Smart Kitchen Helper: System Design and Implementation

Amit Christian, Shukan Thakkar, Sagar Parmar

2024-08-05

# Introduction

The Smart Kitchen Helper project is an innovative solution designed to assist users in managing their household ingredients, finding and cooking recipes based on available ingredients, and tracking their cooking history. The system leverages modern web technologies, combining the flexibility of NoSQL with the robustness of SQL databases, to provide a seamless user experience. The system is targeted toward households and individuals who wish to optimize their cooking processes, reduce food waste, and explore new culinary possibilities.

The project is implemented using a combination of PostgreSQL for structured data management and MongoDB for handling unstructured data such as images and videos. The backend services are built using Node.js, while the frontend is developed with HTML, CSS, and JavaScript. The entire system will be deployed on AWS to ensure scalability, high availability, and security.

**Team Members:** - **Amit Christian (100937637)**: Project Leader, Frontend Development - **Sagar Parmar (100942558)**: Backend Development - **Shukan Thakkar (100944139)**: Data Management and Validation

This document provides a comprehensive overview of the Smart Kitchen Helper system, including the design, implementation, and deployment strategies.

# System Overview

A diagram of a computer

Description automatically generated

The Smart Kitchen Helper system is designed to tackle common challenges faced by households in managing their kitchen inventory and meal planning. By offering features like ingredient tracking, recipe recommendations based on available ingredients, and detailed cooking history, the system aims to simplify daily cooking routines and encourage users to make the most of their pantry.

## Core Features

1. **Ingredient Management**: Users can input their household ingredients into the system, including details such as quantity, unit of measurement, and expiration date. The system will notify users when ingredients are close to expiring, helping them prioritize their use and reduce food waste.
2. **Recipe Search and Suggestions**: Based on the ingredients available in a user’s household, the system can suggest recipes that can be made with those ingredients. This feature also allows users to discover new recipes and expand their culinary repertoire.
3. **Cooking History**: The system keeps track of the recipes that users have cooked, allowing them to revisit past dishes, review their ratings, and make notes for future reference. This history can also be used to refine recipe suggestions based on user preferences.
4. **User Ratings and Reviews**: After cooking a recipe, users can rate and review it, providing feedback that can be shared with the community. This helps other users decide which recipes to try based on collective experiences.

## Technical Stack

* **Frontend**: HTML, CSS, JavaScript
* **Backend**: Node.js
* **Databases**: PostgreSQL (SQL), MongoDB (NoSQL)
* **Deployment**: AWS (EC2, RDS, S3, Load Balancer, Autoscaling)

The system is currently in the local testing phase, with the backend services and databases set up using Python Flask for preliminary testing. The plan is to transition fully to Node.js before deploying the system on AWS.

# Frontend Design

The frontend of the Smart Kitchen Helper is designed to be intuitive and user-friendly, providing a smooth experience for users of all technical levels. The interface is clean, responsive, and organized in a way that makes navigation easy, even for first-time users.

## Key Pages and Features

1. **Index Page**: The index page serves as the main landing page, offering an overview of the system’s capabilities. It features quick links to key sections of the site, such as ingredient management, recipe search, and user profiles. The design emphasizes simplicity and ease of access.

* Index Page Screenshot
* A screenshot of a computer

  Description automatically generated
* The index page includes a navigation bar at the top, which remains consistent across all pages, providing easy access to other sections of the website. The footer contains links to the privacy policy, terms of service, and contact information.

1. **Login Page**: The login page is designed to be secure and straightforward, allowing users to authenticate their credentials quickly. The form is protected by SSL to ensure that all data transmitted is encrypted.

* Login Page Screenshot
* A screenshot of a computer

  Description automatically generated
* The login process includes validation checks to ensure that only valid credentials are accepted. If a user enters incorrect credentials, a helpful error message is displayed, guiding them on how to proceed.

1. **Signup Page**: The signup page allows new users to register for the system. It requires basic information such as username, email, and password. The form includes client-side and server-side validation to ensure data integrity.

* Signup Page Screenshot
* A screenshot of a computer

  Description automatically generated
* During signup, users are prompted to agree to the terms and conditions before their account is created. The system also checks for duplicate usernames and emails, ensuring that each user has a unique identity within the system.

1. **Homepage**: The homepage serves as the user’s dashboard, providing an overview of their current ingredients, recent recipes, and cooking history. It is the central hub from which users can access all the system’s features.

* Homepage Screenshot
* A screenshot of a computer

  Description automatically generated
* The homepage is personalized for each user, displaying relevant data such as expiring ingredients, recommended recipes, and recently cooked meals. This personalization helps users quickly find the information they need.

### Frontend Page Connections

The frontend pages are interconnected to provide a seamless user experience. For example:

* The **Index Page** links to the **Login** and **Signup** pages, guiding new users through the process of creating an account.
* The **Login Page** authenticates users and redirects them to the **Homepage** upon successful login.
* The **Signup Page** registers new users and redirects them to the **Login Page** to authenticate their new account.
* The **Homepage** serves as the central hub, providing access to ingredient management, recipe search, and cooking history.

