We Are Cooking

Contents

1 Executive Summary

1.1 Project Overview

"We Are Cooking" is a social media platform targeting the culinary community. The platform enables users to share recipe ideas, watch and post short-format videos (reels), manage own virtual fridge or cookbook with saved recipes, and receive personalized food recommendations. It also supports interactions between users through reviews.

1.2 Objectives

- Provide a platform for sharing and discovering recipes.
- Offer recipe recommendations based on user preferences.
- Create space for cooking enthusiasts, driven by the community itself.

1.3 Key Deliverables

- Native interface
- Modular microservices
- User authentication and authorization via Keycloak

1.4 Target Audience and Scope

Platform targets cooking enthusiasts of all experience levels, it also covers recipe sharing, video sharing, and reviews.

2 Introduction

2.1 Background and Motivation

We Are Cooking is designed to be an interactive and engaging social media platform entirely dedicated to the world of recipes, cooking, and culinary creativity. The app aims to bring together food enthusiasts, home cooks, and professional chefs in a vibrant community where users can share, discover, and discuss their favorite dishes.

2.2 Problem Statement

Most existing cooking apps focus on static recipe sharing. There's a lack of a dynamic, socially interactive environment that integrates modern digital media (reels, recommendations) and personalized user features like cookbooks and fridges.

2.3 Relevance to Electronic Media in Business and Commerce

By combining food content with social media features, the app creates opportunities for user engagement, content monetization, and collaborations with brands, influencers, and food retailers.

2.4 Document Structure

The document is organized into sections that follow the software development lifecycle: from requirements to deployment, concluding with an evaluation and business impact analysis.

3 Literature Overview

3.1 Overview of Similar Applications

There are various cooking and recipe applications available in the market today, each offering unique features but also presenting notable limitations. Below are some popular platforms:

3.1.1 Tasty

Tasty by BuzzFeed focuses on delivering short, engaging recipe videos. Its strength lies in its user-friendly interface and rich multimedia content. However, Tasty lacks strong community features or personalization beyond simple filters.

3.1.2 Yummly

Yummly uses user preferences and dietary needs to provide recipe recommendations. It includes features such as smart shopping lists and voice-guided cooking. Yet, it offers minimal interaction between users, and there is no support for user-generated video content.

3.1.3 Cookpad

Cookpad emphasizes community-generated content. It allows users to post their own recipes and comment on others. While it promotes community, it doesn't support reels or ingredient-based recommendations, which are increasingly important to modern users.

3.1.4 Whisk

Whisk provides excellent recipe organization and grocery planning tools. It aggregates content from various sources, but it lacks features for video sharing and real-time user interaction.

Comparison: *We Are Cooking* integrates multiple features lacking in other apps:

- Social-media-like reels for short, engaging cooking content
- A virtual fridge and smart recipe recommendations
- Modular microservices for performance and scalability
- A strong community focus with user reviews and sharing

3.2 Industry Trends in E-Commerce and Digital Media

The intersection of e-commerce and digital media is characterized by several major trends relevant to our application:

- **Short-form Video Content:** Platforms like Instagram, TikTok, and YouTube Shorts have normalized reels as the preferred format for content consumption.
- **Personalization:** Users expect personalized experiences, including recommendations based on preferences, history, and real-time inputs.
- **Community and Social Integration:** Modern apps are increasingly integrating social features to boost engagement, virality, and user retention.
- **Mobile-first Design:** With the majority of users accessing services via smartphones, mobile-first design is essential for user adoption.
- **Composable and Microservice Architectures:** These allow scalable and flexible development, enabling rapid feature deployment and system resilience.

We Are Cooking aligns with these trends by offering short-form video features, personalized food suggestions, a mobile-first UI, and a microservice-based backend architecture.

3.3 Research Gaps Addressed

Most existing applications cater either to static recipe sharing or fragmented use-cases like shopping list management or food tracking. Few combine all aspects into a cohesive, community-driven platform. Gaps addressed by *We Are Cooking* include:

- **Lack of ingredient-driven recommendations:** Our virtual fridge and recommender system provide a highly personalized experience.
- **Absence of real-time social media features:** The reels functionality introduces a TikTok-like experience for cooking enthusiasts.
- **Fragmented user experience:** We aim to unify recipes, content creation, and ingredient management under one platform.
- **Limited interaction models in traditional apps:** Features like reviews, profile customization, and real-time content consumption foster deeper user engagement.

