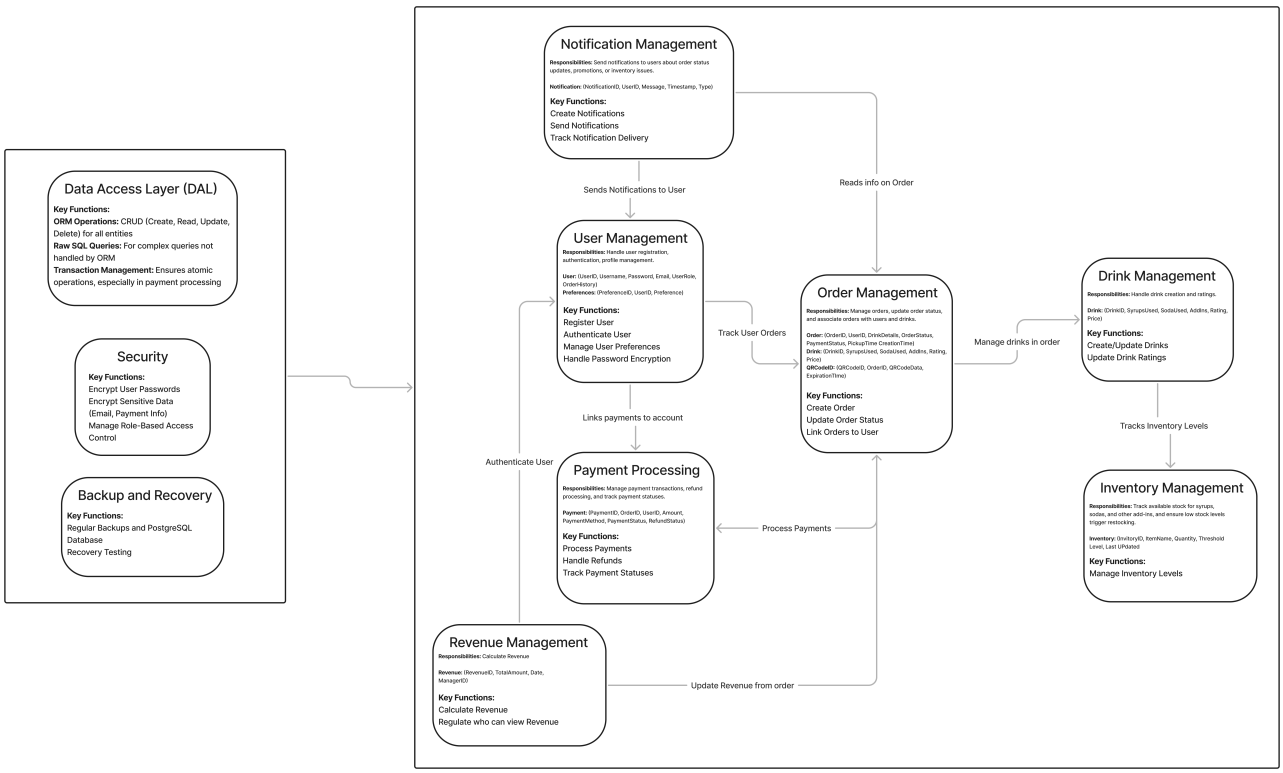
    - **Notifications:** Represents notifications sent to users, informing them of order status updates, inventory issues, or other important messages.
        - **NotificationID:** (Primary Key) A unique identifier for each notification.
        - **UserID:** (Foreign Key) Links the notification to the specific user.
        - **Message:** The content of the notification.
        - **Timestamp:** The time when the notification was sent.
        - **Type:** The type of notification (e.g., order update, promotional message).
    - **QRCode:**
        - **QRCodeID:** (Primary Key) A unique identifier for each QR code entry.
        - **OrderID:** (Foreign Key) Links the QR code to a specific order that contains soda.
        - **QRCodeData:** The data or link that the QR code contains (e.g., a link to open the fridge).
        - **ExpirationTime:** The time when the QR code expires and is no longer usable.
    - **Revenue:**
        - **RevenueID:** (Primary Key) A unique identifier for each revenue entry.
        - **TotalAmount:** The total revenue generated from completed orders.
        - **Date:** The date when the revenue was recorded (e.g., daily, weekly, or monthly).
        - **ManagerID:** (Foreign Key) Links to the manager who has access to the revenue data.
- **Relationships**
    - **User to Order:** One-to-Many (A user an place multiple orders)
    - **User to Preferences:** One-to-Many (Each user can have multiple preferences, and each preference is a unique value for that user).
    - **Order to Drink:** Many-to-Many (An order and include multiple drinks, and a drink can be part of many orders)
    - **Revenue to Order:** One-to-Many (Revenue is updated when an order is completed)
    - **Revenue to User:** One-to-Many (Only a User with the Manager role has access to view revenue)
    - **Inventory to Drink:** Many-to-Many (A drink can use multiple inventory items, and an inventory item can be used in multiple drinks.)
    - **User to Payments:** One-to-Many (A user can have multiple payments)
    - **User to notifications:** One-to-Many (A user can receive multiple notifications)
    - **Order to QRCode:** One-to-One (A QR code is generated an order)
    - **Notification to QRCode:** One-to-One (The QR code is sent to the user as notification)