These connections ensure that users can easily navigate the system and access the features they need with minimal friction.

# Backend Design

The backend of the Smart Kitchen Helper is developed using Node.js, a powerful and efficient runtime environment that allows for scalable and fast backend services. The backend handles all the business logic, data processing, and communication between the frontend and the databases.

## Backend Features

1. **User Management**: The backend manages user accounts, including registration, authentication, and role-based access control (RBAC). It ensures that each user has the appropriate level of access to the system’s features.
2. **Ingredient Management**: The backend provides APIs for managing household ingredients. Users can add new ingredients, update existing ones, and delete items that are no longer needed. The backend also handles expiration date tracking and sends notifications for expiring ingredients.
3. **Recipe Management**: The backend offers APIs for searching and managing recipes. It integrates with the frontend to provide users with recipe suggestions based on their available ingredients. The system also tracks cooking history and user ratings, allowing for personalized recipe recommendations.
4. **Data Validation**: To ensure data integrity, the backend includes strict validation rules for all inputs. This prevents invalid data from being stored in the databases, reducing the risk of errors and inconsistencies.

### Node.js API Endpoints

The backend exposes several API endpoints that the frontend uses to interact with the system. Some key endpoints include:

* **/api/auth/login**: Authenticates users and returns a JSON Web Token (JWT) for secure communication.
* **/api/auth/signup**: Registers new users and creates their accounts in the system.
* **/api/ingredients**: Handles CRUD operations for managing household ingredients.
* **/api/recipes**: Allows users to search for recipes, add new ones, and manage their cooking history.

These endpoints are designed to be RESTful, ensuring that they are stateless and can be easily integrated with other systems if needed.

# Database Design

The Smart Kitchen Helper uses a combination of PostgreSQL and MongoDB to manage its data. PostgreSQL is used for structured data, such as user information and ingredient details, while MongoDB handles unstructured data, such as images and videos associated with recipes.

## PostgreSQL (SQL)

### Key Tables

1. **Users**: This table stores user information, including their usernames, email addresses, hashed passwords, and roles within the system. Each user is assigned a unique ID that is used to link their data across other tables.
2. **Households**: The Households table stores information about each household, including its name and address. Each household is linked to one or more users, who are members of that household.
3. **Household\_Users**: This table tracks the membership of users in households. It includes fields for the household ID, user ID, and the user’s role within the household (e.g., Owner, Member).
4. **Ingredients**: The Ingredients table stores information about each ingredient, including its name and category ID. The category ID links to the Ingredient\_Categories table, which provides more detailed information about each category.
5. **Recipes**: This table stores metadata about each recipe, including its name, cuisine, preparation time, and system rating. The recipes are linked to ingredients through the Recipe\_Ingredients table.
6. **Recipe\_Ingredients**: This table associates ingredients with recipes, specifying the quantity and unit of measurement for each ingredient in a recipe.
7. **User\_Recipe\_History**: This table tracks the history of recipes cooked by each user, including the date and time of cooking.

### Data Validation

PostgreSQL uses several constraints to ensure data integrity:

* **Primary Keys**: Ensure that each record in a table is unique.
* **Foreign Keys**: Maintain referential integrity between related tables.
* **Unique Constraints**: Prevent duplicate entries in fields such as email addresses.
* **Check Constraints**: Enforce rules on data values, such as ensuring that ingredient quantities are non-negative.

## MongoDB (NoSQL)

### Key Collections

1. **Recipes**: The Recipes collection stores detailed information about each recipe, including images, preparation steps, and YouTube video URLs. The flexible schema of MongoDB allows for variations in how recipe data is structured.
2. **Ingredients**: The Ingredients collection manages images and detailed information for ingredients, such as nutritional information and associated images.
3. **Household\_Ingredient\_Usage**: This collection tracks the usage of ingredients within households, including historical usage data.
4. **Recipe\_Ratings**: The Recipe\_Ratings collection stores user ratings and reviews for recipes, allowing for community feedback and recipe improvement.
5. **User\_Preferences**: This collection stores user-specific preferences and settings, such as dietary restrictions and preferred cuisines.

### Schema Validation

MongoDB allows for schema validation using JSON Schema. This ensures that documents in a collection adhere to a defined structure, even though MongoDB is schema-less by default.

# AWS Deployment

The deployment of the Smart Kitchen Helper on AWS is designed to ensure high availability, scalability, and security. The following steps outline the process:

## AWS Infrastructure

### Architecture Overview

The system will be deployed in a multi-tier architecture, with separate layers for the frontend, backend, and databases. This approach provides better security, scalability, and maintainability.

