

Bachelor thesis

RecipeTime: A Web Application to Support Sustainable and Personalized Meal Planning



Alba Delgado Santiago

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C/ Francisco Tomás y Valiente nº 11

**UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR**



Bachelor as Ingeniería Informática Bilingüe

BACHELOR THESIS

**RecipeTime: A Web Application to Support
Sustainable and Personalized Meal Planning**

Author: Alba Delgado Santiago

Advisor: Alejandro Bellogín Kouki

mayo 2025

All rights reserved.

No reproduction in any form of this book, in whole or in part
(except for brief quotation in critical articles or reviews),
may be made without written authorization from the publisher.

© 20 de Mayo de 2025 by UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Alba Delgado Santiago

RecipeTime: A Web Application to Support Sustainable and Personalized Meal Planning

Alba Delgado Santiago

C\ Francisco Tomás y Valiente N^o 11

PRINTED IN SPAIN

AGRADECIMIENTOS

Lo primero, agradecer con todo mi corazón a mis padres y a mi hermano, vuestro amor incondicional y apoyo constante han sido la base sobre la que he construido cada etapa de mi vida.

Luego, mi enorme gratitud para Álvaro, gracias por tus consejos, por acompañarme y por creer siempre en mí.

También, gracias a mis amigos, vuestra amistad y complicidad tanto en las horas de estudio como en las de ocio, convirtieron este camino en una aventura compartida.

Mi reconocimiento también para los profesores que, con su tiempo, dedicación y pasión por enseñar, me han formado y animado a crecer cada día. Vuestra paciencia y compromiso han sido fundamentales para llegar hasta aquí.

Un agradecimiento muy especial para Alejandro Bellogín, quiero agradecerle profundamente todo lo que me ha enseñado y, en especial, por lo bien que me ha guiado durante el desarrollo de este proyecto. Su implicación constante y sus acertados consejos han sido fundamentales para mí. Es un verdadero ejemplo a seguir y me siento increíblemente afortunada por haberle tenido como tutor.

Y, por último, pero no menos importante, a todos los que me habéis acompañado durante mi formación académica, gracias por estar ahí, por vuestro apoyo y por hacer de todo este camino una vivencia muy enriquecedora que no olvidaré jamás.

RESUMEN

Este Trabajo de Fin de Grado presenta RecipeTime, una aplicación web full-stack cuyo objetivo principal es ayudar a los usuarios a cocinar de manera más eficiente y sostenible mediante recomendaciones de recetas personalizadas. Aprovecha al máximo los ingredientes ya disponibles en la cocina virtual del usuario, reduciendo así el desperdicio de alimentos, simplificando la planificación de comidas y fomentando la diversidad alimentaria, al tiempo que se adapta a los gustos y necesidades nutricionales individuales a través de filtros detallados.

Para alcanzar estos objetivos, RecipeTime integra tres estrategias de recomendación. En primer lugar, un recomendador basado en una cocina virtual clasifica las recetas según el número de ingredientes que coinciden con el inventario del usuario. En segundo lugar, un recomendador de filtrado colaborativo emplea un modelo de Alternating Least Squares (ALS) entrenado con más de 180 000 recetas y 700 000 eventos de guardado de un conjunto de datos de Food.com para identificar factores de preferencia latentes y sugerir recetas populares entre usuarios con gustos similares. En tercer lugar, recomendación mediante filtros ofrece opciones de búsqueda dinámicas (por ejemplo, tiempo de preparación, número de ingredientes, etiquetas alimenticias y umbrales nutricionales) a través de endpoints REST de Django protegidos por JWT y un frontend adaptable desarrollado con React y Vite, asegurando una planificación de comidas completamente a medida.

Un estudio de usabilidad con dieciséis participantes evaluó las funciones clave de RecipeTime mediante tareas guiadas que incluyeron la configuración de la cocina, la exploración y filtrado de recetas y la visualización de los detalles de las recetas, concluyendo con la Escala de Usabilidad del Sistema (SUS). La aplicación obtuvo una puntuación “Excelente”, lo que confirma cómo de eficaz e intuitivo es su diseño. Al priorizar recetas basadas en los ingredientes disponibles, RecipeTime promueve hábitos de cocina más sostenibles, animando a los usuarios a cocinar con lo que ya tienen en casa en lugar de comprar nuevos productos y, de este modo, reduciendo significativamente el desperdicio de alimentos.

PALABRAS CLAVE

Inventario de cocina virtual, Recomendación de recetas personalizadas, Planificación de comidas, Reducción del desperdicio de alimentos, Filtrado colaborativo (FC), Alternating Least Squares (ALS), Escala de Usabilidad del Sistema (SUS)

ABSTRACT

This Bachelor thesis introduces RecipeTime, a full-stack web application whose primary objective is to help users cook more efficiently and sustainably by delivering personalized recipe suggestions. It maximizes the use of ingredients already owned in the user's virtual kitchen, thereby reducing food waste, simplifying meal planning and encouraging dietary variety, while adapting to individual tastes and dietary needs through fine-grained controls.

To achieve these goals, RecipeTime combines three recommendation strategies. First, a kitchen-based recommender ranks recipes by the number of matching ingredients drawn from the user's inventory. Second, a collaborative-filtering recommender uses an Alternating Least Squares (ALS) model trained on over 180 000 recipes and 700 000 save events from a Food.com dataset to learn latent taste factors and suggest recipes favored by similar users. Third, a filter-driven recommender offers dynamic search controls (e.g., preparation time, ingredient count, dietary tags, nutritional thresholds) via secure JWT-protected Django REST endpoints and a responsive React/Vite frontend, ensuring highly tailored meal planning.

A formal user study with sixteen participants evaluated RecipeTime's core features through guided tasks covering kitchen setup, recipe discovery, filtering, and detail viewing, culminating in the System Usability Scale (SUS). The app achieved an "Excellent" usability score, confirming its intuitive design. By prioritizing recipes based on your existing kitchen inventory, RecipeTime fosters more sustainable cooking habits, encouraging users to cook what they already have rather than buy new ingredients, and in doing so significantly reducing household food waste.

KEYWORDS

Virtual Kitchen Inventory, Personalized Recipe Recommendation, Meal Planning, Food Waste Reduction, Collaborative Filtering (CF), Alternating Least Squares (ALS), System Usability Scale (SUS)

TABLE OF CONTENTS

1	Introduction	1
1.1	Motivation	1
1.2	Goals	2
1.3	Thesis Structure	2
2	State of the Art	3
2.1	Recommender Systems in the Food Domain	3
2.1.1	Comparison with other food-related approaches	4
2.1.2	Previous Studies	5
2.1.3	Implicit-Feedback Collaborative Filtering Paper	6
2.1.4	The Alternating Least Squares Algorithm	6
2.2	Dataset Selection and Evaluation	7
2.3	Technological Framework and Tools	8
3	Design and Implementation	9
3.1	Project Structure	9
3.2	Requirement Analysis	10
3.2.1	Dataset and Database Module	11
3.2.2	API, Search, and Recommendation System Module	12
3.2.3	Web Module	13
3.3	Design	13
3.3.1	Dataset and Database Module	14
3.3.2	API, Search, and Recommendation System Module	14
3.3.3	Web Module	15
3.4	Implementation	15
3.4.1	Dataset and Database Module	17
3.4.2	API, Search, and Recommendation System Module	20
3.4.3	Web Module	24
4	Experiments and Results	27
4.1	User Study	27
4.1.1	Consent	28
4.1.2	Demographics & Cooking Background	28
4.1.3	Motivations & Preferences	30
4.1.4	Guided Task Walkthrough	31

4.1.5 Post-Test Experience & Feedback	35
4.1.6 Measuring Usability with the SUS	36
4.2 Analysis of results	37
5 Conclusions and Future Work	39
5.1 Conclusions	39
5.2 Future Work	40
Bibliography	44
Appendices	45
A Code Fragments	47
B Web Application	49
B.1 Authentication	49
B.2 Recommendations Flows	50
B.3 User Personal Space	51

LISTS

List of equations

3.1	Similarity equation	23
-----	---------------------------	----

List of figures

3.1	Structure diagram of the project.....	10
3.2	Entity relationship diagram.....	14
3.3	Recipe Detail & Save/Unsave Recipe.....	15
3.4	High-level application flow.....	16
3.5	General search.....	26
4.1	Consent bar chart.....	28
4.2	Age Distribution.....	28
4.3	Gender distribution.....	29
4.4	How often they cook.....	29
4.5	Kitchen skill.....	29
4.6	User motivations for using a recipe recommendation app.....	30
4.7	Interest in ingredient-based recommendations.....	30
4.8	Recipe Selection Criteria.....	31
4.9	Login and register task.....	31
4.10	My kitchen task.....	32
4.11	My shopping list task.....	32
4.12	Recommendations kitchen-based page.....	32
4.13	Save recipe task.....	33
4.14	Delete recipe task.....	33
4.15	Recommendations saved-based page.....	34
4.16	General search task.....	34
4.17	Recipe detail task.....	35
4.18	Clarity of scoring metrics.....	36
B.1	Login page.....	49
B.2	Register page.....	49
B.3	Recipes based on your kitchen page.....	50

B.4	Recipes based on your saved recipes page.	50
B.5	Recommendations page.	51
B.6	My kitchen page.	51
B.7	My shopping list page.	52
B.8	Saved recipes page.	52

List of tables

3.1	Registers in the Database	19
4.1	Post-test experience summary	35

INTRODUCTION

This chapter opens with the main motivation of this work, by examining the environmental, economic, and social dimensions of food waste. It then presents the project's main goals and breaks the primary objective into three key parts. Finally, it outlines the overall structure of the thesis.

1.1. Motivation

In recent years, the scale of food waste has become impossible to ignore. According to the Food and Agriculture Organization, nearly one third of all food produced (about 1.3 billion tonnes) goes to waste before it ever reaches our tables. Instead, it sits in landfills, where it breaks down and releases methane, a greenhouse gas far more potent than carbon dioxide [1]. This is not just an environmental concern: when perfectly good groceries spoil, families and businesses lose money, sometimes hundreds of euros each year, as they toss unused food into the bin [2]. All the while, millions of people around the world go hungry, highlighting a stark contrast between abundance and need [3].

At the same time, our eating habits are linked to a growing public-health burden. Diets high in processed sugars, unhealthy fats, and excess salt contribute to rising rates of obesity, type 2 diabetes, and heart disease [4]. Yet finding recipes that balance taste, nutrition, and personal dietary restrictions can feel like a treasure hunt. Many home cooks spend valuable time hopping between websites, bookmarking recipes that may call for ingredients they do not have or cannot eat. Existing meal-planning tools tend to offer broad collections of recipes or basic shopping lists but rarely tailor suggestions to the ingredients sitting in your own kitchen, your health goals, or your budget.

