HomeCooked
CS 307
Design Document

Team 5:

Colton Scott, Akhil Thata, Hawkins Peterson
Sebastian DSouza, Vincent Cui, and Yash Velagapudi
Github Link: https://github.com/athata1/HomeCooked

Index

1. Purpose

a. Food scarcity and waste are at an all-time high, with a slight improvement and the solutions to this problem typically require professional chefs to make costly food. Our app, HomeCooked*,* will allow the average person a platform to share their home-cooked food to clear space in the fridge and share culture. The consumers will be able to access healthy, home-cooked meals that they wouldn't have access to otherwise, due to either not having enough time or money to make said meals personally.

Functional Requirements:

- As a user, I'd like to
  - Be able to create an account
  - Be able to login to said account after creating it
  - Be able to set name, password, and email address
  - Be able to send emails to users when they ask to forget their password
  - Be able to see the change password verification screen
  - Be able to view the change password screen once the password has been changed
  - Be able to change your password using the "Forgot Password" button
  - Be able to view a settings page so that I can list a pickup address, display reviews, and show a biography
  - Be able to change their address and biography
  - Be able to view dropdowns for states and city
  - Be able to change their profile picture
  - Be able to edit and have a template to format a profile page
  - Have the ability to log out of the page
  - Have the ability to delete an account
  - Be able to search for other consumers and producers
  - Be able to view other producer and consumer profiles
  - Be able to create reviews for producers
  - Be able to utilize a text-based message system
  - Be able to see all current message threads and click on them to view the message board
  - Be able to view their history of interactions on the app.
  - Be able to generate receipts for food they ordered/supplied
  - Ability to export data from an event you are signed up for to your calendar
- As a Producer, I'd like to
  - See my overall rating of myself on my profile page
  - Create a list of food items I'm willing to offer/sell
  - Be able to list a meal on the menu
  - Be able to unlist a meal

  - Be able to label a meal as already sold
  - Import an image to represent a meal
  - Set a pickup location for meals
  - Add ingredients to the food item
  - View labels detailing potential allergies
  - View all outstanding meals available
  - View all meals transactions that have been completed
  - Only provide an address to those to whom I decide to give.
  - Unlist a meal from the page
  - Have the ability to quickly change to a consumer mode/account.
  - Be able to communicate with consumers via an in-built chat application
  - Link to other apps / leave my phone to talk with consumers further
  - Be able to end communication with a consumer/block a consumer
  - Be able to add a personal label to meals as vegetarian, vegan, or other similar qualities
  - See my reviews left by people who have eaten the food
  - Reply to consumers who message inquiries about the food
  - Be able to disclose the meetup location
  - Create an event on the Events page
  - Modify an event page's time, location, and description
  - Get notified of RSVPs and comments
- As a Consumer, I'd like the
  - Ability to search areas other than my current location by use of zip-code
  - Ability to view all people who are providing food within the city of choice
  - Ability to view all offerings of a person
  - Ability to see all offerings in the city of choice
  - Ability to see an image of food being given
  - Ability to see description of the offering
  - Ability to see ingredients that are used in the offering
  - Ability to view the city location of the Producer
  - Ability to view a pickup location for a meal
  - Ability to rate the meal they picked up
  - Ability to communicate with producers via an in-built chat application
  - Ability to end communication with the producer when no longer interested in an item
  - Allow consumers to message via different platforms
  - Ability to leave a review of a post, with a rating of 0-5 and some text
  - Ability to direct message a post owner
  - Ability to switch to producer mode

- Ability to view all items that I have been given and the information about the item.
- Ability to sign up for events that have been listed and receive notifications when they are upcoming
- Ability to get notified of event changes when they are made from an automated message/notification

Non-Functional Requirements

Architecture/Performance: As developers, we plan on having a separate backend and frontend for our application. The backend will be written in Python using Django. Django is a web framework that follows a Model View Template format and comes with a login system, database connection, and a plethora of libraries to utilize.

For the front end, we will be using React.js since it is scalable and simple to develop. In addition, the reusable components of React will remove a lot of redundant code, making the development process more efficient.

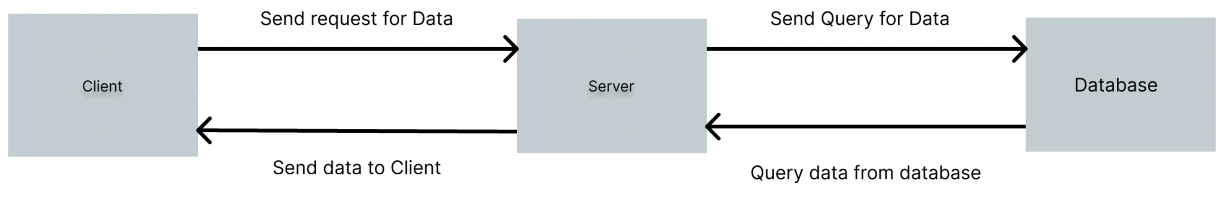
Security: As developers, one issue we see with most services like this is rate limiting. To prevent users from spamming meals, we plan to add a form of rate limiting, which will allow producers to put at most 10 items up within 24 hours of each other. For our SQL statements, we plan to use parameterized statements as opposed to injecting raw string data to prevent SQL injections into our code.

Usability: As a user, the interface should be easy to navigate as well as simple to understand for a consumer to browse through, as well as well equipped with the tools and services for the consumers to advertise and showcase their products for the consumers. We would like for our application to work in polynomial time and minimize latency to at least a second.