By filling these gaps, *We Are Cooking* provides a fresh and modern approach to culinary media, bridging utility with social engagement.

4 System Requirements and Analysis

4.1 Functional Requirements

- The user must be able to sign-up.
- The user must be able to log into the system.
- The user must be able to search for the recipes.
- The user must be able to add recipes.
- The user must be able to delete, or edit his recipes.
- The user must be able to save recipes to his cookbook.
- The user must be able to remove recipes from his cookbook.
- The user must be able to review a recipe.
- The user must be able to get some recommendations.
- The user must be able to see his profile.
- The user must be able to edit the data on his profile.
- The user must be able to add ingredient categories to his restrictions.
- The user should get notifications.
- The user should be able to add ingredients to his virtual shopping list.
- The user should be able to add reels.
- The user should be able to watch reels.
- The user should be able to share recipes.
- The user should be able to add ingredients to his virtual fridge.
- The user should be able to get recipes based on the ingredients in his virtual fridge.

4.2 Non-functional Requirements

- The main database schema should be maintained through the Liquibase versioning system.
- The APIs should be accessed through the Gateway application.
- The user data should be accessible only through the Keycloak Authorization Server.

4.3 Use Case Diagrams

The project is targeted at users and administrators of the system, so there is only one use cases diagram shown in Figure 1. This use cases diagram uses colors to point out the importance of the features.

Figure 1: The use case diagram showing the features

5 System Design

The system was designed as a composition of microservices. Those microservices are connected using docker-compose.

5.1 System Architecture

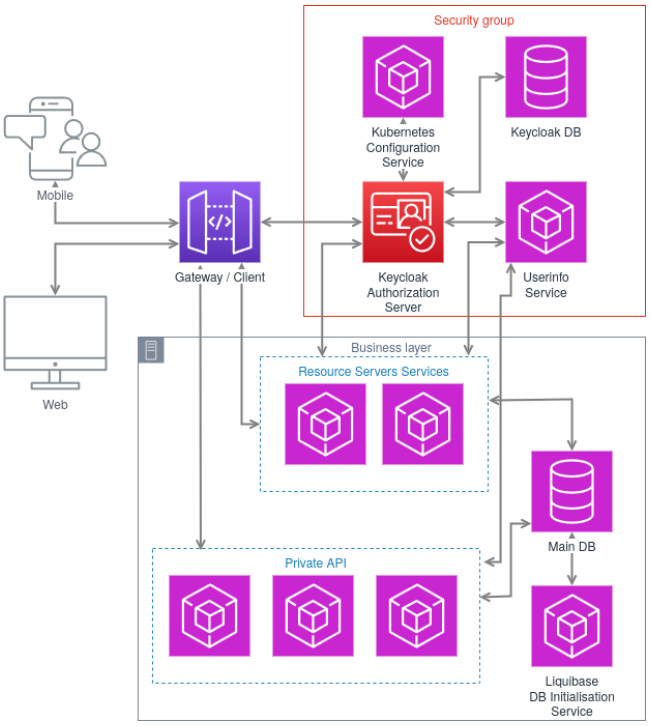

Figure 2: System architecture diagram

5.2 Database Design

There are two databases in the project: Keycloak database and Main database. The Keycloak database is used by the Keycloak Authorization Server to store the information about registered users. The Main database is used to store the data regarding the business logic and the content introduced by the users.

5.3 User Interface Design

The user interface (UI) of *We Are Cooking* was designed with a mobile-first approach, ensuring a seamless experience across both Android and iOS devices. The front-end is implemented using the Flutter framework, which allows for a consistent and responsive design across platforms.

The application focuses on simplicity, accessibility, and visual engagement. The core screens include:

- **Home Screen:** Displays trending recipes, user-recommended content, and featured reels. It uses a scrollable feed layout similar to modern social media apps for intuitive navigation.
- **Recipe Detail Page:** Offers comprehensive information about a selected recipe, including ingredients, description, estimated cooking time, and user reviews. Users can save, share, or comment on the recipe.
- **Reels Section:** A vertically scrollable interface for short-format cooking videos. This section mimics popular platforms like TikTok or Instagram Reels to increase user engagement and content virality.
- **Cookbook and Fridge:** The virtual cookbook allows users to organize and manage saved recipes. The fridge feature displays a list of ingredients currently "owned" by the user, with an option to receive matching recipe suggestions.
- **User Profile:** Shows user details, uploaded content, saved recipes, and interaction history. It also offers an interface for managing dietary preferences and ingredient restrictions.
- **Authentication Flow:** Login, registration, and logout pages are integrated with Keycloak, using a customized theme to maintain visual consistency with the main app.