That gap in the market and the frustration I faced as a university student juggling multiple food intolerances inspired the creation of RecipeTime. First, it lets you keep an up-to-date virtual kitchen, so you always know exactly what is on your shelves. Next, it recommends recipes that use as many of those ingredients as possible, showing you how many ingredients of that recipe you already have. By doing this, you cut down on impulse purchases and give old ingredients a chance to shine. On top of that, the app learns your tastes: once you save a few favorites, it can suggest new recipes that users with similar preferences have enjoyed. Finally, you can search through over 230 000 recipes using

filters for cooking time, number of ingredients, tags such as *easy* or *italian* and nutrition targets like *fats*, *sugars*, or *sodium*. By bringing these features together, we aim to help people waste less, spend less money, and spend more time enjoying the food they cook.

1.2. Goals

The main objective of our project is to help users cook more efficiently and sustainably by delivering personalized recipe suggestions. To achieve this, we divided the problem into 3 key parts.

The first key part is to help users maximize the ingredients they already have in their kitchens. By recommending recipes that include as many ingredients from their kitchen as possible, with the aim of reducing food waste and avoiding unnecessary purchases. The second key part is to simplify meal planning and encourage dietary variety. Offering personalised suggestions by analysing the recipes a user has saved and the preferences of similar users, using collaborative filtering. The third and last key part is to be able to adapt to every individual tastes and dietary need. Providing filters for cooking time, ingredient count, preparation steps, average rating, tags (for example, vegan, gluten-free), and nutritional ranges such as calories, protein, sodium and so on.

By implementing these key parts in our application we aim to reduce food waste, lower household grocery spending, and make meal planning easier for people with diverse dietary requirements and lifestyles.

1.3. Thesis Structure

We will briefly explain the structure of our thesis. It is divided into five chapters, each covering a different aspect of the process:

Chapter 1. Introduction: Presents the motivation, objectives and overall outline.

Chapter 2. State of the art: Reviews prior work on food recommendation systems and presents the chosen algorithm.

Chapter 3. Design and Implementation: Details the design decisions and the development of each module in the application.

Chapter 4. Experiments and results: Describes the usability tests conducted and analyses their outcomes.

Chapter 5. Conclusions and Future Work: We summarised the main findings and outline what ideas would be possible for future work

STATE OF THE ART

In this chapter, we explore the current state of the art in recipe recommendation. We begin by examining four key types of food recommenders (home cooking, grocery shopping, restaurant selection, and health-focused) and highlight what makes each one unique. Next, we briefly discuss related work in the food recommendation field, from broad systematic reviews to the seminal implicit-feedback collaborative-filtering paper. With that context in place, we then explain the core algorithm used in our work, showing how it learns from users' saved events to deliver personalized suggestions. We then review the technologies and design choices that bring these ideas to life. Finally, we discuss how to select and evaluate recipe datasets.

2.1. Recommender Systems in the Food Domain

In domains like film, music, and online retail, recommender systems have become indispensable for predicting user tastes and keeping people engaged [5]. Netflix's matrix-factorization approach (Koren et al [6]) truly transformed movie personalization, demonstrating that alternating-least-squares factor models far outperformed neighborhood methods, van den Oord et al [7] show how deep convolutional nets can predict collaborative-filtering latent factors directly from raw audio an approach that helps streaming services (like Spotify) solve cold-start song recommendations and Sarwar et al's [8] item-to-item collaborative-filtering algorithm underpins Amazon's "Customers who bought X also bought Y" engine by precomputing item-item similarities from huge purchase logs.

When we move into food, however, the stakes and data complexity rise steeply. Meal choices intertwine with cultural identity, health considerations, and social context in ways that movie or product picks rarely do. As Elswailer et al. explain in [9], a food recommender must juggle structured inputs such as ingredient lists, nutritional values, and unstructured sources like cooking instructions and user reviews, all while honoring dietary restrictions and evolving tastes.

Unlike the movie and music realms, which enjoy standardized, large-scale datasets, the culinary domain is fractured: recipe collections and review platforms vary wildly in format and quality. Bondevik et al. [10] highlight this heterogeneity as a key challenge, showing that food recommenders must span

content-based, collaborative-filtering, graph-based and hybrid paradigms to cope with such diverse data sources. Together, these factors call for a comprehensive theoretical framework to guide search, filtering, and hybrid recommendation techniques in the food setting.

2.1.1. Comparison with other food-related approaches

This section is inspired by the “Food Recommender Systems” chapter [9] in the Recommender Systems Handbook by Elswailer et al. [5]. We divide food recommenders into four categories: cooking, grocery, restaurant, and health. We illustrate each with concrete examples from the literature.

Cooking recommenders aim to turn whatever is in the user’s kitchen into complete, workable recipes. Some examples included in the chapter [9] would be the evaluation of the *What’s Cooking dataset* where a model analyzes a partial ingredient list and predicts which items are missing to finish the recipe, helping households reduce waste. The *WikiTaaable* dataset, which suggests safe substitutions whenever allergens or dietary preferences rule out a given ingredient. Both systems rely on structured recipe metadata (ingredient quantities, cooking steps) and also extract meaning from unstructured text such as preparation instructions and flavor descriptors to rank options against constraints like time or skill level.

Grocery recommenders switch the spotlight to individual products rather than full meals. Examples of work on recommending food products to buy during grocery shopping would be *Lucky’s Market’s augmented-reality app*, for instance, overlays allergy warnings and healthier alternatives on store shelves in real time, guiding shoppers toward safer, more nutritious choices. *Open Food Facts* taps into a crowd-sourced database of product labels to recommend nutritionally similar items with lower environmental impact. In each case, price, nutritional content, and sustainability scores become key signals alongside user purchase history and profile data.

Restaurant recommenders suggest where to eat by weighing cuisine type, cost, distance and individual or group tastes. For example, one study uses *TripAdvisor* data to cluster diners into segments (families, business travelers, couples) and then ranks restaurants most popular within each segment. Another system from *Baidu Map dataset* fuses geospatial check-in logs with aggregate user ratings to generate a personalized list of nearby venues, displayed on a map and filtered by opening hours and budget.

Health recommenders integrate medical and nutritional guidelines into every suggestion. We see examples as a diabetes-focused planner described by Elswailer et al. [9] which designs daily menus that stabilize carbohydrate intake and adhere to caloric limits. Another example from the literature would be a toddler-nutrition recommender, which generates recipe suggestions calibrated to meet age-specific dietary standards by combining pediatric nutrient ranges with user flavor preferences.

2.1.2. Previous Studies

The food recommendation literature has grown rapidly, spanning broad surveys and specialized algorithmic innovations. Bondevik et al [10] provide a systematic review that divides existing systems into content-based, collaborative filtering, graph-based, and hybrid approaches, and they highlight gaps in recipe personalization.

Trattner & Elswailer [11] extracted 60 983 recipes (with 1 032 226 ratings from 125 762 users) from Allrecipes.com and pitted a range of collaborative-filtering methods (BPR, WRMF, UserKNN, ItemKNN) against content-based recommenders. When they limit their experiments to very active users and items (a “p-core” filter), collaborative-filtering needs as few as 13 users to outperform the best content-based model. But under a more typical, sparsely sampled dataset, they did not see consistent CF wins until they had over 600 users. Their work clearly shows that the choice of sampling strategy and evaluation protocol can completely reverse which algorithm looks superior. Those same authors, Trattner & Elswailer in [12] analyze the healthiness of 60 983 recipes from Allrecipes.com using WHO and UK FSA nutritional guidelines, investigate how user signals (ratings, bookmarks, comment sentiment) correlate with recipe health scores, and evaluate both single-item recommendation and daily meal-plan generation to assess a recommender’s potential to nudge users toward healthier eating.

The following two papers push beyond traditional CF/CB by learning richer representation for complex data. Majumder et al [13] use transformer encoder-decoder models with attention over user history to generate personalized cooking instructions, demonstrating significant gains in user satisfaction. Teng et al. [14] construct two “ingredient networks” (co-occurrence and user-driven substitution) from a large Allrecipes.com corpus, extract structural features (community memberships, centralities), and show that these network-derived features when combined with simple nutrition and metadata can predict user recipe ratings with nearly 0.79 accuracy, revealing powerful latent relationships among ingredients that support effective recipe recommendation.

Hybrid recommenders combine multiple paradigms to address cold-start and sparsity. Freyne & Berkovsky [15] systematically compare six recipe recommendation strategies (including pure CF, pure CB, and two hybrid algorithms that decompose recipes into ingredients before applying collaborative and content-based reasoning) and show that their hybrid recipe-based methods (recipehr, recipehf) achieve the lowest MAE (<0.20) and broadest coverage, outperforming both standalone CF and CB approaches.

Graph-based and knowledge-driven approaches enrich recommendation with external structure. Ahn et al [16] propose the Flavor Network, which links ingredients by shared chemical compounds and supports harmonious flavor pairings. Haussmann et al [17] present FoodKG, a semantics-driven food knowledge graph integrating recipes, ingredients, nutrients, and provenance to enable context-aware dietary and allergy-safe recommendations.

Although these prior works span a wide array of techniques, we focus on implicit-feedback collaborative filtering, specifically alternating least squares approach.

2.1.3. Implicit-Feedback Collaborative Filtering Paper

Our work draws most directly on the paper “Collaborative Filtering for Implicit Feedback Datasets” by Hu et al [18]. In this landmark study, the authors adapt matrix factorization to settings where users express interest implicitly through actions such as clicks or saves rather than explicit ratings. They introduce a confidence model that weights observed interactions more heavily than unobserved ones and formulate recommendation as a regularized least-squares problem. The paper demonstrates how alternating between solving for user factors and item factors leads to efficient, scalable learning on very large implicit datasets. We chose this paper because it aligns perfectly with our save-based interaction model and provides both the theoretical framework and practical implementation details needed to handle hundreds of thousands of recipes.

2.1.4. The Alternating Least Squares Algorithm

Alternating least squares, or ALS, is a collaborative filtering technique that fits our save-based interaction data perfectly. At a high level, ALS assumes that both users and recipes can be represented by a small number of latent “taste” factors. During training, the algorithm learns one set of vectors for users and another set for recipes so that the dot product of a user’s vector and a recipe’s vector predicts the strength of that user–recipe relationship.

Concretely, let $R \in \mathbb{R}^{\text{recipes} \times \text{users}}$ (recipes on rows, users on columns) be our interaction matrix, where each element R_{iu} records how many times recipe i was saved by user u . ALS approximates R as the product of two lower-rank factor matrices, $R \approx XY^T$ where

- $X \in \mathbb{R}^{\text{users} \times f}$ holds the **user-factor** vectors $x_u \in \mathbb{R}^f$ (one row per user).
- $Y \in \mathbb{R}^{\text{recipes} \times f}$ holds the **recipe-factor** vectors $y_i \in \mathbb{R}^f$ (one row per recipe).

The transpose in Y is because by writing $R \approx XY^T$, the (u, i) entry becomes $(XY^T)_{ui} = x_u \cdot y_i$, exactly the dot product of the user’s embedding x_u and the recipe’s embedding y_i . That dot product is our model’s prediction of how likely user u is to save recipe i .

After decomposing R into X and Y , ALS trains these factor matrices by repeating two simple steps until the model stops improving:

- 1.– **Update user factors** by holding all recipe vectors Y fixed, and for each user u solve a regularized least-squares problem to find the vector x_u that best predicts that user’s save history.
- 2.– **Update recipe factors** by holding all user vectors X fixed, and for each recipe i solve the analogous least-squares problem to update y_i .