Hosting/Deployment: Since the application has both a frontend and a backend, as developers, we plan on hosting it on Heroku. It allows users to host full-stack applications in a fast and efficient manner.

2. Design Outline

a. Client Server Model



b. We chose the Client-Server model because of its simplicity and usability, given the tech stack we decided to use. The Django server will act as a data hub for all the clients who will send requests to receive the data they need to run the website. The SQL server will serve as a relational database for our application that will allow our server to store the client data.

c. Client
   i. Clients will create the UI for our users
   ii. Clients will receive data from the server and format the UI to cater to the data that needs to be presented.
   iii. Clients will send JSON requests to the server alongside a token when needed to access data private to the user

d. Server
   i. Server will serve as an intermediary between the data being stored in the database and the client who receives the data
   ii. Server will validate tokens given by the client and will either accept or reject the request
   iii. Server will send queries to the database to poll the appropriate data

e. Database
   i. Database will store all of the user's data as well as public data
   ii. Database will receive queries to either create, delete, or mutate the data
   iii. Database will send appropriate data back to the server to either mutate further or immediately send it back to the client to format into the UI.

3. Design Issues

Functional Issues

a. Issue 1: What credentials are needed to log in and what should be unique
    i. Option 1: Username and Password
    ii. Option 2: Username, email, and password
    iii. Option 3: Username, password, and phone number

   Choice: Option 2

   We chose option two because we plan to implement a forgotten password on our login page. In order to do so, we must have an email to send the verification to in order to allow the user to reset their password

b. Issue 2: How should users be able to tag foods?
    i. Option 1: Allow users to manually add tags to their foods
    ii. Option 2: Make the application decide what tags to add to the food
    iii. Option 3: Both options for tagging

   Choice: Option 3

   The reason we opted to allow for both is that we know that our users will not be 100% knowledgeable on how foods intersect with each other and what allergies they may cause. So we will allow the system to mark food with warnings based on the ingredients listed. Also, since users may want to market their food differently, allowing them to tag their food will be beneficial as well.

c. Issue 3: To what specificity will we have our user's locations publicly shown?
    i. Option 1: Give the address of the user's drop-off location
    ii. Option 2: Only give city/state

   Choice: Option 2

   We want our users to have enough privacy to where they can give their location only to users who they completely trust. Because of this, we will only provide the city and state of the Producer until they trust the consumer enough to disclose their specific dropoff location in the chat.

d. Issue 4: How should users be able to create posts
    i. Option 1: Create recipe then be able to turn it into a post
    ii. Option 2: Create posts individually

   Choice: Option 1

   A user may want to create multiple posts for a singular item type. As such, allowing the user to create recipes, then turn recipe into a post will allow them to create multiple of the same post easier

e. Issue 5: What should be done to allow for communication for events?

i. Option 1: Create a discussion post, with the ability to reply
ii. Option 2: Create a comment thread about the event
iii. Option 3: No thread for communication

Choice: Option 1

We opted for using a discussion post method similar to Twitter/Reddit style because it fosters communication not only between producer and consumer, but also consumer and other consumers.

Nonfunctional Issues

a. Issue 1: What frontend technology should be used?
   i. Option 1: Vanilla JS
   ii. Option 2: React
   iii. Option 3: Angular

   Choice: Option 2

   We decided to use React because it is already a strong framework within the frontend toolsets. Also, there is a lot of functionality for signup/login through firebase, which we plan to utilize.

b. Issue 2: What backend technology should be used?
   i. Option 1: NodeJS
   ii. Option 2: Django
   iii. Option 3: Spring Boot

   Choice: Option 2

   While we all have java experience, we opted for using Django because it has a SQL database built into the software itself, which will be useful for our persisting data

c. Issue 3: What database should we use?
   i. Option 1: Postgres
   ii. Option 2: MongoDB
   iii. Option 3: SQL

   Choice: Option 3

   As previously stated, Django comes with SQL prebuilt into the server, so it will be easier to integrate than the other two databases mentioned

f. Issue 4: How will the useability of the application be emphasized for the users
   i. Option 1: Ensure fast algorithms
   ii. Option 2: Ensure efficient data usage

iii. Option 3: All of the above

Choice: Option 3

Having both a fast and reactive application as well as an efficient data storage paradigm will allow the user to have better usability of the software being developed.