The overall design prioritizes user-friendliness, with minimalistic icons, bold call-to-action buttons, and a clear typography hierarchy. The color scheme is warm and food-inspired, enhancing the visual appeal while maintaining

good readability. Future iterations may include accessibility features such as voice navigation, high-contrast themes, and support for screen readers.

5.4 Security Considerations

Application is communicating with outside authorization server in new and secure OIDC model. Main security point is the gateway API which authorizes every call to API. We used simple API keys for development purposes and didn't implement minimal password complexity.

6 Technology Stack

6.1 Programming Languages

Project microservices were written in various programming languages.

- *C#*,
- Python,
- Java,
- Rust,
- Dart

6.2 Frameworks and Libraries

- Flutter
- .NET 8
- Spring Boot
  - Spring Cloud
  - KeyCloak 2.0
  - Cloud Gateway
- Actix Web
- Scikit Learn
- FastAPI

6.3 Database Management System

The Main database was managed by the Liquibase system. The Keycloak database was managed by the Keycloak Authorization Server. The databases were deployed using Docker Compose.

6.4 Development Tools and Platforms

- Visual Studio 2022
- Visual Studio Code
- DBeaver
- Docker
- Redis
- Liquibase
- PostgreSQL

7 Implementation

7.1 Module-wise Breakdown

7.1.1 Keycloak

Provides cutting edge, powerful and more developer friendly authorization server where we used it for OIDC standard. Operating this server is mainly trough graphical interface which speeds up the process and allows us to even consider OIDC as an option instead of resorting to simple JWTs.

7.2 Gateway

Gateway is service which provides routing and security features for entire app. It is client service which secures endpoints and routes to services laying behind it and to keycloak for login/registration/logout pages. It also decodes OIDC tokens for easier access of necessary user data.

7.2.1 Frontend

The front-end implemented in Flutter.

7.2.2 Recipes API

The recipes API was implemented using the .NET 8 platform with C# programming language. The scope of the Recipes API is as follows:

- Recipe searching
- Recipe addition/edition/removal
- Recipe reviews
- Ingredient searching
- Ingredient categories searching
- User cookbooks
- User fridge
- User restricted categories

Not every implemented feature is being actively used by the front-end application.

7.2.3 Recommender API

The Recommender API is a core component of the *We Are Cooking* application, designed to provide personalized recipe suggestions to users based on their preferences, restrictions, and available ingredients.

This microservice was developed using Python and the FastAPI framework for its simplicity, speed, and ability to handle asynchronous operations efficiently. The recommendation logic leverages a hybrid approach that combines both content-based filtering and collaborative filtering techniques.

Key Features of the Recommender API:

- **Ingredient-Based Recommendations:** Users receive recipe suggestions based on the contents of their virtual fridge, ensuring minimal food waste and personalized cooking options.
- **User Preferences:** Recommendations are filtered by dietary restrictions (e.g., vegetarian, gluten-free) and ingredient categories added by users in their profile.
- **Behavioral Learning:** As users interact with recipes (saving, reviewing, viewing), the system improves suggestions using collaborative filtering, identifying trends in user behavior and preferences.
- **Real-Time Suggestions:** The API is designed to respond quickly and integrate seamlessly with the front-end, offering real-time updates as user preferences or fridge contents change.

The Recommender API is stateless and integrates with other services (such as the user profile service and the recipes API) via secure endpoints routed through the Gateway. In a production scenario, this service could be enhanced using machine learning frameworks such as Scikit-learn or TensorFlow for more advanced recommendation models.

While the current model uses predefined scoring logic, the architecture allows for easy integration of more complex algorithms or third-party ML models in the future.

7.2.4 Reels API

The Reels API microservice was written using the Rust programming language as well as the Actix Web framework for it's speed and reliability. This microservice handles user input related to video processing. Authorised users can interact with the proper endpoints to upload and receive videos.

7.2.5 User Info API

The Userinfo service is providing necessary user information, generates profiles and is main service for communication with authorization server. It is written in spring framework in java.