Because each subproblem reduces to standard ridge regression and only touches the observed (nonzero) entries of R , ALS runs quickly even on extremely sparse data. Once the model converges, making a recommendation is just a matter of computing the dot products $x_u \cdot y_i$ and selecting the highest scores.

Empirical studies confirm that ALS excels in this implicit-feedback setting. He et al [19] evaluated several matrix-factorization variants on large real-world datasets (Yelp and KDD Cup) and report that, with the right optimizations, ALS achieves both the fastest training times and the best ranking accuracy. Their experiments show that ALS not only has solid theoretical footing but also proves to be a reliable, scalable solution for recommending recipes at large scale.

2.2. Dataset Selection and Evaluation

We investigated several food-related datasets that we will present next, such as Food.com Recipes and Interactions, RecipeNLG, Epicurious Recipes with Rating and Nutrition, the Recipe Ingredients Dataset, AllRecipes, RecetasDeLaAbuela, Recetas Cocina, and two popular APIs (Spoonacular and Edamam). The Food.com collection on Kaggle offers over 180 000 recipes and 700 000 user reviews with 18 years of interaction history [20]. RecipeNLG provides a large corpus of ingredient lists and instructions but lacks any user feedback [21]. Epicurious brings ratings and nutrient data for around 20 000 recipes, though its scale is modest compared to other sources [22]. The Recipe Ingredients Dataset delivers JSON-formatted ingredient lists without interactions [23]. AllRecipes' scraped data includes user ratings but again falls short on nutritional details and interaction depth [24]. RecetasDeLaAbuela and Recetas Cocina supply traditional Spanish recipes with ingredient and instruction metadata but no user reviews or nutrient values [25, 26]. Finally, the Spoonacular API grants real-time recipe, ingredient, nutrition, and some interaction data for developers [27], while the Edamam API focuses on high-resolution nutritional analysis alongside basic recipe search [28].

After weighing size, richness of user interactions, metadata quality, and ease of reuse, we settled on the **Food.com Recipes and Interactions dataset** [20]. It combines over 180 000 recipes with 700 000 and save events covering from 2000 to 2018. More than just raw text, the dataset comes pre-packaged into a set of files that make modeling straightforward:

- **PP_recipes.csv** lists each recipe's title, ingredient list, preparation steps, tags, and nutritional values (calories, protein, fat, sodium).
- **PP_interactions.csv** captures every save event, linking user IDs to recipe IDs with timestamps.
- **PP_users.csv** summarizes user activity, including total saves and average interactions per recipe.
- **ingre_map.csv** maps internal ingredient IDs to their human-readable names.
- **train/validation/test** splits let us evaluate models out of the box without custom data slicing.

Because the dataset comes in straightforward CSV files with clear labels it facilitates the next steps

which would be loading, preprocessing, and experimenting with it. For instance, researchers like Majumder et al leveraged this dataset to develop transformer-based recipe generators [13], underscoring its breadth and adaptability.

In addition, we also looked at Ethan Schacht's exploratory analysis on Kaggle [29], where he digs into ingredient popularity, recipe length, and text patterns. His findings confirmed what we suspected: this dataset holds plenty of rich signals for advanced modeling. We intend to carry out our own data analysis, but this prior study has guided our approach to data exploration.

2.3. Technological Framework and Tools

Choosing the right back-end and front-end frameworks has a major impact on how quickly a recommendation system can be developed and how easily it can be maintained. Selecting a back-end framework begins with choosing the programming language: Python is widely regarded as the go-to language for building web applications. In 2024 it was the third programming language most used with a 51 % usage after HTML/CSS with 52,9 % and JavaScript with 62,9 % ¹. Within the Python ecosystem, Django leads with 63 % adoption, followed by Flask at 42 %, and FastAPI at 41 % ², the latter's growth fueled by its native support for asynchronous operations.

We therefore first chose Python for its broad community, mature libraries, and strong tooling, and then selected Django. Our decision to use Django builds on the practical experience I gained in my PSI (Proyecto Sistemas Informáticos) coursework. Django's integrated support for PostgreSQL via its ORM, its out-of-the-box authentication system and admin interface lets us focus on implementing the recommendation logic itself rather than wiring up boilerplate.

On the front end, modern JavaScript frameworks shape the user experience. The 2024 State of JS survey reports that React is used by 82 % of developers, Vue.js by 51 %, Angular by 50 %, and Svelte by 26 % ³. Additionally, Vite has emerged as the fastest-growing build tool, with a +25 % year-over-year jump in usage ⁴, thanks to its instant hot-module replacement and zero-config setup.

By combining Django REST Framework on the back end with a Vite-powered React front end, we maintain a clean separation of concerns: Django efficiently delivers JSON endpoints that our front end can fetch without delay, and Vite ensures rapid rebuilds when we tweak UI components. This decoupled architecture accelerates our workflow, allowing back-end changes and front-end experiments to proceed in parallel without blocking one another.

¹<https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/>, last accessed in May 2025.

²<https://blog.jetbrains.com/pycharm/2024/12/the-state-of-python/>, last accessed in May 2025.

³<https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/>, last accessed in May 2025.

⁴<https://2023.stateofjs.com/en-US/awards/>, last accessed in May 2025.

DESIGN AND IMPLEMENTATION

In this chapter, we aim to provide a clear and comprehensive overview of our system's design, explain the implementation details, and discuss the key decisions that guided its development. The project, at its core, is divided into three major modules, each focusing on a distinct part of the overall workflow. These modules are integrated to offer users a unified platform for accessing personalized recipe recommendations.

3.1. Project Structure

Our project is organized into three interconnected modules, as depicted in Figure 3.1, each distinguished by a unique color in the system's architecture diagram, that together create a robust recipe recommendation platform.

The first module, **Web Application** (Blue), forms the user interface layer. Developed with React and Vite, it is integrated alongside Django to deliver a dynamic and intuitive frontend, handle routing, and interact with the backend via API endpoints. Its design ensures that users can search the recipe database, view personalized recommendations, and manage their culinary preferences in real time.

The second module, **API, Search, and Recommendation System** (Orange), serves as the intermediary between the frontend and the underlying data. By taking advantage of the Django REST Framework, this module processes incoming requests from the web interface and communicates with the database to retrieve and update information. It offers advanced filtering and search functionalities, including a specialized feature that suggests recipes based on the ingredients users have available in their kitchens. Additionally, the collaborative filtering component analyzes users' saved recipes and interactions to recommend new recipes that align with their taste profiles. A general search function also allows users to explore the entire dataset, ensuring comprehensive access to all available recipes.

The third module, **Dataset and Database** (Green), underpins the entire system by managing and storing all essential data. For our project, we sourced a comprehensive dataset from Food.com and applied preprocessing to clean, standardize, and enrich the raw recipe data addressing issues such as duplicates, inconsistent formatting, and missing values. An in-depth analysis of this refined dataset

informed the design of an Entity-Relationship schema that organizes recipes, user profiles, and interactions, which we then used to populate a PostgreSQL database. This approach ensures reliable data storage and efficient retrieval, supporting accurate search results and personalized recommendations.

Each of these primary modules is further divided into specialized submodules that address specific tasks, as it will be explained later. Together, these components interact to form a cohesive system, enabling a dynamic and responsive experience for users exploring and discovering new recipes. Figure 3.1 provides a visual representation of the data flow and interactions among these modules, offering a summary of the project's architecture. The diagram outlines the data journey, beginning with the pre-processing of the Food.com data set and culminating in the delivery of personalized recipe recommendations through the Web application.

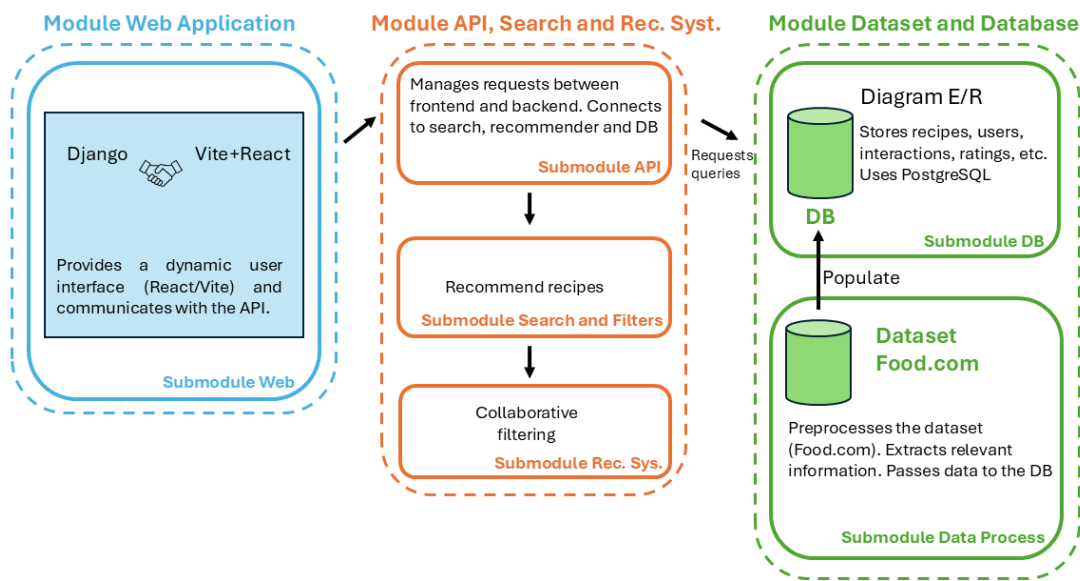


Figure 3.1: Structure diagram of the project.

3.2. Requirement Analysis

In this section, we present the requirements analysis for the various modules of the system. For each module, the requirements are divided into two categories: functional requirements and non-functional requirements. The functional requirements detail the specific capabilities and operations that each module must support, while the non-functional requirements outline the performance, security, scalability, and maintainability criteria that ensure overall system robustness.

3.2.1. Dataset and Database Module

Functional Requirements

- FR-DD-1.**— The module shall persistently store all recipe-related data in a PostgreSQL database managed via Django.
- FR-DD-1.1.**— The database schema must include key entities such as Recipe, Ingredient, RecipeIngredient, Interaction, UserProfile, UserCategory, UserFavorite, UserKitchen, and UserShoppingList.
- FR-DD-1.2.**— All relationships (one-to-one, one-to-many, and many-to-many) must be correctly implemented and enforced according to the design.
- FR-DD-2.**— The module shall ensure data integrity by validating incoming data for missing values, duplicates, and proper foreign key references.
- FR-DD-2.1.**— Critical fields (e.g., recipe_id, ingredient names, date fields) shall be indexed to optimize query performance.
- FR-DD-3.**— The dataset creation and ingestion process must be fully automated.
- FR-DD-3.1.**— The system shall include processes to inspect source data files for structure, quality, and relationships.
- FR-DD-3.2.**— Data population processes must insert records in batches with robust error handling to manage duplicates and ensure efficiency.
- FR-DD-4.**— The module shall integrate with the user authentication system to support user-specific data management.
- FR-DD-4.1.**— Each user must have an associated profile enabling features such as saving recipes, managing kitchen inventory, and categorizing ingredients.
- FR-DD-5.**— The module shall support post-ingestion adjustments and validation processes.
- FR-DD-5.1.**— Mechanisms must be provided to verify that the final table counts and data integrity match expected outcomes.

