Smart Recipe & Meal Planner

Design Documentation

Microservices and Containers KM00DT91-3005

Oulu University of Applied Sciences

16 March 2025

Nico Tukiainen

1. Introduction

Project Overview

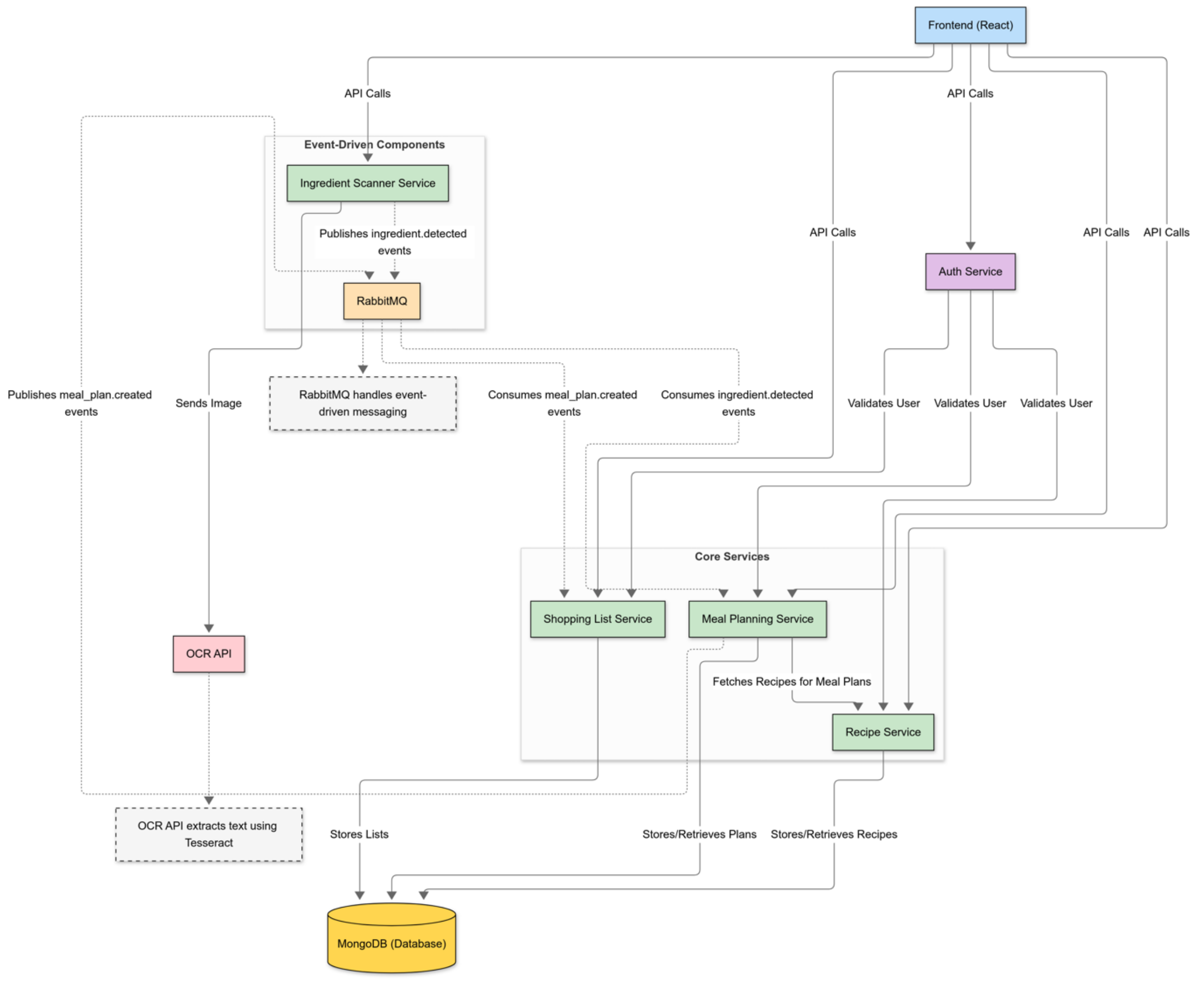
The Smart Recipe & Meal Planner is a microservice-based distributed application designed to help users generate meal plans based on available ingredients, dietary preferences, and nutritional goals. The system integrates multiple microservices to handle recipes, ingredient recognition via OCR, meal planning, and shopping list generation. Objectives below:

- Allow users to scan ingredients using OCR and generate a list.
- Provide personalized meal suggestions based on available ingredients and dietary restrictions.
- Generate a shopping list for missing ingredients.
- Ensure scalability and modularity by using microservices.

2. Architecture Overview

System Architecture Diagram

The following diagram illustrates the architecture and interaction between microservices:



Microservices breakdown

The system consists of the following microservices:

1. Recipe Service

- Stores and manages a collection of recipes.
- Provides APIs to retrieve recipe details based on ingredients and dietary preferences.
- Uses MongoDB as the primary database.
- Responds to Frontend requests for recipe retrieval and filtering.
- Responds to Meal Planning Service requests for recipe data when creating meal plans.

2. Ingredient Scanner Service

- Uses OCR (Optical Character Recognition) to detect ingredients from images.
- Sends images to the OCR API for text extraction.
- Extracts text and maps detected ingredients to known food items.
- Publishes detected ingredient events to RabbitMQ for asynchronous communication.
- Notifies the Meal Planning Service when new ingredients are identified.

3. Meal Planning Service

- Listens for ingredient detection events from RabbitMQ.
- Processes user-selected ingredients and dietary preferences.
- Fetches recipe details from the Recipe Service to generate meal plans.
- Publishes meal plan creation events to RabbitMQ when a meal plan is created.
- Notifies the Shopping List Service when a meal plan is created.