7.3 Key Algorithms and Functions

The recommender system in *We Are Cooking* is a hybrid solution composed of multiple algorithms, each tailored to a specific recommendation strategy. The system is modular and extensible, supporting various types of user interactions such as ratings, saves, and fridge ingredients. The primary algorithms used are as follows:

1. Popularity-Based Recommender

This algorithm ranks recipes based on a Bayesian average score, which avoids overrating recipes with few reviews. It enhances recommendations by:

- Adding a **recency boost** for trending recipes in the last N days.
- Including **cookbook saves** as an engagement metric.
- Optionally filtering by cuisine type.

This is particularly effective as a cold-start or fallback strategy when no user-specific data is available.

2. Content-Based Recommender

This model builds a recipe-to-recipe similarity matrix using ingredient data with a TF-IDF-style weighting. If a user has rated recipes, the model recommends new ones that are most similar to their highly-rated items. It works well for users with limited history and provides explainable results based on shared ingredients.

3. Collaborative Filtering Models

Multiple collaborative filtering approaches are supported:

- **User-Based / Item-Based Cosine Similarity:** Constructs a user-recipe interaction matrix and recommends recipes similar to what the user has rated or saved.
- **SVD Recommender:** A matrix factorization model using the `surprise` library. It learns latent factors to predict how a user might rate unrated recipes.
- **ALS (Alternating Least Squares):** A scalable implicit feedback model using the `implicit` library. It works with interactions such as views or saves, and is well-suited for large sparse datasets.

4. Fridge-Based Recommender

This model recommends recipes based on the ingredients a user has in their virtual fridge. Features include:

- Ingredient importance (core vs. optional).
- Substitution support (e.g., butter can be replaced with oil).
- Respecting dietary restrictions (e.g., vegan, gluten-free).
- A bonus for recipes that require minimal shopping effort (≤ 2 missing ingredients).

5. Hybrid Recommender

The final recommendation list is produced using a weighted voting system that aggregates the outputs from multiple recommenders. The weights are configurable and allow tuning based on user engagement or domain-specific insights. This hybrid approach ensures balanced recommendations that are personalized, fresh, and diverse.

7.4 Integration Strategy

The recommender system is encapsulated in a dedicated microservice that communicates with other services via RESTful APIs. Key elements of the integration strategy include:

- **Input Data Sources:** The Recommender API consumes data from the Recipes API (for metadata), the User Info service (for preferences), and the Review/Cookbook services (for user activity).
- **Stateless API Design:** Each API call generates recommendations on demand, ensuring real-time responsiveness without storing session-specific data.
- **Gateway Routing:** All external calls are routed through the centralized Gateway service, which handles authentication, authorization, and load balancing.
- **Asynchronous Processing (optional):** For scalability, the system can be extended with background jobs that precompute popular or trending recipes.
- **Model Modularity:** New models can be plugged into the hybrid framework without modifying the overall architecture, supporting future growth and experimentation.

This approach ensures that the recommendation logic is both scalable and maintainable, while also being adaptable to various user contexts and data availability scenarios.

8 Testing and Evaluation

8.1 Testing Methodologies

Manual testing only, pointing out edge cases. If not tested, the branch with the feature should not be pulled into the main (or develop) branch.

8.2 Test Cases and Results

Tested manually, some things are working right.

8.3 Bug Tracking and Resolution

If during manual testing a bug occurred but was not fixed immediately, it was placed on Trello as a task.

8.4 Performance Evaluation

The performance was not evaluated.

9 Deployment

9.1 Deployment Architecture

Current deployment architecture comprises of running docker-compose files locally.

9.2 Hosting Platforms and Environment

Docker containers running Linux system. The docker containers are connected together through common networks.

9.3 Version Control and Release Management

The version control tool used is Git, the platform used for version control is GitHub.

10 Business Application and Impact

10.1 Application Use Cases in Commerce

The application is aimed at a cooking niche and would need to be widely promoted to compete with other, more general, social media applications. In the current state, the impact would be near zero, as the features implemented are not enough for the end user to see We Are Cooking as a valid alternative.

10.2 Benefits to End Users and Businesses

10.3 Cost Analysis and ROI

Cost analysis is a complex issue with multiple factors that must be taken into account.