#### Key Components

1. **VPC (Virtual Private Cloud)**: A dedicated VPC will be created to host all the components of the Smart Kitchen Helper. This provides network isolation and security for the resources.
2. **Subnets**: The VPC will be divided into multiple subnets, with separate subnets for the public-facing components (e.g., the frontend) and the private components (e.g., databases).
3. **EC2 Instances**: The backend services will be deployed on EC2 instances. These instances will be part of an auto-scaling group to ensure that the system can handle varying levels of traffic.
4. **RDS (Relational Database Service)**: PostgreSQL will be deployed using AWS RDS, which provides a managed database service that is highly available and automatically backed up.
5. **MongoDB Atlas**: MongoDB will be hosted on MongoDB Atlas, a cloud-hosted service that provides a fully managed database with automated scaling and backups.
6. **Load Balancer**: An Elastic Load Balancer will distribute incoming traffic across multiple EC2 instances, ensuring high availability and fault tolerance.
7. **Autoscaling**: Autoscaling will be configured to automatically adjust the number of EC2 instances based on traffic levels. This ensures that the system can scale up during peak usage and scale down to save costs during low traffic periods.
8. **S3 (Simple Storage Service)**: S3 will be used to store static assets, such as images and videos, as well as backups of the databases.

### Deployment Steps

1. **Create a VPC**: Begin by creating a VPC and dividing it into public and private subnets. Assign an Internet Gateway to the VPC to allow internet access to the public subnets.
2. **Launch EC2 Instances**: Launch EC2 instances in the public subnets for the backend services. Install Node.js and other necessary dependencies on these instances.
3. **Configure RDS**: Set up PostgreSQL using RDS in a private subnet. Ensure that the security group allows connections only from the EC2 instances in the public subnets.
4. **Set Up MongoDB Atlas**: Create a MongoDB cluster using MongoDB Atlas. Configure network access to allow connections from the EC2 instances.
5. **Deploy the Backend**: Deploy the backend Node.js application on the EC2 instances. Configure the application to connect to the PostgreSQL database on RDS and the MongoDB Atlas cluster.
6. **Configure Load Balancer**: Set up an Elastic Load Balancer to distribute traffic across the EC2 instances. Configure health checks to ensure that traffic is only routed to healthy instances.
7. **Set Up Autoscaling**: Configure autoscaling policies based on CPU usage or network traffic to automatically adjust the number of EC2 instances as needed.
8. **Deploy the Frontend**: Deploy the frontend application to S3 and configure it to serve static content. Use CloudFront to distribute the frontend globally with low latency.
9. **Set Up Monitoring and Alerts**: Use AWS CloudWatch to monitor the health and performance of the system. Set up alerts to notify the team of any issues, such as high CPU usage or failed health checks.

## System Capabilities and Limitations

### What the System Can Do

1. **Manage Users and Roles**: The system supports role-based access control, ensuring that users have appropriate permissions based on their roles.
2. **Track Ingredients**: Users can manage their household ingredients, including adding new items, updating quantities, and tracking expiration dates.
3. **Suggest Recipes**: The system can suggest recipes based on the ingredients available in a user’s household. This helps users make the most of their pantry and reduce food waste.
4. **Log Cooking History**: The system keeps track of recipes that users have cooked, allowing them to review their cooking history and revisit past meals.
5. **Handle Images and Videos**: The system stores and manages images and videos associated with recipes, enhancing the user experience with visual content.

### What the System Cannot Do

1. **Auto-Approve Content**: The system requires manual approval for new recipes and user reviews. This ensures that all content meets quality standards.
2. **Process Invalid Data**: The system cannot handle invalid or corrupted data. Input validation is enforced to prevent such data from being stored in the databases.
3. **Override RBAC**: Users cannot perform actions outside their assigned roles, ensuring that sensitive operations are restricted.

### Detailed Data Validations

1. **Email Validation**: The system checks that all email addresses are in a valid format before storing them in the database.
2. **Password Hashing**: Passwords are hashed before storage to ensure that they cannot be easily compromised in the event of a data breach.
3. **Quantity Validation**: The system ensures that all quantities are non-negative, preventing errors in ingredient management.
4. **Date Validation**: Expiration dates are checked to ensure they are in the future, preventing the use of expired ingredients.
5. **Role Validation**: The system enforces role-based access control, ensuring that users can only perform actions that are appropriate for their role.

### User Roles and Permissions

1. **Guest**: Limited access, primarily for exploring public content and viewing recipes.
2. **Member**: Standard access, including the ability to manage household ingredients and submit ratings and reviews.
3. **Owner**: Elevated access, allowing for full household management, including adding and removing members and approving content.
4. **Moderator**: Global access to moderate content and manage user-generated content across the system.
5. **Administrator**: Full system access, including user management, role assignment, and system configuration.
6. **Content Creator**: Specialized access for submitting and managing recipes.
7. **Viewer**: Read-only access, primarily for viewing content without making changes.

# Conclusion

The Smart Kitchen Helper is a comprehensive system designed to streamline household kitchen management. By leveraging modern web technologies and cloud infrastructure, the system offers robust features, including ingredient tracking, recipe suggestions, and user-generated content management. The project is currently in the local testing phase, with plans to deploy it on AWS to ensure scalability and high availability.

The implementation of the system on AWS will follow a well-defined architecture, ensuring that all components are secure, scalable, and fault-tolerant. The use of both PostgreSQL and MongoDB allows the system to handle both structured and unstructured data efficiently, providing a rich user experience.