g. Issue 5: Where will we be hosting

i. Option 1: Heroku

ii. Option 2: Fly.io

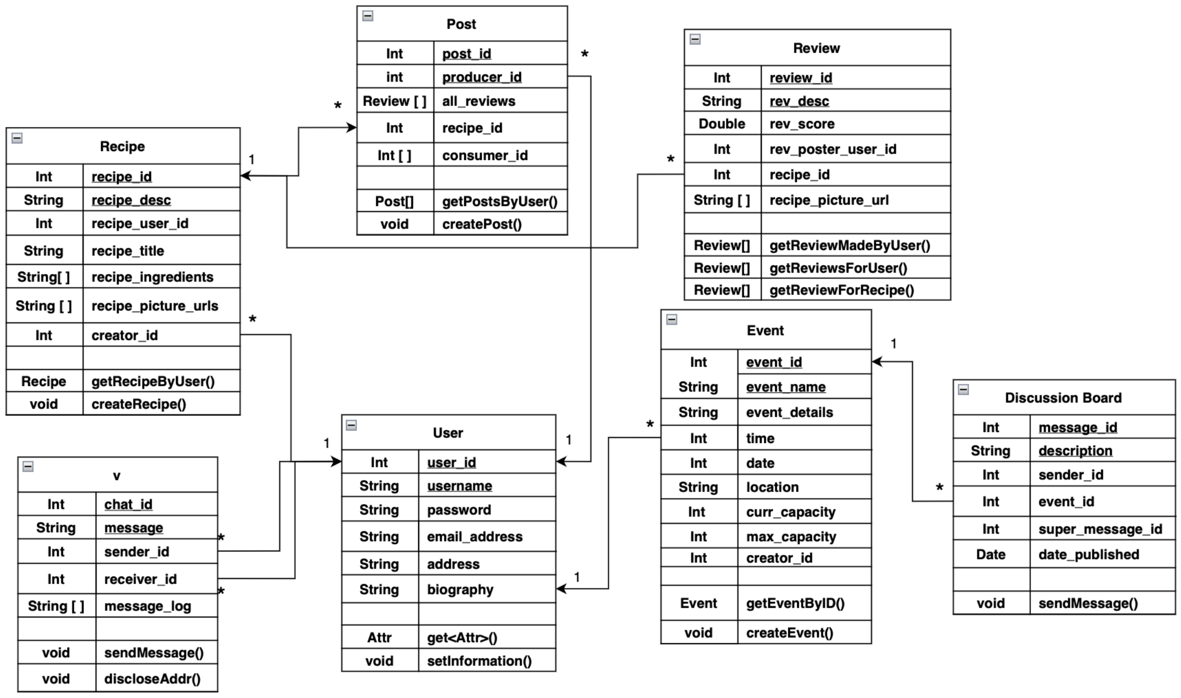
iii. Option 3: AWS

Choice: Option 1.

We opted to use Heroku due to many of us having predisposed experience with utilizing Heroku for web application projects. It allows for both frontend and backend hosting capabilities

4. Design Details

Class Design



Descriptions of Classes and Interactions Between Classes

The classes are created based on the object within our application. Each class has a list of attributes owed by each instance of an object.

User

- A user object is created when a person signs up
- Each user has a unique ID
- Each user has a unique username, password, and email for login
- Each user will pick their username, location, biography, and password
- Each user will have a list of recipes they have tried/picked up from a producer

Posts

- Each post has a unique post ID
- Each post will have a poster ID of the user who posted it
- Each post will have a record of reviews attached to it
- Each post will have a seller and customer ID
- Each post will have a rating function to calculate the overall rating of the product within the post

Reviews

- Each review will have a unique ID
- Reviews will have review descriptions
- Each review will have a poster ID
- Each review will have a post ID it will be associated with
- Each review will have a review score
- Reviews will have pictures associated with it

Events

- Each event will have a unique ID
- Events will have a date, time, location, current capacity, and max capacity fields
- Each event will be associated with a creator (user) ID

Discussion Board

- Each discussion board will have a description
- Each discussion board will have an event ID attached to it
- Each discussion will have message IDs attached to it

Recipe

- Each recipe will have a unique ID
- Recipes will have a description, title, ingredients, and picture attributes
- Each recipe will have a creator ID
- Each recipe will be linked to a post