Database Design:

- **Relational Database (PostgreSQL):** Well-suited for managing complex relationships and ensuring transactional integrity. PostgreSQL supports advanced features like JSONB fields, which can be useful for storing unstructured data like user preferences or dynamic order details
- **Major Tables in PostgreSQL** (Mapped from Django Models):
    - **Auth_user Table:**
        - Stores user data.
        - Columns: id: username, email, password, role, dataJoined, lastLogin
    - **Order Table**
        - Stores order data.
        - Columns: id, userId, status, paymentStatus pickupTime, createdAt
    - **Drink Table**
        - Stores drink details.
        - Columns id, itemName, quantity, thresholdLevel, lastUpdated

- **Inventory Table:**
  - Stores inventory data.
  - Columns: id, itemName, quantity, thresholdLevel, lastUpdated
- **Payment Table:**
  - Stores payment information.
  - Columns: id orderId, userId, amount, paymentMethod, paymentStatus, createdAt
- **Notification Table:**
  - Stores notifications.
  - Columns: id, userId, message, timestamp, type
- **Preference Table:**
  - Stores user preferences for drinks
  - Columns: id, userID preferences
- **Revenu Table:**
  - Stores store revenue data accessible to managers
  - Columns: id, totalAmount, date, managerId
- **QRCode Table:**
  - Stores QR codes for orders that include sodas. These codes allow users to open the fridge
  - Columns: id, orderId, QRCodeData, expirationTime

- **Relationships in PostgreSQL:**
  - Defined through Foreign Keys and Many-to-Many relationships in Django models. Django automatically handles join tables for Many-to-Many relationships.
- **Indexes**
  - Django creates indexes on primary keys and foreign keys by default. You can create additional indexes using the db_index=True option in model fields or Meta class options for performance optimization.



Data Access Layer:

- In Django, the data access layer is primarily handled through the ORM, which abstracts SQL queries into Python code.
- **Data Access Patterns:**
  - **ORM Queries**: Django's ORM allows you to perform CRUD operations (Create, Read, Update, Delete) directly through model methods. For instance
    - *Order.objects.create(user=user, status='Pending', ...)*
    - *Order.objects.filter(user=user).order_by('-created_at')*
  - **Raw SQL Queries:** Django allows executing raw SQL if needed using *Model.objects.raw()* or *connection.cursor()*. This is useful for complex queries or optimizations.
  - **Transitions:** Django provides transaction management via *transaction.atomic()*, ensuring that operations like placing an order and processing a payment are treated as a single transaction.
  - **Caching:** Use Django's caching framework with backends like Redis for improving performance on frequently accessed data (e.g., drink menus, user preferences)
- **Migration Management:**
  - Django handles database migrations through the makemigrations and migrate commands. This keeps the database schema in sync with your models.
- **Data Encryption:**
  - Django automatically hashes passwords using PBKDF2 by default, and you can enhance this with Argon2 or bcrypt.
  - For sensitive data fields, consider using Django-encrypted-fields or custom encryption methods.
- **Access Control:**
  - Implement Django's built-in permissions and groups to enforce role-based access control.
  - Use Django's middleware and decorators (@login_required, @permission_required) to secure views.
- **Backup and Recovery:**
  - Regularly back up the PostgreSQL database using tools like pg_dump. Automate this process with cron jobs or Django management commands.
  - Test recovery procedures to ensure data can be restored effectively.

6. Integration Points (External Interfaces)

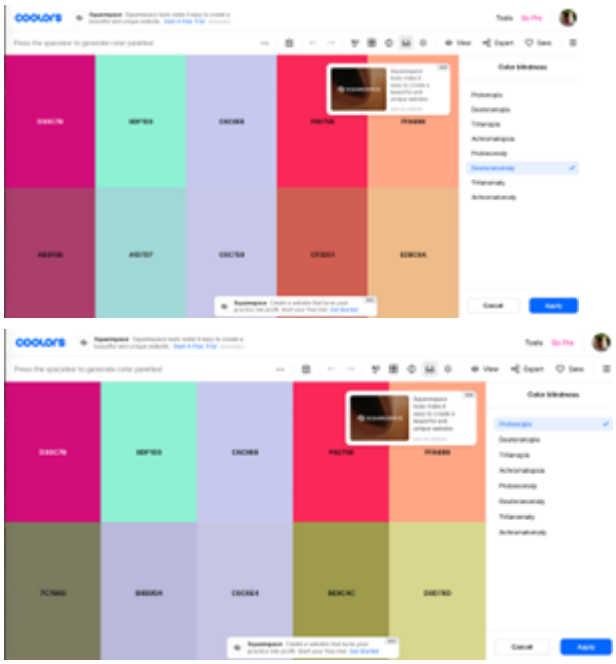
- **External Systems and APIs:**
  - **Payment System:** Stripe
    - Free
    - Secure payments- offers built-in fraud prevention tools
    - Support for variety of payment methods- like Apple Pay or Google Pay
    - Erik is familiar with it- we can troubleshoot with him if needed
  - **Geolocation:** Mapbox
    - Offers Python SDKs (meaning we do not need to manually create HTTP requests. The SDK handles that for us)
    - Free for up to 50,000 geolocation requests/month
    - Many other tools require third-party libraries to implement features (ex. OpenStreetMap), creating a more extensive setup process. Mapbox does not have this
  - **AI Chatbot:** DialoGPT from huggingface.co
    - Basic AI to run the customer help/complaint center of the app

        - This is a Natural Processing model (NPL)
        - DialoGPT is trained on conversational datasets, making it naturally suited for a chatbot
        - Low configuration, making it a quick setup (We aren't too concerned with this being a super high functioning bot. Most of our concern will be with the drink suggestion AI models)
        - Free
        - Can be implemented with Python
    - **Notifications**
        - **Push:** Firebase Cloud Messaging (FCM)
            - One of the most widely used and robust push notification services
            - Free
            - Integrates well with mobile platforms (Android and iOS), web push notifications, and backend server to send notifications
                - Firebase Admin SDK includes option for Python in the backend (which is likely where we will implement the push notifications)
                - By using the SDK instead of REST API we will not have to worry about HTTP requests
        - **Email:** Django
            - When the user signs up to the app or website, they will be sent a confirmation email to ensure they used the right email address and that they are the ones signing up.
            - Django's email functions can be used to accomplish this (send_mail()), using a token to verify the email. While this isn't external, it is included in this section since it is related and putting it here makes it easy to find.