4. Shopping List Service

- Listens for meal plan creation events from RabbitMQ.
- Identifies missing ingredients from a meal plan.
- Generates a structured shopping list and stores it in MongoDB.
- Provides APIs for retrieving, updating, and managing shopping lists.

5. Authentication Service (Auth Service)

- Handles user authentication and profile management.
- Issues and validates JWT tokens for secure API access.
- Ensures only authorized users can interact with Recipe, Meal Planning, and Shopping List Services.

6. OCR API

- Processes images received from the Ingredient Scanner Service**.**

- Uses Tesseract OCR to extract text.
- Returns extracted text data to Ingredient Scanner Service.

7. RabbitMQ

- Handles event-driven messaging for asynchronous communication between services.
- Manages message queues and exchanges for reliable message delivery.
- Enables decoupled communication between microservices.

8. Frontend Service (React + Tailwind CSS)

- Provides an intuitive user interface.
- Allows users to interact with the backend services.
- Fetches data via REST APIs from microservices.
- Manages user interactions, meal planning, and shopping list visualization.

3. Technology Stack

Component	Technology
Frontend	React, Tailwind CSS, TypeScript
Backend	FastAPI/Flask (Python)
Database	MongoDB
OCR API	Tesseract OCR
Containerization	Docker
Service Communication	REST APIs / RabbitMQ (Event-driven)
Deployment	Docker Compose

---

4. Data Flow & Communication

User Flow:

1. User uploads an image of available ingredients.
2. The Ingredient Scanner Service extracts text and identifies ingredients.
3. The Meal Planning Service suggests recipes based on the available ingredients.
4. If missing ingredients are found, the Shopping List Generator creates a shopping list.
5. The frontend displays recommendations and the shopping list.

Inter-service Communication:

- Microservices communicate using REST APIs.
- RabbitMQ is used for asynchronous event-driven communication.
- Data exchange follows JSON format.

5. Deployment Strategy

- Each microservice is containerized using Docker.
- Services communicate within an isolated network using Docker Compose.
- Environment variables are used for configuration management.

Example Docker Compose Configuration:

```
version: '3.8'
services:
  recipe-db:
    image: mongo
    container_name: recipe-db
    ports:
      - "27017:27017"

  ingredient-scanner:
    build: ./ingredient-scanner
    container_name: ingredient-scanner
    depends_on:
      - recipe-db
    ports:
      - "5001:5001"

  meal-planner:
    build: ./meal-planner
    container_name: meal-planner
    depends_on:
      - recipe-db
    ports:
      - "5002:5002"
```

---

6. API Endpoints

Authentication Service

- **POST** `/auth/register` - Register a new user
- **POST** `/auth/login` - Login and get an access token
- **POST** `/auth/refresh` - Refresh access token
- **POST** `/auth/logout` - Logout and invalidate tokens
- **GET** `/auth/profile` - Get user profile
- **PUT** `/auth/profile` - Update user profile

Recipe Service

- **GET** `/recipes` - List all recipes with optional filtering
- **GET** `/recipes/{id}` - Get a specific recipe

- **POST** `/recipes` - Create a new recipe
- **PUT** `/recipes/{id}` - Update a recipe
- **DELETE** `/recipes/{id}` - Delete a recipe

Ingredient Scanner Service

- **POST** `/scan` - Upload an image for ingredient detection
    - *Publishes detected ingredients to RabbitMQ (`ingredient.detected`)*
- **POST** `/manual-input` - Manually input ingredients
    - *Publishes manually entered ingredients to RabbitMQ (`ingredient.detected`)*

Meal Planning Service

- **POST** `/meal-plans` - Create a meal plan based on ingredients and preferences
    - *Fetches recipes from Recipe Service*
    - *Publishes meal plan creation event to RabbitMQ (`meal_plan.created`)*
- **GET** `/meal-plans` - List all meal plans
- **GET** `/meal-plans/{id}` - Get a specific meal plan
- **DELETE** `/meal-plans/{id}` - Delete a meal plan

Shopping List Service

- **POST** `/shopping-lists` - Create a shopping list from a meal plan
    - *Can be triggered by `meal_plan.created` events from RabbitMQ*
- **GET** `/shopping-lists` - List all shopping lists
- **GET** `/shopping-lists/{id}` - Get a specific shopping list
- **PUT** `/shopping-lists/{id}/items/{ingredient}/check` - Check/uncheck an item
- **DELETE** `/shopping-lists/{id}` - Delete a shopping list

7. Database Schema

User Collection

```
{
  "_id": "ObjectId(567890)",
  "username": "johndoe",
  "email": "john@example.com",
  "preferences": {
    "dietary_preferences": ["Vegetarian", "Low-Carb"],
    "favorite_cuisines": ["Italian", "Asian"]
  },
  "created_at": "2025-03-10T12:00:00Z"
}
```

Recipe Collection

```
{
  "_id": "ObjectId(123456)",
  "title": "Spaghetti Carbonara",
  "ingredients": ["Spaghetti", "Eggs", "Pancetta", "Parmesan cheese",
"Black pepper", "Salt"],
  "prep_time_minutes": 10,
  "cook_time_minutes": 15,
  "servings": 4,
  "tags": ["Italian", "Pasta", "Quick"],
  "created_at": "2025-03-10T12:00:00Z"
}
```

---

8. Conclusion

This document outlines the design and implementation plan for the Smart Recipe & Meal Planner microservices application. It ensures modularity, scalability, and maintainability while following the Domain-Driven Design (DDD) principles.

By using Docker-based deployment, REST API communication, and a well-structured microservices architecture, the system effectively supports real-world applications in meal planning and ingredient recognition.