- Payment, estimating every member (5 members) of the team spent 60 hours and were payed 60PLN/h, it gives us 18 000 PLN
- Social security and insurance of members (19.64%) is estimated for 3 535 PLN
- Total direct costs (Payment + insurance) equal 21 535 PLN
- Indirect costs (around 10%) equal 2 154 PLN
- Total cost equals 23 689 PLN
- Profit equals 0 PLN
- VAT TAX (23%) equals 5 448 PLN

10.4 Potential for Scalability

The system was designed as a set of microservices, and if the project were to grow, the system could be scaled horizontally or vertically. The gateway application could be duplicated with the addition of load balancers. Redis is a distributed caching system that already is used in multiple large systems, so it would be used more broadly throughout the project environment.

11 Challenges and Limitations

11.1 Technical Challenges

11.1.1 Docker host development

Even though Docker provides containerization, which offers independence from the host system, some features still needed to be developed on the host systems. This, combined with our preference for using different OSes (two Linux distributions, Windows, and Windows with WSL), consumed a significant amount of development time.

11.1.2 OIDC

We utilized many new and powerful technologies that are not easy to implement. A significant portion of our time had to be invested in R&D, mainly concerning Keycloak and OIDC. The entire authorization system is relatively new, and its documentation is limited. Many of our previous understandings, which formed the basis for our application, had to be redesigned.

11.2 Business Constraints

11.2.1 Short timeline

Due to limited time we couldnt develop as much as we wanted and our product is in its demo version. We had to focus on and most important features which are not as importance is not visible on a glance of an eye. Better visual representation or any bug fixes had to be cut short.

11.3 Lessons Learned

11.3.1 Cooperation

Every participant in a project of this size should be communicating actively, not once per issue encountered. Tasks should be descriptive. The back-end and front-end applications should be equally important.

11.3.2 Features

Quality counts more than quantity, while many data endpoints were implemented, there was no consumer for the API, in the end, many features had to be cut due to the lack of time to work on the front-end application. Not every feature that was checked as 'Must Have' was finally implemented, and one additional functionality was added throughout the project lifetime which was the Reels functionality, which it was decided, was a good idea, based on other social media applications.

11.3.3 Recipes

The recipes are implemented in a way that does not include the steps needed to complete the recipe, this is a design issue that could be corrected if the project were to be continued.

12 Appendices

12.1 Glossary of Terms

- User - the client of the system
- Administrator - person responsible for the system work
- Ingredient - cooking ingredient
- Recipe - set of ingredients with quantities, has a name and description
- Reel - posted video by a given user
- Cookbook - set of recipes saved by the user
- API - Application Programming Interface

12.2 Source Code Repository Link

The repository is available here: github

12.3 User Guide / Installation Manual

12.3.1 Early development project installation (Working)

- First step is to install Docker or Docker Desktop, specific for your OS type.
- Go to root directory of a project and enable command console
- Start infrastructure by running

```
            docker compose -f infrastructure/compose.dev.infra.yaml up -d
```

  There might be one time error while starting Keycloak, so to see if everything is fine, enter

```
            docker ps --all
```

  And see if if there is auth-keycloak service. If there isn't or it is but status says "Exited" it means u need to start again. Best with this command:

```
            docker compose -f infrastructure/compose.dev.infra.yaml up -d auth-keycloak
    auth-keycloak-config-cli
```

  After everything all two services should be running: auth-keycloak and auth-db

- Regenerate and copy secret for gateway in Keycloak. Go to Admin console for Keycloak at localhost:8080 (Login: admin, Password: admin). Click on dropdown list "Keycloak master" and select "Techstructure" realm. Go to Clients -¿ wearecooking-gateway -¿ Credentials -¿ Client Secret -¿ Regenerate. Click it and copy it to ProjectDirectory/services/gateway/src/main/resources/application.yaml into value field for "client-secret"
- Start main project database and it's versioning system

```
            docker compose -f infrastructure/compose.dev.db.yaml up -d
```

- Start main project services no less than a minute after keycloak service will build and start

```
            docker compose -f infrastructure/compose.dev.services.yaml up -d
```

  If there is a problem with one of the services it is advertised to run this command again

- As authorization server is hosted locally, it's needed to add following line to hosts ip mapping. It is necessary to do it with administrative privileges! On Linux it's /etc/hosts file On Windows it's %SystemRoot%
  system32
  drivers
  etc
  hosts

```
            127.0.0.1 login.techstructure.com
```

  On Linux also u need to reset the Network Manager service.

12.3.2 Finished product (Work in progress)

go to root project directory, then /scripts and run setup.sh

```
            sh ./setup.sh
```