Non-Functional Requirements

- NFR-DD-1.**— The dataset creation and ingestion process must be efficient, capable of handling large volumes of data with minimal memory overhead.
- NFR-DD-2.**— The module must provide high-performance query responses by leveraging optimized batch processing and proper indexing.
- NFR-DD-3.**— The system must be scalable, allowing for future expansion and increased data volumes without compromising performance.
- NFR-DD-4.**— The module must be robust, with comprehensive error handling to ensure data consistency and integrity.
- NFR-DD-5.**— The codebase shall be modularly structured, and should be able to be improved through iterative enhancements (e.g., via Django migrations).
- NFR-DD-6.**— The system must enforce secure access controls following standard authentication and authorization practices.
- NFR-DD-7.**— ER diagrams and model-design documents for the database shall be updated with every schema change so they always reflect the current implementation.

3.2.2. API, Search, and Recommendation System Module

Functional Requirements

FR-ASR-1.— The module shall provide a secure and structured REST API using Django REST Framework to mediate between the PostgreSQL backend and the React/Vite frontend.

FR-ASR-1.1.— The API shall convert complex Django models into JSON responses using well-designed serializers, including computation of additional fields (e.g., average ratings, rating counts) to support the frontend display requirements.

FR-ASR-2.— The module shall support standard CRUD operations on recipe-related data via viewsets.

FR-ASR-2.1.— Viewsets shall handle listing, retrieving, updating, and deleting records.

FR-ASR-2.2.— Custom actions shall be implemented to support functionalities such as recommending recipes based on user kitchen ingredients, aggregating collaborative recommendation scores, and filtering recipes according to various criteria.

FR-ASR-3.— The module shall expose user-centric endpoints for managing personalized data.

FR-ASR-3.1.— Endpoints shall allow users to add ingredients to their virtual kitchen, manage organizational categories, and save or unsave recipes (ensuring that all operations are specific to the authenticated user).

FR-ASR-4.— The module shall integrate robust authentication and account management mechanisms.

FR-ASR-4.1.— Endpoints need to be protected and shall require a valid JSON Web Token (JWT) in the Authorization header to ensure secure access.

FR-ASR-4.2.— The account management system shall support secure user registration and login via custom serializers.

FR-ASR-5.— The module shall enable efficient search and filtering of recipes.

FR-ASR-5.1.— The search functionality shall retrieve recipes based on matching ingredients from the user's virtual kitchen.

FR-ASR-5.2.— The filtering functionality shall support text-based search and filtering by criteria such as ratings, preparation times, step counts, ingredient counts, and nutritional values.

FR-ASR-6.— The module shall support personalized recipe recommendations through collaborative filtering techniques.

FR-ASR-6.1.— The system shall apply an algorithm to help manage and reduce the size of the interaction data for improved processing efficiency.

FR-ASR-6.3.— A precomputed similarity table shall aggregate similarity scores from saved recipes and provide a ranked list of recommendations.

Non-Functional Requirements

NFR-ASR-1.— The system shall leverage Django's standard security framework for all user-related operations.

NFR-ASR-2.— The module shall be highly performant, with optimized query response times achieved through efficient data processing, appropriate indexing, and the use of pagination where needed.

NFR-ASR-3.— The recommendation functionality shall scale effectively to handle large datasets and high volumes of user interactions.

NFR-ASR-4.— The module shall be designed for maintainability and extensibility, following best practices in code modularity, clear separation of concerns and easy to understand documentation.

NFR-ASR-5.— The overall system shall deliver a responsive user experience by minimizing latency through backend processing optimizations.

3.2.3. Web Module

Functional Requirements

FR-WEB-1.— The module shall provide a dynamic and responsive user interface using React coupled with Vite.

FR-WEB-1.1.— The frontend shall implement a single-page application (SPA) model, supporting smooth, client-side routing and navigation via React Router.

FR-WEB-1.2.— The application architecture shall be component-based, ensuring reusability and maintainability.

FR-WEB-2.— The module shall reliably integrate with the backend through secure API calls.

FR-WEB-2.1.— All API calls shall include proper JWT-based authentication to ensure secure data exchange.

FR-WEB-2.2.— The module shall implement error handling and provide clear user feedback for any failed requests.

FR-WEB-3.— The module shall offer an optimized search interface for recipes and associated tags.

FR-WEB-3.1.— Auto-complete functionality shall be provided for ingredient and tag search inputs, leveraging efficient database query techniques to ensure rapid response times.

FR-WEB-4.— The module shall ensure that the user interface adapts to various screen sizes.

FR-WEB-4.1.— Modern visual design principles, including responsive layouts and contemporary styling (such as “glass” effects), shall be implemented using dedicated CSS.

FR-WEB-5.— The module shall manage client-side state effectively.

FR-WEB-5.1.— Persistent storage solutions shall be used to retain user preferences such as search terms, active filters, and sorting options.

FR-WEB-6.— The module shall provide clear and consistent feedback to users regarding their interactions, including actions like logging in, saving recipes, or updating kitchen items.

Non-Functional Requirements

NFR-WEB-1.— The web module shall exhibit high performance and responsiveness, minimizing latency through optimized API integration and efficient client-side processing.

NFR-WEB-2.— The system shall scale effectively to support a high volume of concurrent user interactions without performance degradation.

NFR-WEB-3.— The web UI shall function correctly in the latest stable versions of Chrome, Firefox, and Edge.

NFR-WEB-4.— The user interface shall maintain visual consistency and offer a pleasing experience across multiple screen sizes.

NFR-WEB-5.— The module's codebase shall be modular, maintainable, and extensible, supported by clear documentation to facilitate future enhancements.

NFR-WEB-6.— The module shall incorporate resilient error detection.

3.3. Design

This section outlines the architecture of the recipe recommendation system, detailing the design and interaction of its core components. It provides a foundation for understanding how data management, advanced recommendation algorithms, and a responsive web interface work together to deliver personalized recipe suggestions.

3.3.1. Dataset and Database Module

This module is responsible for the structured organization and persistent storage of recipe-related data. We developed a comprehensive class diagram to clearly define the core entities, such as *Recipe*, *Ingredient*, *RecipeIngredient*, *Interaction*, *UserProfile*, *UserCategory*, and *UserFavorites* and their relationships (shown in Figure 3.2). The diagram served as a blueprint for designing the database schema in Django, ensuring that relationships (one-to-one, one-to-many, and many-to-many) are properly defined.

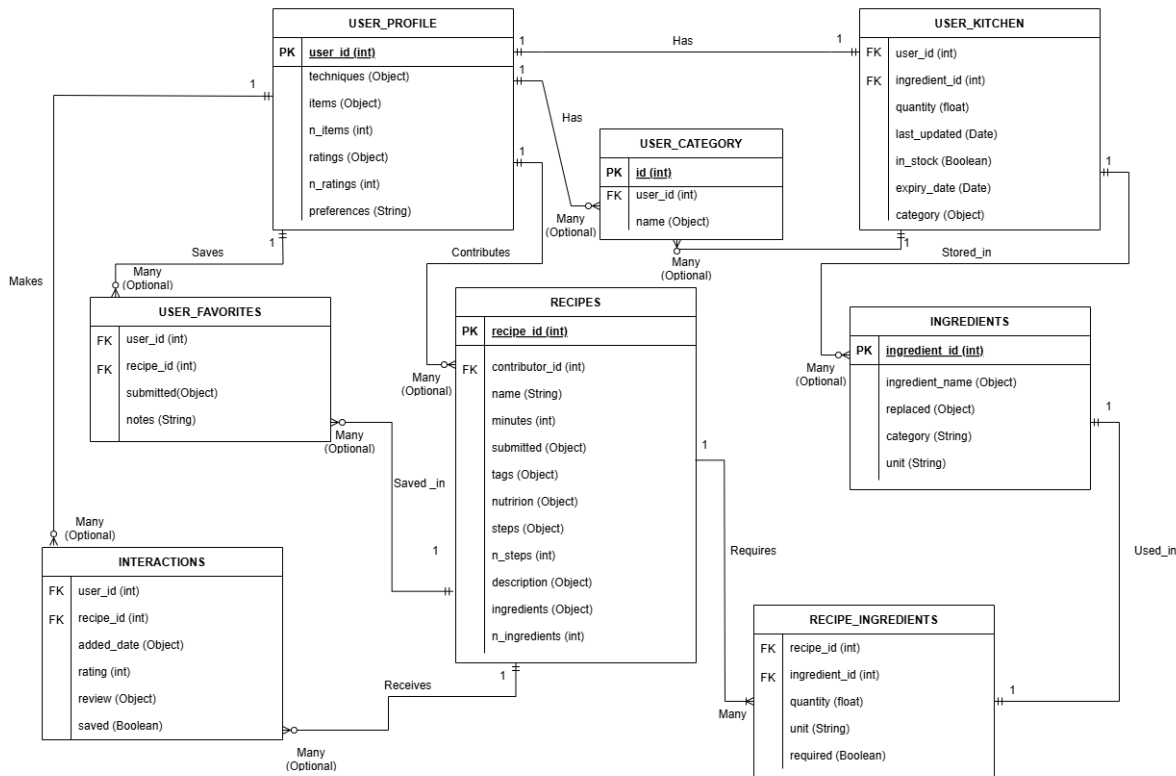


Figure 3.2: Entity relationship diagram.

3.3.2. API, Search, and Recommendation System Module

This module employs Django Rest Framework to provide a comprehensive set of endpoints that bridge the gap between the database and the user interface. It enables the dynamic retrieval and filtering of recipe data, supporting personalized recommendation functions, such as suggestions based on the user's kitchen ingredients or recipes they have already saved. The Figure 3.3 illustrates a typical interaction when a user requests detailed recipe information, checks whether the recipe is already saved, and toggles its saved status. This sequence clarifies how the React/Vite frontend, the Django REST API, and the PostgreSQL database work together to handle user-specific operations, ensuring that any changes to the user's saved recipes are accurately recorded and promptly reflected in the interface.