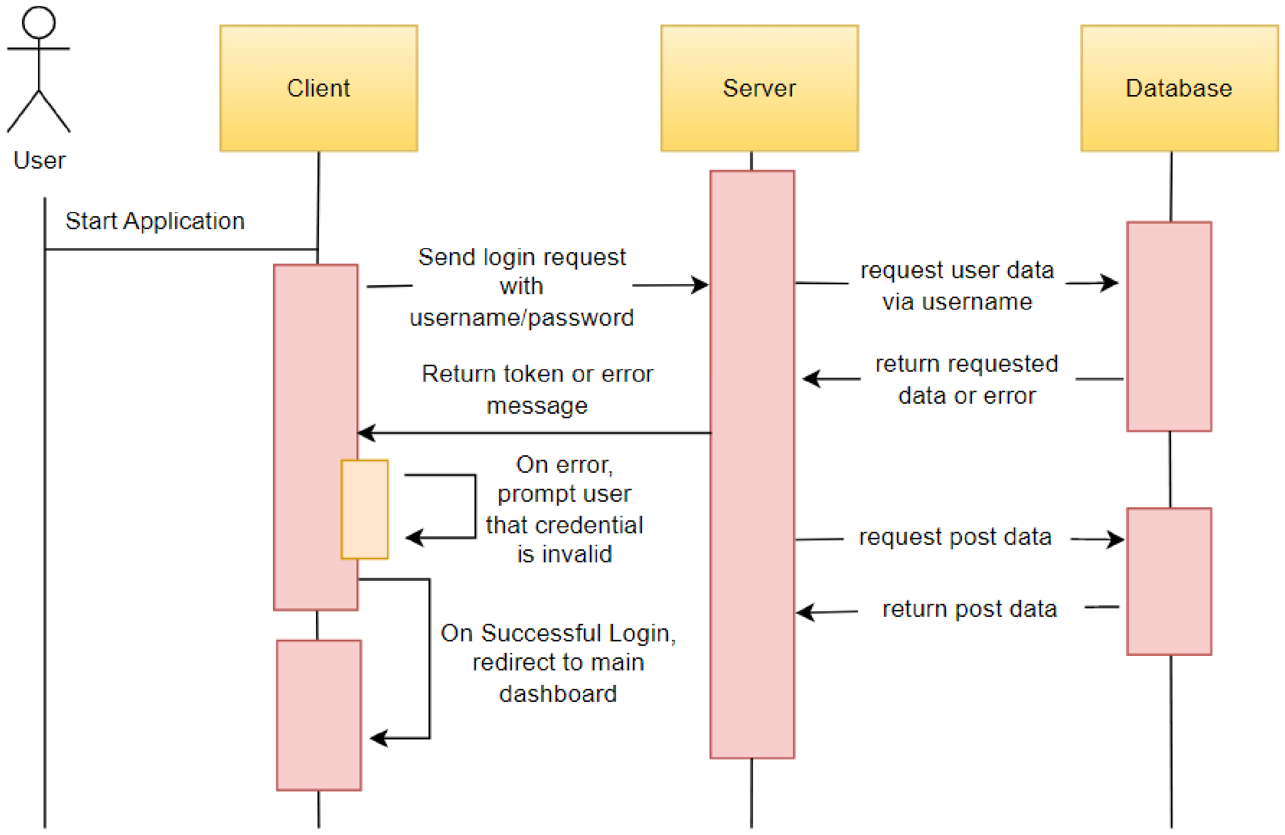
Chat Message

- Every chat exchange will have a unique chat ID
- Every chat exchange will have sender and receiver IDs
- Every chat will have a message history and a message attached to it

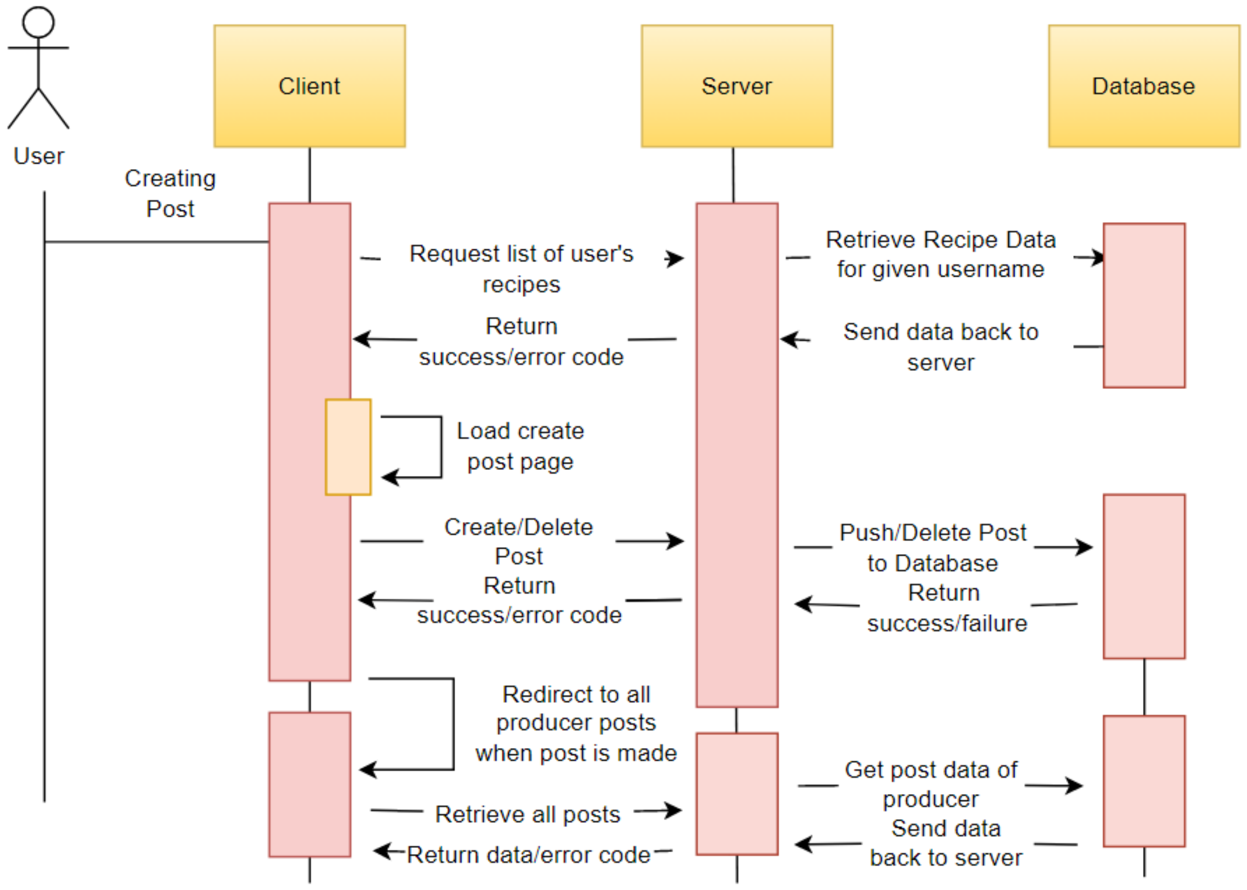
Sequence Diagram

The diagrams below will show the user flow of major events within this application such as user login, creating a post, creating an event, and password reset. When a user performs certain actions on the interface, the UI will send a request to the server which will send a data query to the database to retrieve data. The server may have some middleware to perform some data processing before sending the appropriate data back to the user display.