7. User Interface (UI) Design Overview

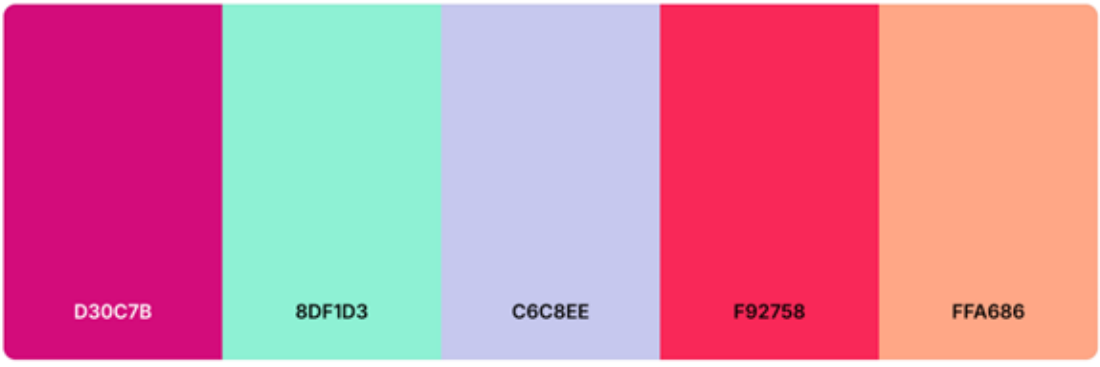
- **UI/UX Principles**: High-level UI/UX principles (e.g., responsiveness, accessibility).

    - We aim to keep the app simple and intuitive so as to provide a frustration free user experience for all our users as our app has a wide target audience.
    - The design focus will be primarily for a phone application but we will also make sure the interface is responsive and compatible with any interface. We will utilize flex-box in the CSS design to ensure this because it is good for responsive design.
    - Design color choices and navigation style will stay consistent for all types of users including managers and admin accounts so users remain familiar with the layout.
    - Navigation will primarily happen through a nav bar containing descriptive graphic icons that will persist on all pages of the app. With this, a user is able to access all the app's functionality more easily from one to two clicks.
        - Some exceptions to this include obvious and brightly colored buttons for navigation to pages such as the account creation page or the payment page which is accessed from the cart.
    - Accessibility
        - Color blindness
            - The color palette chosen is shown in the following graphics as seen by some of the more common forms of color blindness.
            - Based on this analysis, colors like teal and purple will not be used right next to each other in the app so as to keep easy readability for all users.



    - Each page will have screen-reader compatibility and tab-controlled navigation options.
    - Web Content Accessibility Guidelines (WCAG)

- **Mockups**: High-level mockups or wireframes of key screens.
    - Color way
        - The color way has been chosen specifically to reflect the bright colorful nature of the app while also providing good contrast for useability.
        - Hex values (L-R)
            - D30C7B
            - 8DF1D3
            - C6C8EE
            - F92758
            - FFA686
    - Style Guide
        - Corners of boxes and buttons will be rounded.

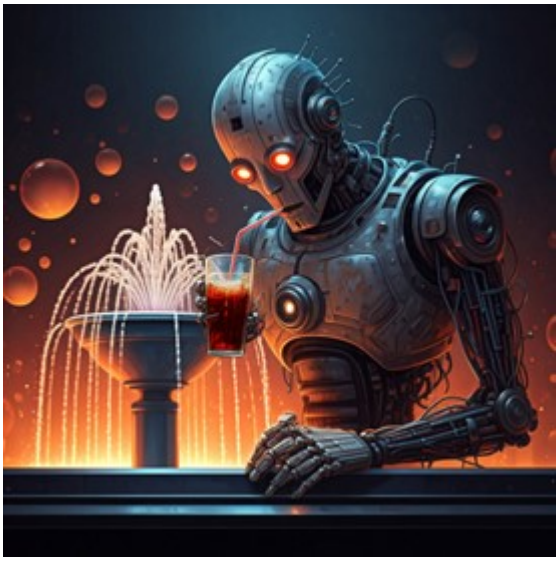


- **Navigation Flow**: Overview of how users will navigate the app.
    - Pages will not be more than 2-3 clicks deep
    - Pages:
        - Home page
            - Nav bar