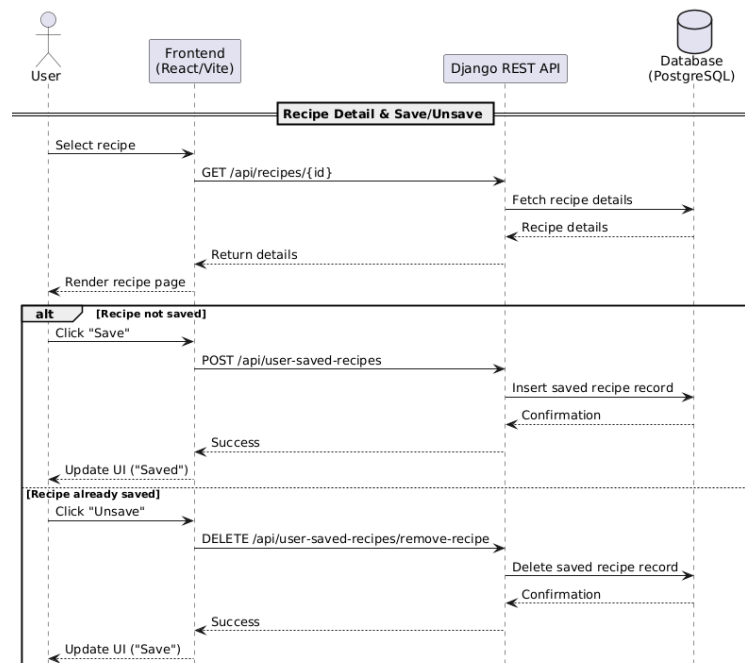


Figure 3.3: Sequence Diagram: Recipe Detail & Save/Unsave Recipe.

3.3.3. Web Module

The Web module provides a dynamic and responsive user interface using React coupled with Vite, effectively integrating with the backend through API calls. The design emphasizes a smooth flow from user interactions, such as logging in, browsing recommendations, and accessing personalized sidebar features to the corresponding responses received from the API. To illustrate these interactions, this module includes a high-level sequence diagram, shown in Figure 3.4. It captures the flow of actions from initial user input, through API communication, to data retrieval from the database, thereby ensuring a cohesive and user-focused experience throughout the system.

3.4. Implementation

In this section, we describe how each of the three modules introduced in the project structure was brought to life. First, we outline the data acquisition and storage processes within the Dataset and Database module, highlighting its preprocessing routines and PostgreSQL schema. Next, we detail how the API, Search, and Recommendation System module manages filtering logic, collaborative filtering, and general search capabilities. Finally, we explore the Web module, where a React and Django based interface allows users to effortlessly interact with the system's features. Through these steps, we illustrate the core submodules and the practical considerations that guided their development.

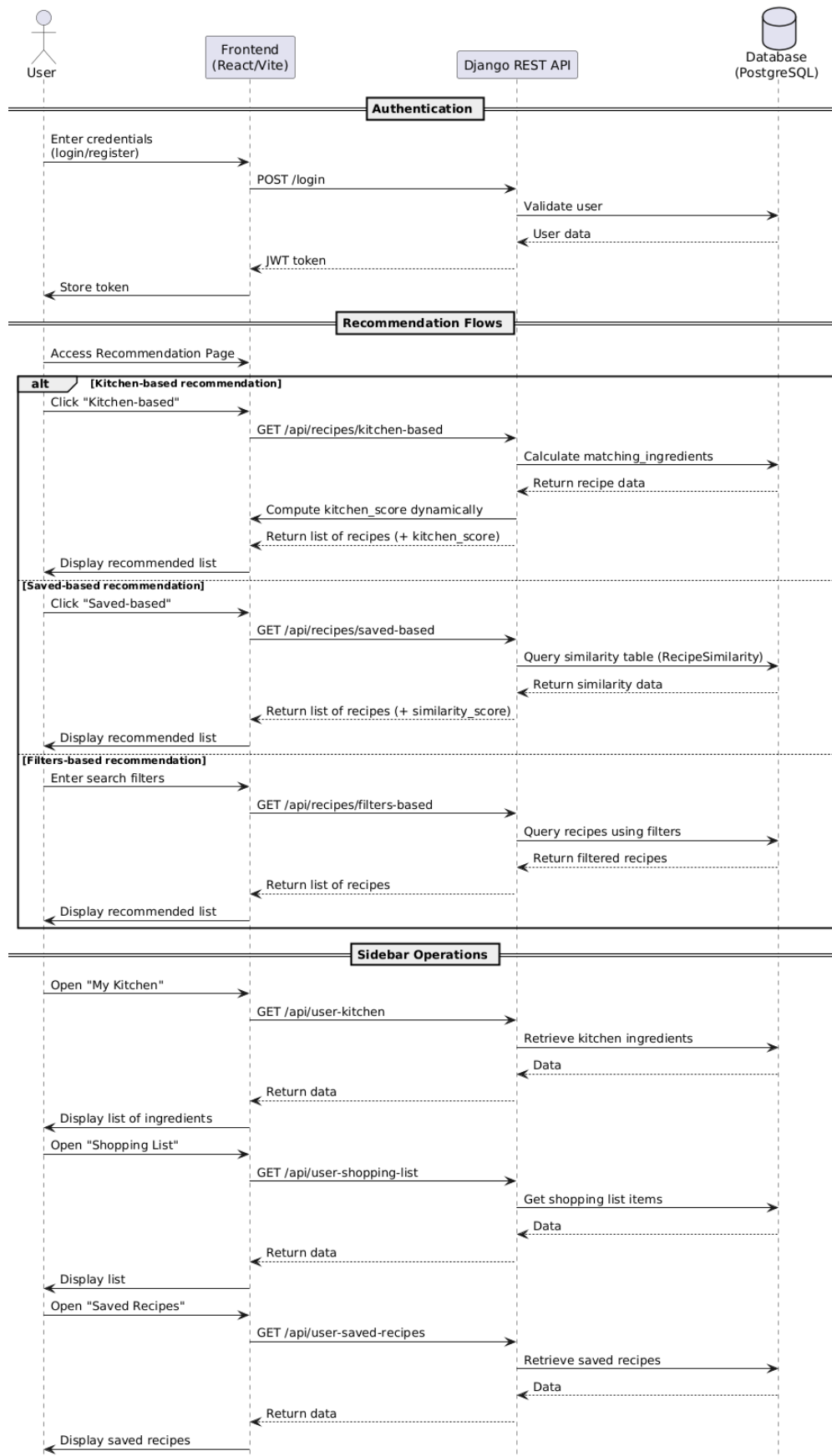


Figure 3.4: Sequence diagram: High-level application flow.

3.4.1. Dataset and Database Module

Data Preparation

Dataset Selection and Initial Analysis

The starting point for this module was identifying a comprehensive and well-structured dataset suitable for a recipe recommendation system. The Food.com dataset emerged as the ideal choice, offering a large volume of recipes, user interactions, and partial user profiles. Before any database design or model creation, we created two main Python scripts, named *food_database_analysis* and *food_database_explore* so we could inspect the CSV files. Each was in charge of checking different aspects of the dataset.

1. *food_database_analysis.py*:

- **Structure & Relationships:** This script checks column data types, potential missing values, and overall table size. It also validates foreign key relationships, such as whether every recipe in the interactions table actually exists in the recipes table.
- **Data Insights:** The script computes averages like the mean number of ingredients per recipe and the average number of recipes rated per user.

2. *food_database_explore.py* (extended version):

- **Data Quality:** Building upon the initial script, it delves into detecting duplicate rows, outliers, and potential string-format inconsistencies.
- **Patterns & Constraints:** It examines text field lengths (e.g., *name*, *description*) and date ranges (*submitted*), which guided decisions on field sizes and optionality in the Django models.
- **Relationship Validation:** It reaffirms key constraints, such as ensuring no invalid references exist between users and recipes.

Both scripts offered valuable insights. For instance, they revealed that each recipe has an average of roughly nine ingredients, descriptions can be very long, and user coverage in the interactions file might not fully match the user IDs found in the user CSV. This knowledge shaped how the data would be modeled and handled during insertion.

Ingredient Pickle Analysis

In addition to the CSV files, the dataset included a pickle file (*ingr_map.pkl*) containing ingredient details. Attempting to load this file initially caused the error:

```
No module named 'pandas.core.indexes.numeric'
```

This issue stemmed from version mismatches in the local Python environment. Downgrading both pandas (to 1.5.3) and numpy (to 1.24.2) resolved the conflict [30]. Afterward, a dedicated script named *ingredient_data_analysis.py* was created to open the pickle, inspect its columns, and verify data consis-

tency (e.g., checking for null values and data types). This analysis confirmed the presence of thousands of distinct ingredient records, many of which were duplicated or required merging.

Database Setup

Models and Relationships

Following the data inspection phase, the core database schema was designed within Django's *catalog/models.py*. The schema leverages Django's User model for authentication and introduces a **User-Profile** model as a one-to-one extension to store additional attributes (e.g., *csv_user_id*, *techniques*, *preferences*). Key entities include:

- **Recipe**: Holds metadata like *name*, *minutes*, *description*, and an array of ingredients (*ingredients*). It references UserProfile via contributor, ensuring each recipe can be traced back to its author.
- **Ingredient**: Stores individual ingredient entries, each identified by a unique *ingredient_id*.
- **RecipeIngredient**: Resolves the many-to-many relationship between Recipe and Ingredient, capturing fields like *quantity* and *unit*.
- **Interaction**: Tracks user ratings and reviews for specific recipes. It references Django's default User (linked to a UserProfile) and a Recipe.
- **UserFavorite**, **UserKitchen**, **UserShoppingList**, and **UserCategory**: Support additional user-specific features, such as saved recipes, items in stock, shopping lists, and custom categories.

Each model corresponds to a table in the PostgreSQL database. Fields are indexed where high-volume queries are expected (e.g., *recipe_id*, *ingredient_name*, or *date*) to optimize performance.

Hosting and Migrations

Initial attempts to host the PostgreSQL database on free-tier platforms (Neon, Railway, Render) were hindered by strict storage limits. Ultimately, the project's tutor offered to host the database on a more flexible server.

Data Population

Given the dataset's size (millions of records in total) a single script approach was neither efficient nor easy to manage. Instead, three separate scripts handled distinct parts of the loading pipeline. Before running them, we verified that the final schema in PostgreSQL matched the expectations derived from the analysis scripts, using **DBeaver** to query and confirm table counts.

The first script is called **populate.py**. This script focuses on populating the core data structures (**Ingredients**, **Recipes**, **UserProfile**, and **RecipeIngredient**) while performing essential cleanup and handling duplicates.

- 1.– **Clearing Tables**: All related tables (RecipeIngredient, Recipe, Ingredient, UserProfile, and User) are truncated to start fresh.
- 2.– **Loading Ingredients**: Reads from *ingr_map.pkl*, where many duplicate IDs existed. Uses `bulk_create(...`,

ignore_conflicts=True) to insert thousands of records efficiently while skipping duplicates.

3.— **Loading Recipes and Contributors:** Reads RAW_recipes.csv for recipe information, converting lists from CSV fields (e.g., ingredients, steps) using ast.literal_eval. Creates a Django User and corresponding UserProfile for each unique contributor_id found in the CSV. Inserts recipes in batches of 5,000 to minimize memory overhead.

4.— **Mapping Recipes to Ingredients:** Reads PP_recipes.csv to retrieve the final set of ingredient_ids for each recipe, creating entries in the RecipeIngredient table in large batches.

During this phase, we also noted a few discrepancies in the data. For example, the original ingredient pickle listed 11 659 items, whereas only 8 023 were finally inserted due to conflict handling. Additionally, because many contributor_id values in the recipe file did not appear in PP_users.csv, the system ended up with more UserProfile records than there were user entries in that CSV, leaving some profiles without extra user information.