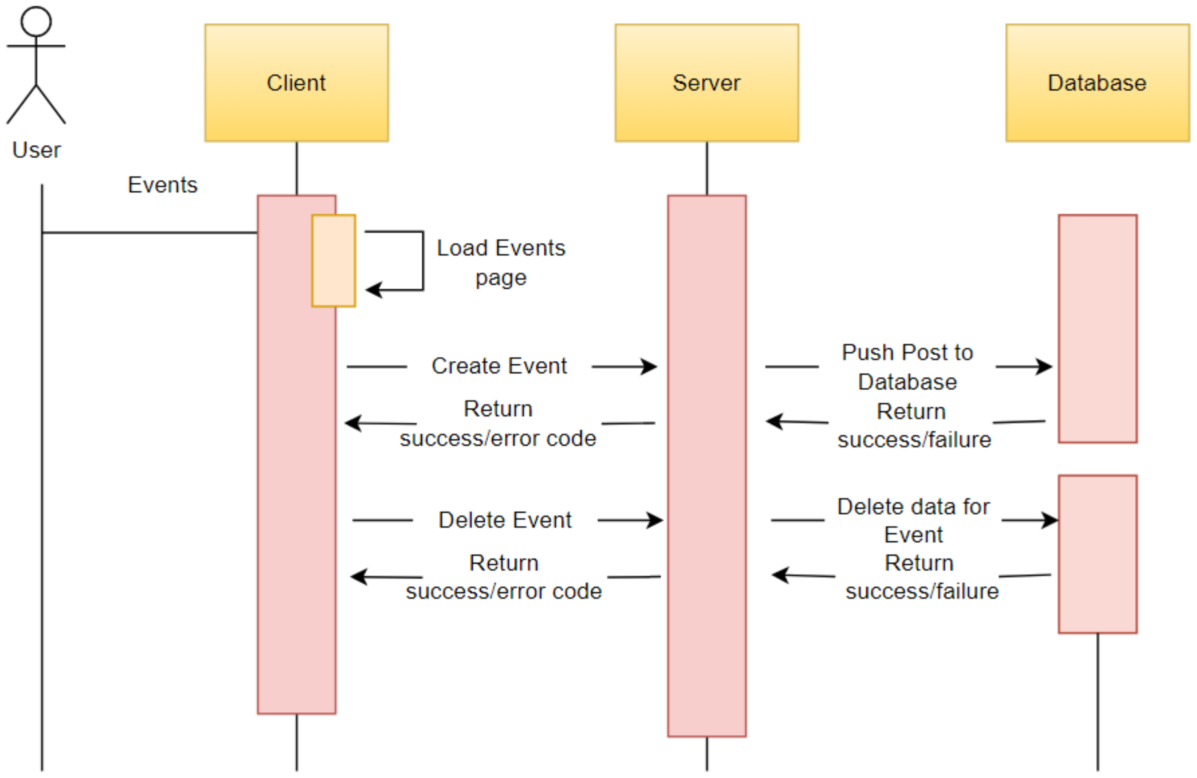
1. Sequence of events when a user logs in


User
Client
Server
Database
Start Application
Send login request with username/password
request user data via username
return requested data or error
Return token or error message
On error, prompt user that credential is invalid
request post data
return post data
On Successful Login, redirect to main dashboard

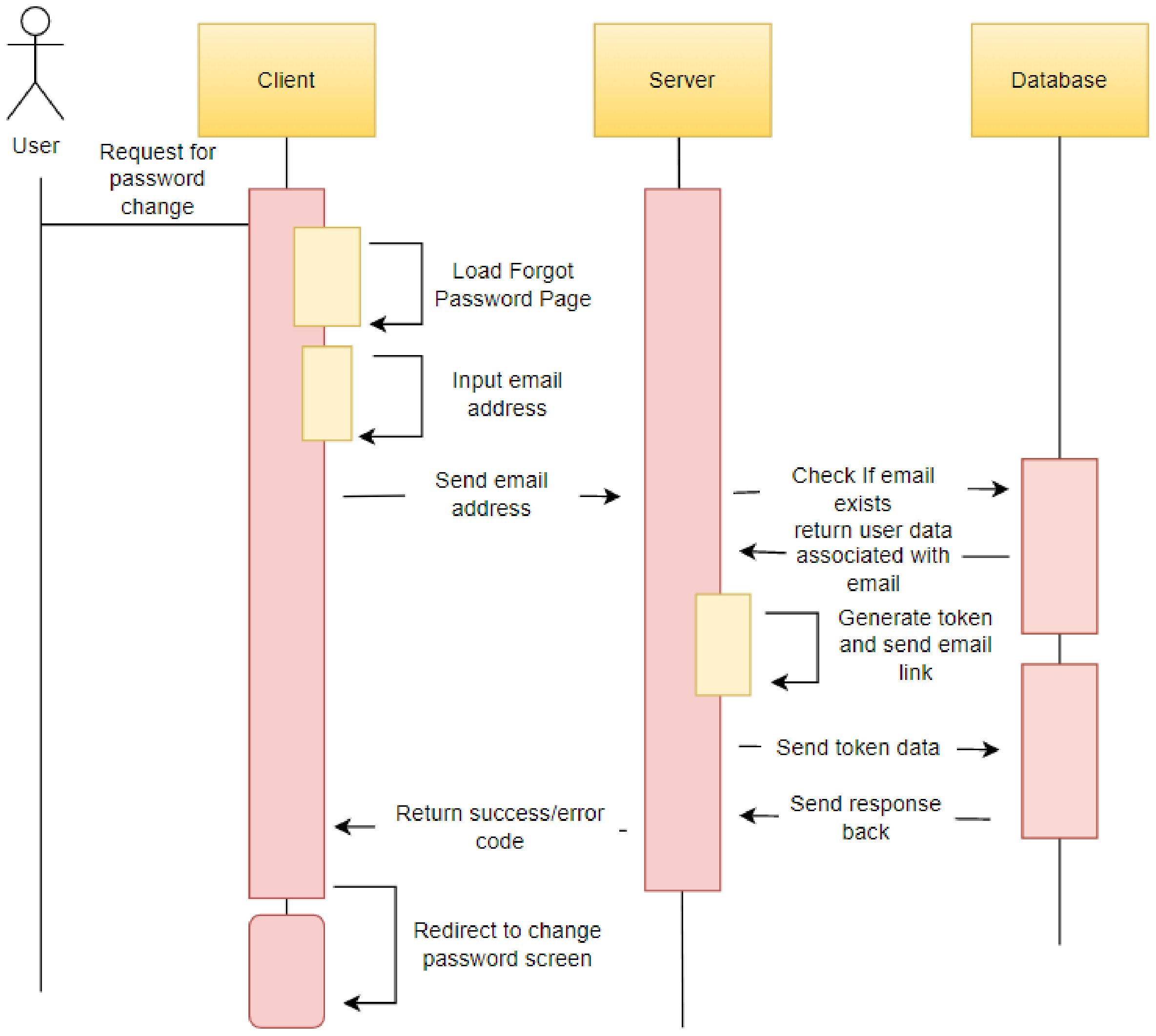
2. Sequence of events when a user creates a post