        - Cart button - link to cart page
        - Link to drink design page
        - Link to Account user home page
        - Link to complaints page
    - Seasonal drinks menu carousel
    - Generate random button (from AI)
    - Create account button (for non-account users)
- Sign in page
    - Simple page with text entry boxes for username and password
    - Login button
    - Automatically displayed error message or taken to home page after login
- Complaints page
    - Simple page with a text entry box with a complaint prompt - users will receive AI generated response messages after entering complaints
- Account user home page
    - Saved drinks
    - Update preferences button
        - Favorite drinks and favorite ingredients for users to take into account
        - Option to enable/disable geolocation
- Payment page
    - User is taken here from the “checkout” button in the cart
    - Stripe API used to take user payment information
    - After payment information is submitted, there is a notification for users
    - Option for user to track with geolocation (default selected) or select a time for it to be ready
        - If geolocation is disabled this button should be grayed out and there should be a message letting the user know how to enable geolocation
- Cart page
    - Drinks
    - Options to remove things from cart
    - Button to checkout
- Confirmation page
    - After a user pays for their drink, they are taken to a page with a link to the complaints page (Didn’t get their drink?” button) as well as a rate your drink section where a user can rate their drink out of 5.
- Drink design page
    - generative/responsive graphic created when a user makes drinks
    - Add-in options are displayed with easily identifiable graphics instead of a list so options are easy to choose
    - There is a way to search for options
    - A van bar for different add in options
    - Also a way to remove options - have the graphics be selected (added) or unselected (removed) with a visible difference for ingredients that are added
    - Drink graphic, nav bar (soda (can choose more than one), syrups, juices (lemon, lime, pineapple, coconut etc.), ice (light, regular), extra, no ice), search bar
    - An add to cart button

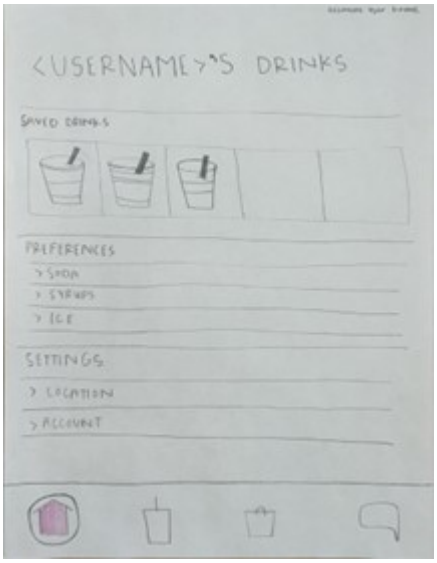
        - A size and soda selection are required to add to cart, everything else is optional and the default is “none”. An error will pop up if the user tries to add a drink to the cart without selecting a drink size or soda.
- Manager dashboard
    - A dashboard that contains links to a store revenue report and a store inventory report.
        - Data such as total revenue, inventory costs, total user accounts will be displayed in an easily understandable format
    - AI will be used to estimate when supplies need to be ordered to notify the manager and also find the best places to purchase ingredients.
- Admin dashboard
    - A simple dashboard to view all functional user accounts with options to delete, disable, and reinstate accounts. An admin also has the permissions necessary to create manager accounts and grant managers permission to view certain data.
- Loading screens
    - Typical loading screen:

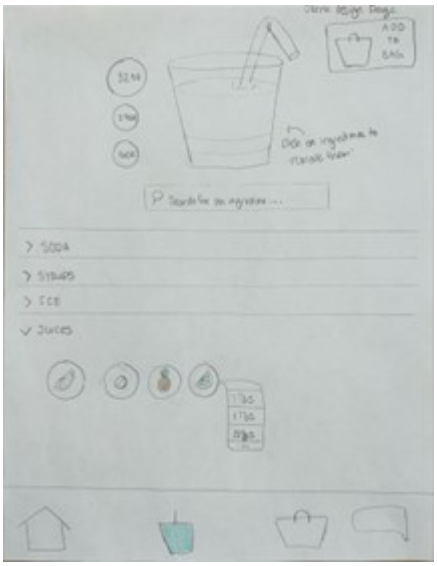
    - Loading screen for customer service:
        - Bob



- UI diagrams:

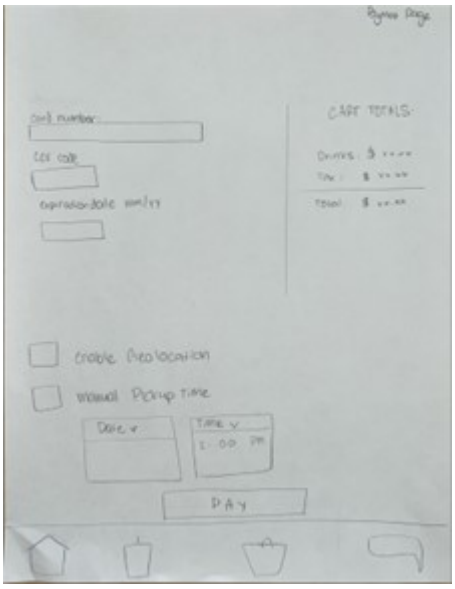

Codepop
CREATE ACCOUNT

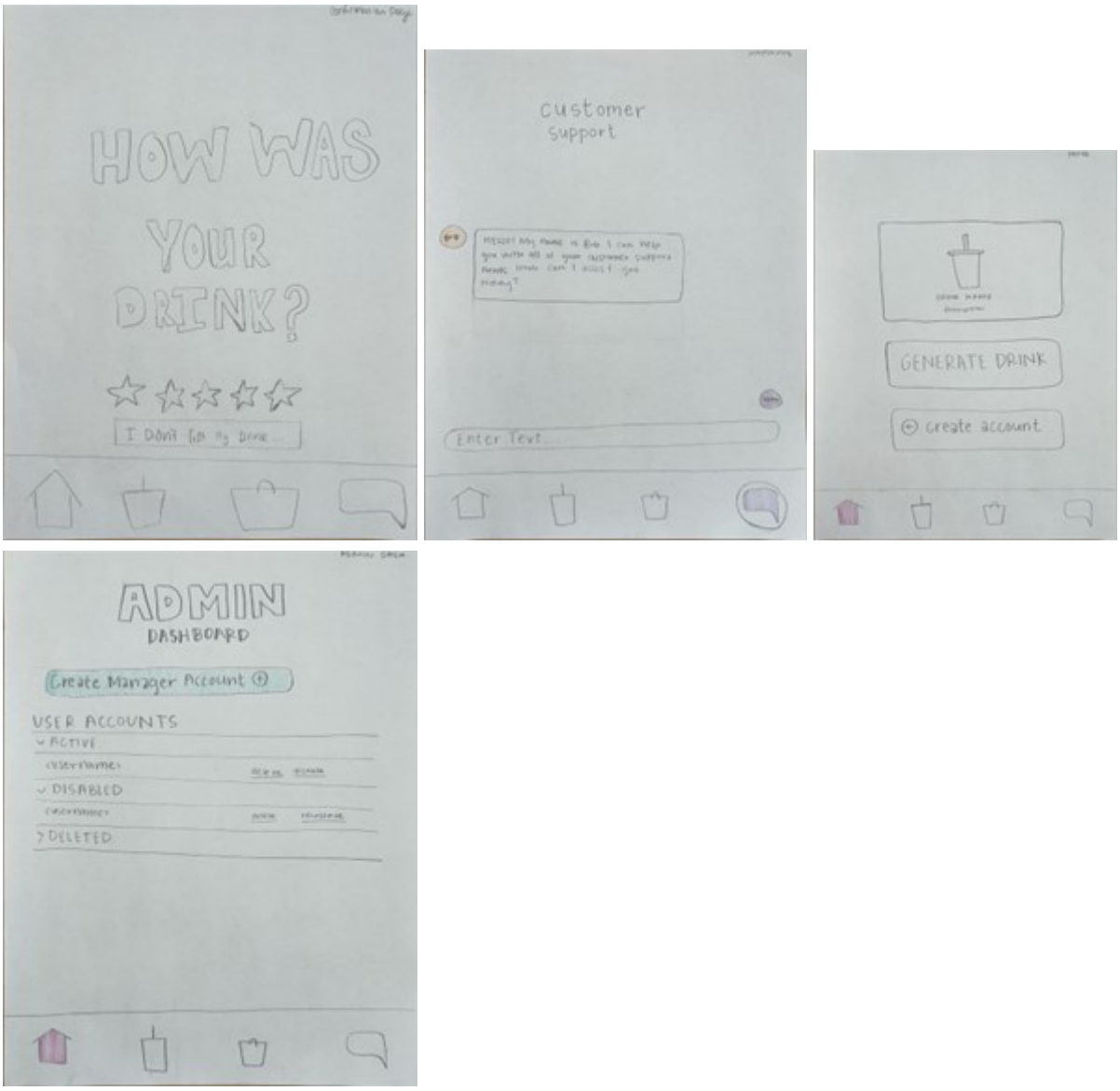

<USERNAME>'S DRINKS
SAVED DRINKS
PREFERENCES
> SODA
> SYRUPS
> ICE
SETTINGS
> LOCATION
> ACCOUNT


ADD TO BAG
> SODA
> SYRUPS
> ICE
> JUICES


MANAGER
DASHBOARD
INVENTORY REPORT


<USERNAME>'S ITEMS
$ 2.50
CHECK OUT


CART TOTALS
enable Geolocation
manual Pickup time
PAY



8. Input and Output (I/O)

Note: Much of this section may be a repeat of what has already been documented, but it is repeated here to make I/O items easier to find and relate to each other.

- Input
    - User Information
        - Username
        - Email
        - Password
        - Preferences
        - Payment Method
        - Customer Complaints
    - Geolocation (MapBox)
        - How close the user is to a store location
        - If user does not consent to geolocation, an “I’m ready” button or a set time will be input by the user instead
    - Stripe
        - Confirmation that the user’s payment went through

    - AI
        - AI chatbot responds to user complaints and allows for further response (from user)
        - AI drink results are given back to the user with an option to confirm or rerandomize
    - Navigational Input
        - User will use buttons, drop-down menus, etc. to navigate through the app/website

- Output
    - Notifications
        - User will get notifications either through email (sign up confirmation) or by push notification (drink-is-ready indicator, event notifications [ex. Birthday, holiday, change to seasonal menu], etc.)
    - Geolocation (MapBox)
        - Start tracking once the user has given consent
    - Stripe
        - Send user payment to simulate a purchase
    - AI
        - Customer complaints sent to an external AI chatbot
        - User preferences given to AI to randomize a more personalized drink (more likely to choose preferences over something completely random)
        - User ratings used to train the AI as to what flavors/drinks are more popular
    - UI Output
        - User navigation input will bring them to different screens/sections of the app that will be shown visually to the user
    - Store Information
        - API may be used to show the manager graphs of the store's revenue, stock, etc.