The second script is called **populate_user.py**. This script handles user-centric data beyond the basic user profile:

- **UserFavorite:** To ensure the user_favorites table contained meaningful entries, each user was automatically marked as a “favorite” all of the recipes they created. This approach guaranteed the table was not empty, providing data useful for later analysis and recommendations.
- **UserKitchen:** To ensure every user has at least one record in the user_kitchen table by default, each user is automatically associated with a “placeholder” ingredient (in this case, ingredient_id = 0). This placeholder corresponds to a simplified version of *Hershey’s semi-sweet baking chocolate* (labeled as *’s baking chocolate*), so that the user’s kitchen did not start empty.

Finally, the **Interaction** table (*ratings* and *reviews*) is populated from RAW_interactions.csv with our third script called **populate_interactions.py**:

- 1.— **Skipping Missing Users:** Interactions referencing user IDs not found in UserProfile are discarded. This resulted in omitting 418 825 records.
- 2.— **Batch Insertion:** Each batch of 5 000 interactions is inserted into the database to manage memory usage.
- 3.— **Confirmation:** After insertion, around 713 542 interactions remained valid and were successfully loaded.

To summarize, Table 3.1 shows the registers that were finally loaded into the DB, excluding the User_category which began empty.

Recipes	Ingredients	Recipe_ingredients	User_profile	User_kitchen	User_favorites	Interactions
231 637	8 023	1 602 903	27 926	27 926	231 637	713 542

Table 3.1: Every table in the Database with its corresponding number of loaded registers.

Additional Data Adjustments

After these scripts were run, further refinements became necessary to support application features:

- **Setting User Emails and Passwords:** A script was introduced to assign placeholder emails (e.g., `user_123@example.com`) to users who lacked one, and to encrypt their passwords via Django's `make_password`. This ensured compatibility with the standard Django authentication system.
- **UserCategory Initialization:** Another script was written to create default user categories for each user, ensuring that related application functionalities could operate without encountering empty or missing category data.

These post-population scripts reflect how real-world development often involves iterative changes. As the rest of the application took shape, new constraints emerged, prompting small but necessary updates to the data already in the database.

Important decisions and validation

Throughout the data loading process, the use of **large batch sizes** greatly accelerated performance, as it minimized the overhead of inserting records individually. In combination with the **`ignore_conflicts=True`** parameter that allowed the scripts to bypass errors caused by duplicate keys, although it was at the cost of some discrepancies from the original CSV totals.

The recipe dataset also included contributor IDs that did not appear in the user CSV, leading to skipped interactions and explaining why the final count of interactions is lower than the total found in the raw data. To reconcile CSV user IDs with Django's authentication system, a **custom `UserProfile` model** was introduced, **extending Django's built-in `User` model**. This one-to-one relationship maintained standard login mechanisms while supporting additional user attributes. After each data population script, queries run in **DBeaver** confirmed that table counts matched the script logs, ensuring data integrity and consistency with prior analyses.

By the end of these steps, the Dataset and Database module provided a robust PostgreSQL foundation of recipes, users, ingredients, and interactions. Although certain discrepancies such as missing user records and duplicated ingredient IDs were inevitable, the final dataset remains clean, efficient, and well-aligned with the Food.com domain.

3.4.2. API, Search, and Recommendation System Module

API Implementation

This API module uses **Django REST Framework (DRF)** [31] to provide a secure and structured communication layer between our PostgreSQL backend and the React and Vite frontend. DRF converts complex Django models into JSON using well-designed serializers and supports robust features

like custom viewsets and actions. To maintain stateless, scalable security, the API utilizes **JSON Web Tokens (JWT)** for authentication, ensuring that each request is associated with a verified user.

Serializers

The API employs a series of serializers such as *RecipeSerializer*, *UserFavoriteSerializer*, and *UserKitchenSerializer* to transform Django model instances into consumable JSON data. These serializers not only output basic recipe information but also compute additional fields like average ratings and rating counts, effectively bridging the gap between the raw database models and the frontend's display requirements.

Views and ViewSets

At the heart of our API, DRF's viewsets manage common operations such as listing, retrieving, updating, and deleting records, while custom actions enhance functionality. For example, the **kitchen-based** action recommends recipes based on the ingredients in a user's kitchen, and the **saved-based** action aggregates similarity scores for collaborative recommendations. In addition, the **filters-based** action allows users to choose filters based on text matching in names, descriptions, and tags, as well as criteria like ratings, preparation times, step counts, ingredient counts, and nutritional values, providing a versatile and customizable search experience.

User-Centric Endpoints

Specific endpoints have been developed to manage user-specific data. The **UserKitchenViewSet**, for instance, allows users to add ingredients to their kitchen along with associated details such as quantity, expiry date, and custom categories. Similarly, the **UserCategoryViewSet** empowers users to create and manage their own organizational categories (e.g., "Fruits & Veggies"), ensuring that each kitchen entry reflects their personal preferences.

Authentication and Accounts

Finally, the module integrates authentication using DRF's JWT mechanism. The accounts app includes custom serializers such as **RegisterSerializer** and **MyTokenObtainPairSerializer** to handle user registration and login. With every protected endpoint requiring a valid JWT token in the Authorization header, the system guarantees that only authenticated users can access or modify their data.

Search Implementation

The search functionality is designed to efficiently handle large datasets and deliver precise recipe results based on the ingredients users have added to their virtual kitchen. Users can add the ingredients they have at home into the application, and these ingredients are stored in the *UserKitchen* model. This ensures that recipe recommendations are personalized and closely aligned with the user's current inventory.

Virtual Kitchen Integration and Recipe Ranking

When a user accesses the kitchen-based endpoint, the system automatically retrieves the ingredients from their virtual kitchen. It compares these ingredients with those required by each recipe, using Django's *annotation* and *aggregation* functions to count the matching ingredients. This count is then used to rank the recipes. For example, if a user has three ingredients in their virtual kitchen, recipes that contain all three ingredients will appear at the top of the list, followed by recipes that only match two, and so on.

Ranking Based on Matching Ingredients

The ranking mechanism is a key part of this feature. Recipes are sorted by the number of matching ingredients in descending order. This means that the most relevant recipes (those that include the highest number of ingredients from the user's virtual kitchen) are prioritized and displayed first. This approach ensures that users see recipe suggestions that best match what they already have available.

Performance Optimization and Response

To efficiently manage the large dataset, pagination is implemented to limit the number of recipes returned per page. This minimizes processing overhead and ensures that the recommendations are delivered quickly. Once the recipes are ranked, they are serialized into JSON format and sent to the frontend, allowing for a smooth and responsive user experience.

Collaborative Filtering Implementation

In our recipe recommendation app, users maintain a personal collection of saved recipes. We assume saved recipes reflect genuine preference, and we use them to generate new suggestions via collaborative filtering. This functionality leverages (already known, but also new) user–recipe interaction data. The following sections detail the implementation, key components, and optimization steps taken to ensure efficient, high-quality recommendations.

Data Structures & Scripts

Our system begins with constructing a **sparse rating matrix** that captures user–recipe interactions (ratings and reviews), with rows representing recipes and columns representing users. Given our dataset of 199 191 recipes and 17 777 users, a dense matrix would be largely empty. To overcome this, we use the **Compressed Sparse Row (CSR)** format, which stores only nonzero entries, significantly reducing memory usage and allowing rapid matrix construction in RAM.

From Brute-Force to Scalable Similarity Computations

Early attempts at computing recipe similarities using a brute-force approach quickly proved infeasible. With nearly 200 000 recipes, calculating pairwise cosine similarities directly would have required tens of billions of comparisons, overwhelming both CPU and memory resources. Instead, building on

prior work that couples ALS and neighborhood methods [32] we implemented a 2-stage hybrid pipeline:

1. Dimensionality Reduction via ALS

We apply the **Alternating Least Squares (ALS)** algorithm (explained in Section 2.1.4) to the sparse user–recipe interaction matrix $199\,191 \times 17\,777$ by invoking the implicit library’s *AlternatingLeastSquares* class [33]. In the *train_als_model* function, an ALS instance is created with three hyperparameters: latent factors f , regularization λ , and iteration count N which together determine the embedding dimensionality, shrinkage strength, and number of alternating updates (the CSR-format recipe \times user matrix is transposed internally to match the library’s [33] user \times item expectation). During training, ALS alternates for N iterations between solving a regularized least-squares problem for all user-factor vectors $x_u \in \mathbb{R}^f$ (with Y fixed) and for all recipe-factor vectors $y_i \in \mathbb{R}^f$ (with X fixed). Upon completion, the learned embeddings appear in *model.item_factors* as an array of shape $(199\,191 \times f)$, with each row providing an f -dimensional representation of a recipe’s latent affinities. By embedding all n recipes into an f -dimensional space via ALS, whose item-factor update runs in $O(f^2N + f^3n)$ per (item-factor) sweep [18] we obtain one f -length vector per recipe. A naive all-pairs similarity search over n recipes costs $O(n^2f)$, but once the recipes are in \mathbb{R}^f , a single nearest neighbor query only requires $O(n \cdot f)$ and serves as the foundation for even faster Approximate Nearest-Neighbor lookups.

The choice of the hyperparameter values was guided by previous results from the recommendation community. We set the embedding dimensionality to $f = 64$ which falls within the 20–200 factor range originally advocated for implicit-feedback models by Hu et al [18] and reaffirmed in tuning guidelines by Dhama [34]. The iteration count was configured as $N = 15$, following a comment by B. Frederickson (the author of the implicit library) in a GitHub discussion on default hyperparameter values [35], a recommendation that is further corroborated by his performance benchmarks presented in [36]. Finally, the regularization parameter was fixed to $\lambda = 0.1$, matching with Dhama’s Stack Overflow recommendation, which reports this value as effective for most cases [34].

The full implementation of the *train_als_model* function is listed in A.1

2. Approximate Nearest-Neighbor Search with Annoy

Once we obtain the f -dimensional recipe embeddings, we construct an Annoy index [37]. In function *build_annoy_index*, Annoy builds a forest of random projection trees by repeatedly splitting the unit sphere with random hyperplanes, organizing points into small leaf buckets, this code snippet is shown in A.2. At query time, invoking *get_nns_by_item(i, k)* where k denotes the number of nearest neighbors to retrieve, the implementation of this function is included in A.3. It traverses each tree to assemble a candidate set in $O(\log N)$ time per query rather than $O(N)$ [38] and returns the k closest items by angular distance. The raw angular distance d is then transformed into a cosine-style similarity via:

$$\text{sim} = 1 - d \tag{3.1}$$

where d is the angular distance returned by Annoy, producing a top- k similar-recipe list in milliseconds (orders of magnitude faster than exhaustive pairwise scans). In this transformation, we exclude the query itself, otherwise an item would appear as its own neighbor.

In our implementation, the Annoy index is built with 10 random-projection trees (Spotify's recommended default [37]) and, at query time, the top-50 nearest neighbors ($k = 50$) are retrieved per recipe to balance lookup speed against recommendation quality.