User
Client
Server
Database
Creating Post
Request list of user's recipes
Return success/error code
Retrieve Recipe Data for given username
Send data back to server
Load create post page
Create/Delete Post
Return success/error code
Push/Delete Post to Database
Return success/failure
Redirect to all producer posts when post is made
Retrieve all posts
Return data/error code
Get post data of producer
Send data back to server

3. Sequence for creating an event

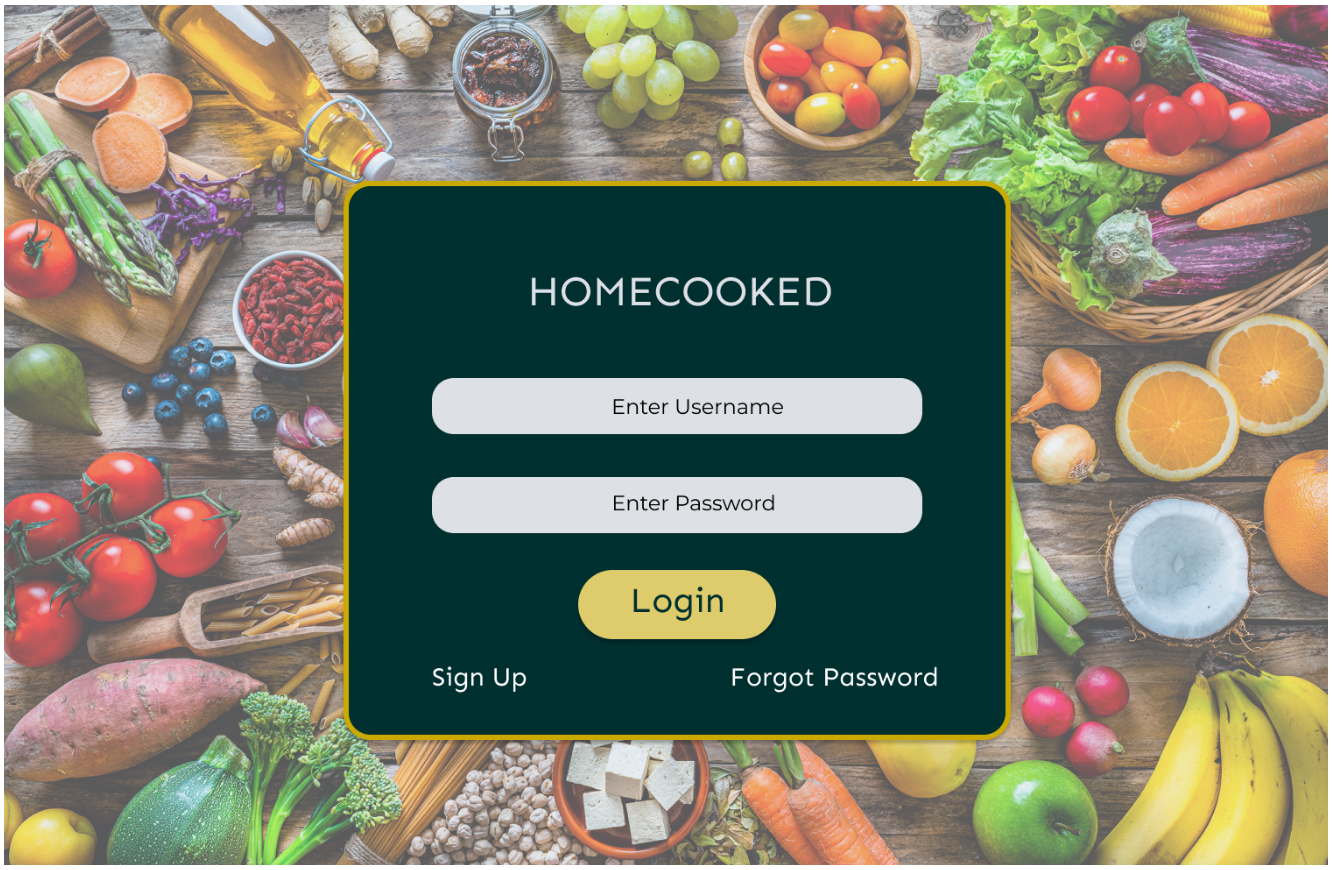


4. Sequence to reset password


User
Client
Server
Database
Request for password change
Load Forgot Password Page
Input email address
Send email address
Check If email exists
return user data associated with email
Generate token and send email link
Send token data
Send response back
Return success/error code
Redirect to change password screen