9. Security 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.

10. Testing Strategy

- **Unit Testing**

  CodePop will implement unit tests as we go along with our software production.

- Unit tests will provide an automated way to run tests and prevent the need to manually input over and over again.

- These tests will be created as the project gets created. As an example, once all of the sign-in page functionality is up-and-running, unit tests will be created to ensure the user can only input valid emails, and that everything gets properly stored in the database. Creating the tests in this fashion will:

    - Prevent rushing them later on in the project's development.
    - Ensure a section works before moving on
    - Make testing easier as developers merge code together. Did someone's merge break someone else's unit test?
      Unit testing may add more complication to the project, especially if developers are not that familiar with it. However, unit testing will ensure that the project has less bugs and also provide a visual as to what has and hasn't been tested.

- **Manual Testing**

  In the case that unit tests do not work, manual testing will be used.

- In order to make the testing as smooth and consistent as possible, a document will be created describing each test case, that way everyone on the team has access to every test (and could copy/paste).

- Without this document, test cases may get left out during testing or forgotten, which will cause problems later on in development.

11. Risks and Mitigations

- **Identified Risks**: List of known risks (e.g., technology choice, dependencies).
- **Mitigation Plans**: Strategies for addressing these risks.
- **User Geolocation**
    - **Risk:** Geolocation will be used to track how close the user is to the CodePop location. However, there is a chance this gets hacked and the user's location will be revealed and tracked by unknown parties
    - **Mitigation:** User location will be encrypted/hashed so it is more secure, and location will be accessed sparingly throughout the program. The user also has the option to opt out of geolocation and set a time for their drink to be ready instead.
- **User Input (AI)**
    - **Risk:** AI could get fed bad input from the user preventing it from working properly or causing it to reveal secure information.
    - **Mitigation:** Users will either not be able to directly input into the AI, or in cases where they do input (i.g. Preferences, complaints) user input will be searched for any risky words or symbols, which will then get parsed out before being sent to the AI.
- **Payment Information**
    - **Risk:** Anytime we deal with people's money there is a big risk that relevant data will be hacked resulting in financial harm to our customers
    - **Mitigation:** We will be using Stripe's payment API so that we avoid directly handling our customers' sensitive information. This will allow our customer's data to be kept safe by Stripe who has much more time and money to create robust security than we do.
- **Allergies**
    - **Risk:** Some of our customers may have food allergies that could result in bodily harm if they get contaminated drinks.
    - **Mitigation:** We will need to clearly label what allergens a drink contains so that a user can make an informed decision when they purchase a drink and ensure that it won't cause harm to them.
- **User Information**
    - **Risk:** User accounts may be hacked and information may be stolen or used to buy items through the app.
    - **Mitigation:** We will encrypt sensitive user information in the database (i.e. email, password). We will also allow users to contact administrators if they fear their data has been compromised to allow them to either freeze their account or change passwords.
- **Location Revenue Information**
    - **Risk:** Manager accounts may be hacked and information may be stolen.
    - **Mitigation:** We will encrypt sensitive user information in the database (i.e. email, password). We will have a stricter password policy for managers and admins that will require them to have longer, more complicated passwords.

- **Legal Issues**
    - **Risk:** Customers may try to hold the company liable if something were to happen such as the AI generating them a drink that has something they are allergic to in it.
    - **Mitigation:** We will have a warning statement for non account users when they order a drink created though AI. Account users will sign an agreement upon account creation to agree to not hold the company liable for personal harm.

Interactions Diagram

Below is a breakdown of interactions in the CodePop app. To obtain the information on the far right column, our app will utilize HTTP Requests and Django's ORM.

For future reference, the following will be handled in app:

- Email confirmation (By Django's built-in function, send_mail())
- AI drink suggestions (By Scikit-Learn. This will be built into the CodePop app.)