Precomputed Similarity Table & Aggregated Recommendations

After obtaining the top- k ($k = 50$) similar recipes through Annoy, we store these relationships in a precomputed similarity table. This table logs each recipe's identifier along with the identifiers and similarity scores of its most similar recipes. At runtime, when a user accesses their recommendations, the system aggregates the similarity scores from all saved recipes and produces a ranked list of suggestions, filtering out recipes the user has already saved to ensure fresh recommendations.

Loader Performance & Verification

Due to the large volume of data (around 10 million rows), we optimized the loading process by converting the JSON data to CSV. This approach reduced the data load time from hours to minutes. Verification queries (using tools like DBeaver) were then run to confirm the row counts and detect any anomalies, ensuring that the similarity table accurately reflects the precomputed relationships.

3.4.3. Web Module

Frontend Operations and User Experience

Frontend Architecture and Component Organization

The frontend is developed using React's modular structure, which promotes the reuse of components and eases maintenance. The entry point of the application is an *App.jsx* file, where **React Router** is used to define distinct routes for major pages such as login, registration, main dashboard, recipe detail, kitchen management, shopping list, and various recommendation views. This method ensures smooth transitions across pages and effective state management via React hooks and context. For example, **React hooks** [39] are extensively utilized to manage state and side effects within the application. In many components, the *useState hook* is applied to track local state such as toggling a modal window open or closed when a user clicks a button. Similarly, the *useEffect hook* is used to handle side effects like fetching recipe details when a component mounts or refreshing the list of ingredients after an update, ensuring that the UI always displays the most recent data.

The code structure promotes **reusability** by centralizing components and utilities. The *sidebar* is a shared component that consistently displays the application's name and key navigation links, while the

authentication fetch utility handles secure API requests.

Dynamic User Interface and Interaction

The Web Module enhances user engagement with a **dynamic and responsive design**. Every component is styled using dedicated **CSS files** that introduce modern “glass” effects and responsive layouts, allowing the interface to adapt to a variety of device sizes. Interactive elements including modals, forms, and auto-complete features for ingredient and tag searches are integrated into each page. Recipe lists are paginated using a simple next/prev controls, ensuring fast, manageable browsing. [40]

Responsive Routing and Client-Side State Management

The application adopts a **single-page application (SPA)** model, taking full advantage of React Router for client-side routing. This minimizes the need for full page reloads, allowing users to transition quickly between the recipe detail view, main recommendations, and the kitchen view. Additionally, client-specific data such as search terms, active filters, and sorting options are **stored locally** (using tools like `localStorage`), ensuring that **user preferences persist across sessions**. This approach reduces server round-trips and enhances the overall responsiveness and continuity of the user experience.

Backend Integration and Performance

Integration with Django Backend and API

A significant strength of the Web Module lies in its seamless connection with a Django-powered REST API. The module uses a custom function, *fetchWithAuthentication*, which **secures every API call** by attaching JSON Web Tokens (JWT). This function also manages token refreshes automatically when tokens expire [41], ensuring uninterrupted and secure interactions.

Performance Optimization and Search Functionality Enhancements

Performance was an important goal for us. To meet this goal, both general and tag-based search functionalities underwent significant optimization. Initially, substring searches were handled via the *icontains operator*; however, to enhance performance, these were replaced with exact-match lookups using the ***__contains operator***. This change leverages PostgreSQL’s ***Generalized Inverted Index (GIN)*** [42] on the array fields storing tags, dramatically reducing search latency. Additionally, a dedicated tag suggestion endpoint utilizes PostgreSQL’s array and text search functions employing operators like *unnest* and *istartswith* to provide rapid and accurate auto-complete suggestions, thereby improving both the speed and relevance of the search results.

Security and Unified System Integration

Security, Error Handling, and User-Centric Feedback

Security is a core focus of the Web module. By integrating with Django’s authentication system,

every client-server interaction is safeguarded through the use of **JWTs**. The *AuthenticationFetch.jsx* file manages this process, including automated token refreshes to gracefully handle expired tokens. In addition, **error checking** is built into every API call so that if issues arise, such as failing to save a recipe or add an ingredient, the user is immediately informed via on-screen alerts, ensuring clear and user-friendly feedback.

Final Integration and Visual Consistency

The final Web module merges functionality with a clean, consistent design. The integration of React components and Django APIs results in a front-end that is interactive, secure, and responsive. Each element (from page layouts to modal dialogs) is designed for clarity and ease of use. Dedicated CSS ensures consistent visual cues that support smooth, scalable data flow and reinforce a cohesive visual identity throughout the application.

Visual Overview of Core Recommendation Flows

Figure 3.5 shows our “General search” interface, complete with the search bar, active filter chips (e.g., rating, time, ingredient limits), and nutrition sidebar controls. These dynamic controls let users instantly refine and explore hundreds of recipes. This screenshot belongs to one of our core recommendation flows, alongside “Recipes Based on Your Kitchen” in Figure B.3 and “Saved-Based recommendations” in Figure B.4.

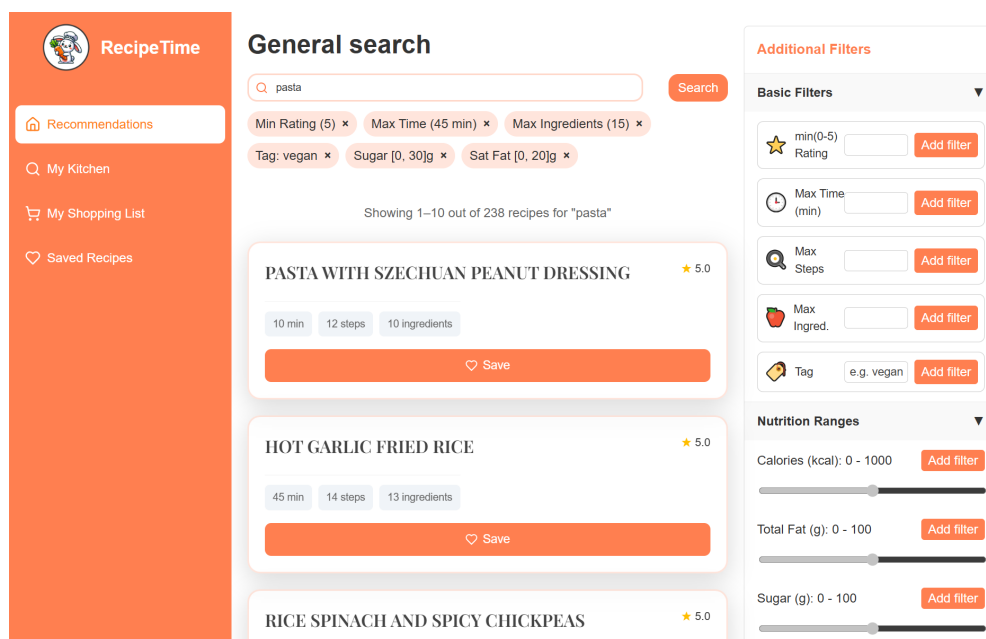


Figure 3.5: General search.

A complete set of additional UI screens is available in the Appendix, including: the Login page (Figure B.1), the Register page (Figure B.2), the Recommendations page (Figure B.5), the My Kitchen page (Figure B.6), the Shopping List page (Figure B.7), and the Saved Recipes page (Figure B.8).

EXPERIMENTS AND RESULTS

In this chapter, we present the design and execution of our user study, conducted to evaluate RecipeTime's core features and overall usability and present key conclusions drawn from participant feedback.

4.1. User Study

In this section, we describe the user study conducted to evaluate the usability of RecipeTime. We outline its objectives and methodology. Specifically, we sought to determine whether prospective users would actually use our recipe recommendation app, to assess the perceived usefulness of its recommendation features, and to gather actionable feedback and suggestions from a diverse sample.

We recruited sixteen native Spanish speakers through personal networks, university contacts, and community connections to ensure a diverse sample and accurate comprehension of the visual content. All participants were at least eighteen years old and possessed basic English level, but to facilitate comprehension the questionnaire was presented in Spanish. Consequently, all graphics and chart annotations appear in Spanish, even though the narrative and analysis below are provided in English for broader accessibility.

Each participant attended a single session on our computer while filling a Google Form on their phones. After providing informed consent, users completed an initial questionnaire capturing demographic data (age range, gender), cooking frequency, self-rated skill level, motivations for using a recipe application, and criteria for selecting recipes. Next, they were guided through a standardized series of tasks in RecipeTime: account registration and login, adding ingredients to “My Kitchen” and to the shopping list, exploring recipe recommendations based on both kitchen contents and saved recipes, performing a filtered search, viewing recipe details, saving and deleting recipes. After each task, participants rated its difficulty on a five-point scale and noted any confusing or unexpected behavior.

Upon completing the guided tasks, users provided global ratings for ease of use, navigation clarity, overall satisfaction, recommendation utility, and visual aesthetics, followed by open-ended questions about unclear concepts and desired improvements. Finally, they completed 10 statements for the System Usability Scale (SUS) to yield a standardized usability score. All responses were anonymous to

encourage sincere feedback. The sections that follow summarise the results for each questionnaire item and present the most relevant charts.

4.1.1. Consent

Before beginning the survey, we asked all participants to confirm that they had read and understood the study information and agreed to participate freely, voluntarily, and anonymously. As shown in Figure 4.1, all 16 respondents (100 %) checked the consent box, showing that every participant gave informed consent before continuing.

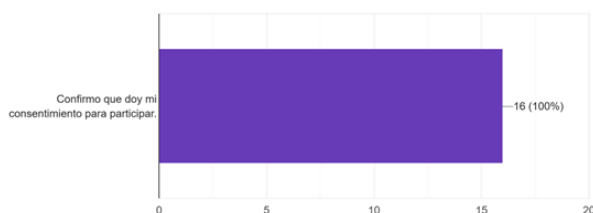


Figure 4.1: Consent bar chart.

4.1.2. Demographics & Cooking Background

We collected basic demographic and participants' own assessment of their cooking background from all 16 participants. Figures 4.2 – 4.5 summarize their profiles:

Age Distribution

Half of our sample fell into the youngest bracket, with 50 % aged 18–29. Another 12.5 % were between 30–47 years, and 37.5 % ranged from 48–65 years old. No one reported being over 65. As shown in Figure 4.2 our participants primarily consisted of younger and middle-aged adults.

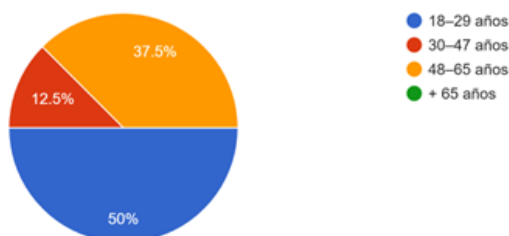


Figure 4.2: Age distribution.

Gender

A slight majority of respondents identified as male (56.3%), with the remaining 43.8% identifying as female. No one selected “Other” or “Prefer not to say.” As shown in Figure 4.3, the group was reasonably balanced but leaned towards male.

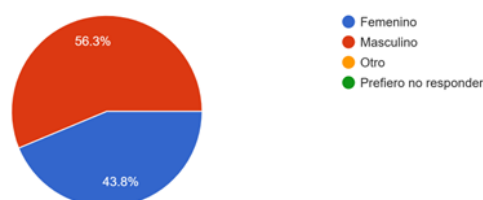


Figure 4.3: Gender distribution.