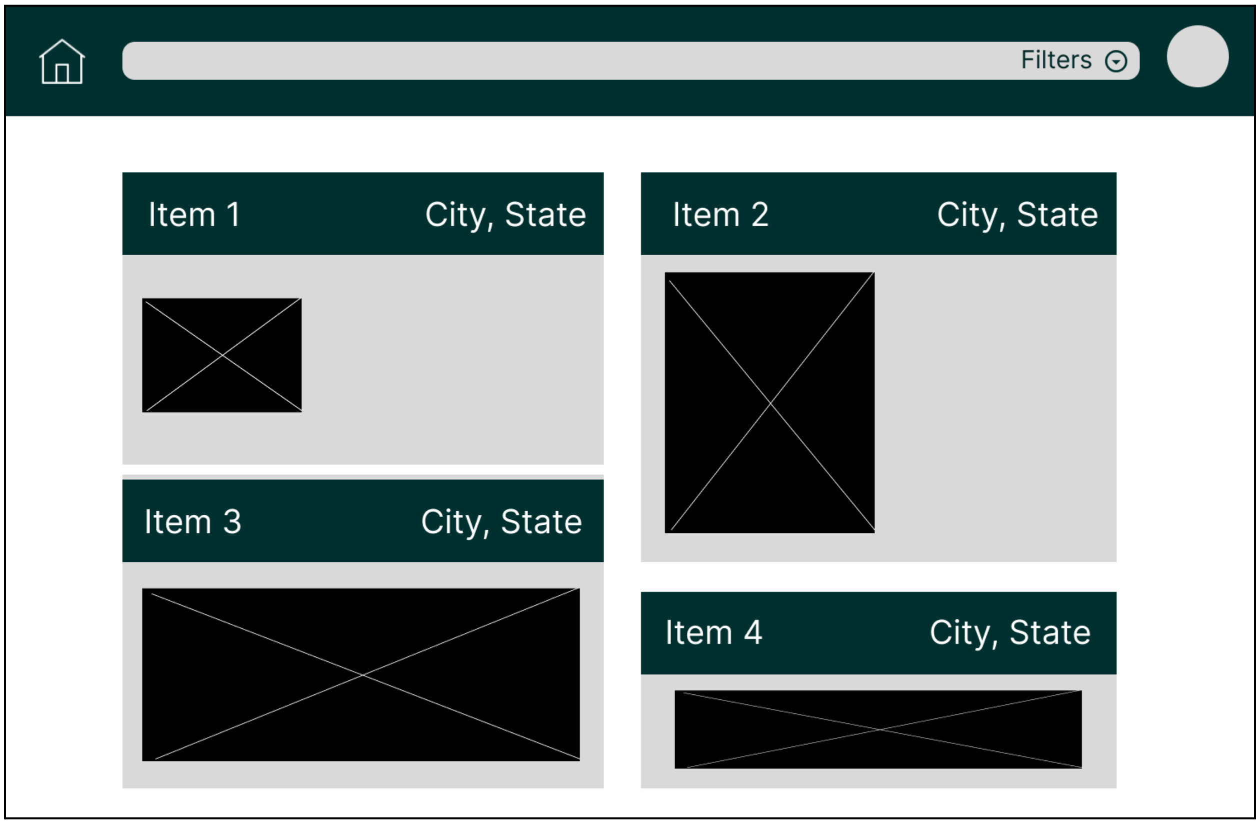
UI Mockups

- Login Screen



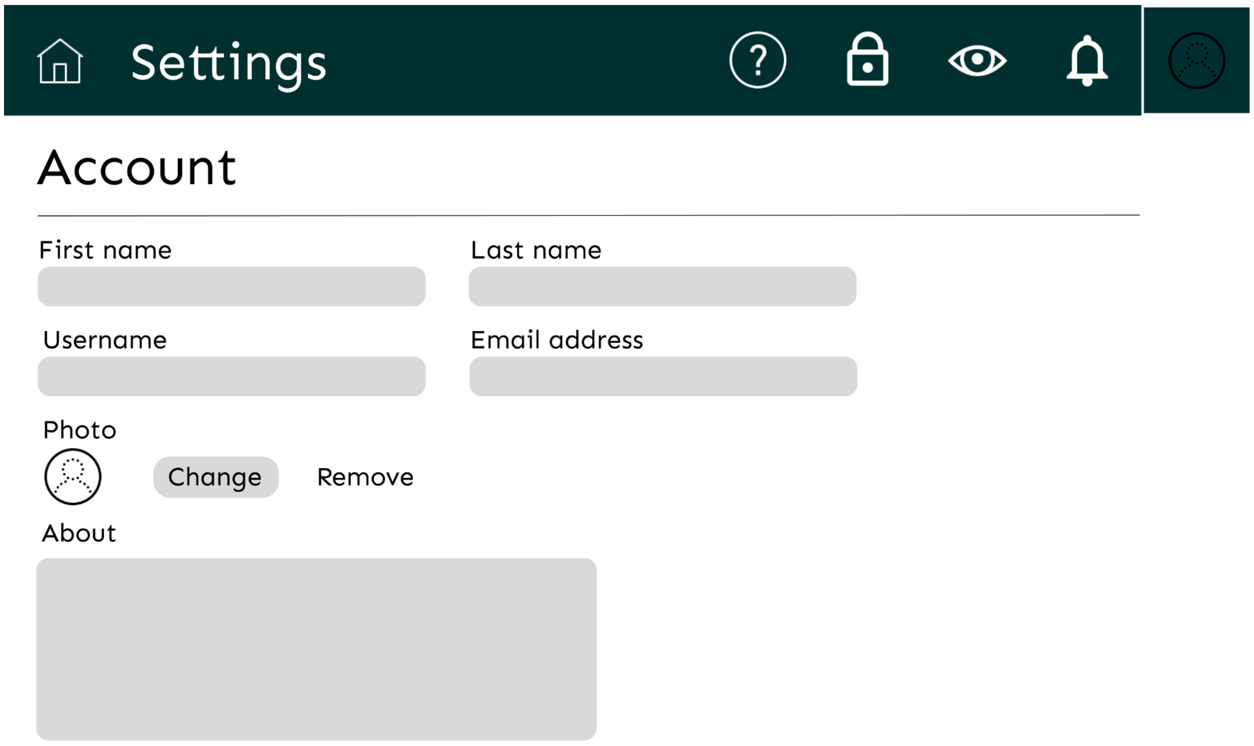
This is the login page. Users will input their username and password. On success, they will be redirected to the consumer dashboard, which they can change to producer should they choose to do so.

- Consumer Dashboard



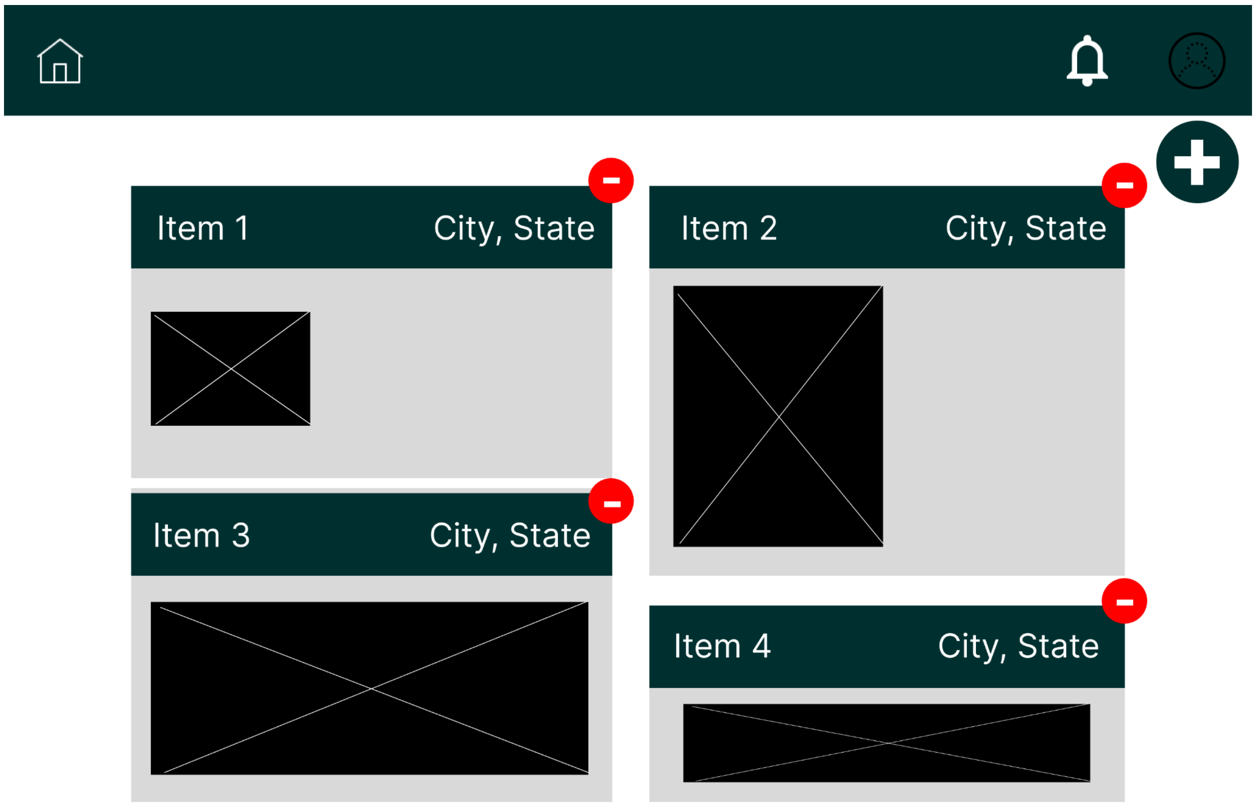
This is the Consumer page. This is where consumers will be able to see items that they may wish to obtain. Once they find an item, they can click on the item, which will bring them to a chat UI with the user

- User Settings Page



This is the user Settings page. Users will be able to not only view their information, but also be able to update certain pieces of information on their page such as profile picture and their about sections.

- Producer Dashboard



This is the Producer's dashboard. From here, producers can see all of their posts and they can delete a post by clicking the red button on the top right. If they want to create a new post. They can do so from the button on the top right of their screen.