How Often They Cook

Cooking frequency varied (see Figure 4.4): 6.3% never cook, 12.5% cook rarely, 37.5% cook sometimes, 25% cook frequently, and 18.8% cook very frequently. This mix indicates both occasional home chefs and regular cooks in our sample.

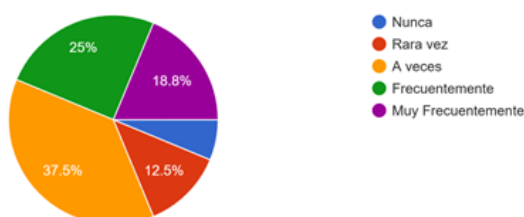


Figure 4.4: How often they cook.

Self-Assessed Kitchen Skill

When asked to rate their own kitchen level, 37.5% described themselves as beginners, 43.8% saw themselves as intermediate cooks, and 18.8% rated their skills as advanced. As shown in Figure 4.5, the majority of users consider themselves at least moderately skilled in the kitchen.

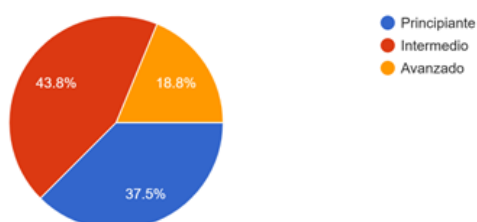


Figure 4.5: Kitchen skill.

4.1.3. Motivations & Preferences

Motivations for Using a Recipe Recommendation App

We asked participants to select up to three reasons why they would use a recipe recommendation app. As shown in Figure 4.6, the overwhelming majority (93.8 %) chose “Save time on meal planning”, making it by far the top motivation. The next most cited reasons were “Optimize use of ingredients I already have” (68.8 %) and both “Try new recipes” and “Learn to cook better” at 50 % each. Over one-third (37.5 %) indicated they wanted personalized suggestions, and no one selected “None.”

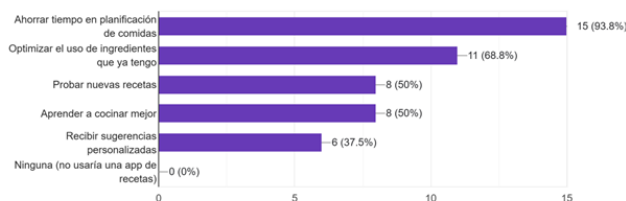


Figure 4.6: User motivations for using a recipe recommendation app.

Interest in Ingredient-Based Recommendations

We also asked whether users would like recipe suggestions based on the ingredients they have at home. As shown in Figure 4.7, 15 out of 16 participants (93.8 %) said yes, they would welcome that feature, while only one respondent (6.2 %) preferred to search on their own. This strong enthusiasm confirms that ingredient-driven recommendations align closely with user needs.

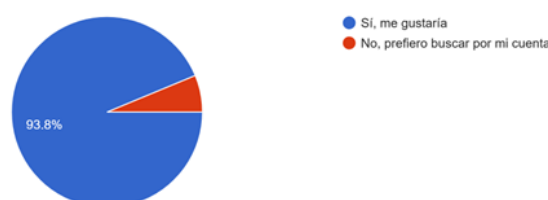


Figure 4.7: Interest in ingredient-based recommendations.

Factors Influencing Recipe Choice

Participants were also asked which factors they consider when choosing a recipe (multiple selections allowed). As shown in Figure 4.8, 100 % of respondents rank cooking time as a key criteria, followed closely by ease of preparation (93.8 %). Strong user reviews (“good ratings”) were important for 75 %, while only 25 % cited number of ingredients, number of steps, or familiarity with the dish as decision factors. This indicates that users choose recipes based more on quick, easy preparation and good reviews rather than on complexity or familiarity.

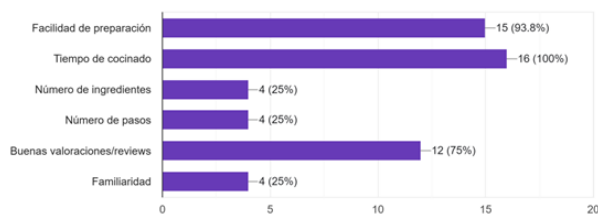


Figure 4.8: Recipe Selection Criteria.

4.1.4. Guided Task Walkthrough

In this section, all participants followed the same step by step path through the core features of RecipeTime, allowing us to observe how different users tackled identical tasks under identical conditions. After each task, beginning with account registration and continuing through kitchen setup, recipe search, filtering, and recipe detail. Respondents rated the difficulty on a 1 (“Very difficult”) to 5 (“Very easy”) scale and noted any points they found confusing or unexpected. We will now examine each task in detail, using numerical ratings alongside participants’ written feedback to give a clear view of overall usability and to identify specific challenges at every step of the user journey.

Login and Register

For the first task, participants were asked to create a new RecipeTime account and then log in. Most participants found registration and login effortless 100 % rated it “Very easy.” In the open responses, most either wrote “No” or left the question blank, and only one user suggested adding a “show password” toggle to verify their entry. As shown in Figure 4.9, every participant selected the highest usability score.

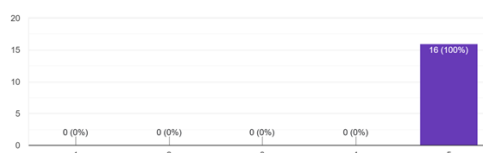


Figure 4.9: Login and register task.

My Kitchen

In the kitchen management stage, participants were asked to create a “Fruits and Vegetables” category in My Kitchen and add “avocado”, “banana”, and “melon”. The vast majority found this step straightforward 81.3% rated it “Very easy” and 18.8% rated it “Easy.” When asked if anything was confusing, 75 % (12 of 16) either left the field blank or said there were no issues. Only four users left comments, each suggesting a different tweak: one asked for a better sort order in the ingredient search, another that the ingredient slide originate directly from the name input box, a third requested support for decimal quantities, and a fourth noted surprise at the number of similar ingredient options. Overall,

kitchen setup was intuitive, with just a few minor interface refinements needed as seen in Figure 4.10.

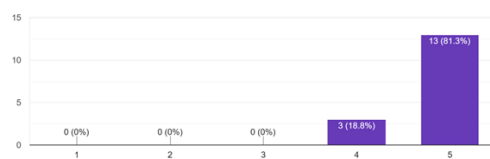


Figure 4.10: My kitchen task.

My Shopping List

In the Shopping List step, participants were instructed to add three items to their list, mark one as purchased, and then remove it. This proved straightforward: 75 % (12/16) rated it “Very easy” (5) and the remaining 25 % (4/16) rated it “Easy” (4). Here again, only four participants left any comments, suggesting for example, adding a legend to the table, making the checkbox more visually clear, providing a filter for purchased items, or improving the search function. As shown in Figure 4.11, all ratings fell at the top end of the scale, reflecting a uniformly simple experience.

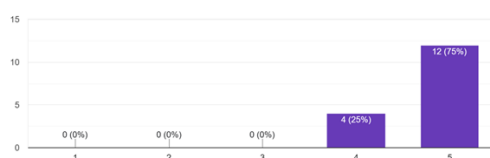


Figure 4.11: My shopping list task.

Recommendations based on your kitchen

Participants then navigated to the Recommendations section, chose the “I want to use my own ingredients” button and sorted results by the fewest ingredients. Ease remained high: 87.5 % rated it a 5 (“Very easy”) and 12.5 % rated it a 4 (“Easy”). When asked if anything was confusing or unexpected, only two users left suggestions, one asked to move the Sort control below the main search bar, and another recommended displaying scores with different colors. The rest either wrote “No” or left the field blank. As shown in Figure 4.12, all difficulty ratings sit at the top end of the 1–5 scale.

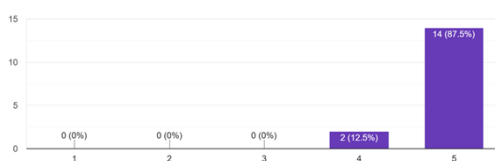


Figure 4.12: Recommendations kitchen-based page.

Save recipe

In the Save Recipe step, users were asked to select four recipes they liked and save them from the current page. This action was nearly effortless: 93.8% (15/16) rated it a 5 (“Very easy”) and the remaining 6.3% (1/16) rated it a 4 (“Easy”). When asked if anything was confusing or unexpected, the majority reported no problems, and only one user suggested enhancing the rating display by using the style Amazon uses with colored stars filled proportionally to the average score. As shown in Figure 4.13, all difficulty scores fall at the top end of the 1–5 scale.

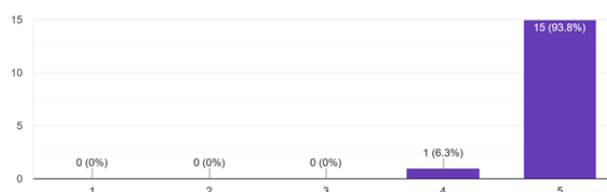


Figure 4.13: Save recipe task.

Delete recipe

In the Delete Recipe step, users were asked to go to their saved recipes and remove one of their choice. Most participants found deleting a recipe straightforward: 68.8% (11/16) rated it a 5 (“Very easy”), 12.5% (2/16) a 4, and 18.8% (3/16) a 3. The vast majority reported no problems, and only four users left comments, each mentioning difficulty spotting or understanding the delete control and suggesting a clearly labeled “Delete” button or familiar bin icon. In later conversations, participants who use social media apps frequently said the removal gesture felt intuitive and natural, while those less accustomed to such interfaces preferred an explicit delete button to confirm their action. As shown in Figure 4.14, most ratings sit at the top end of the 1–5 scale.

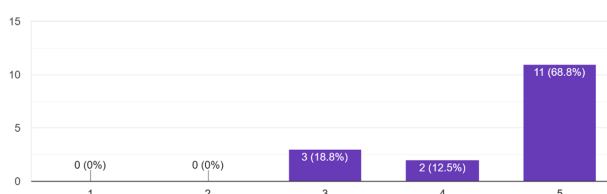


Figure 4.14: Delete recipe task.

Recommendations based on user’s saved recipes

In the “Recipes Based on Your Saved Recipes” task, participants navigated to Recommendations, selected the option “I want to see recipes I’m sure I will like,” and moved to page 2 to view new suggestions. All 16 users (100%) rated this step “Very easy” (5/5). When asked if anything seemed confusing or unexpected, 7 out of 9 respondents (77.8%) reported no problems, and only one user commented pointing out that many similarity scores appeared as “100%”. As shown in Figure 4.15,

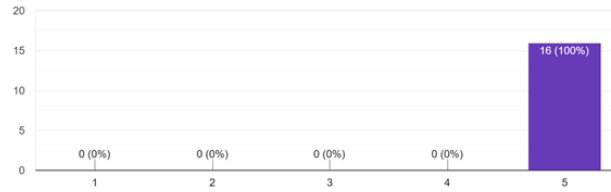


Figure 4.15: Recommendations saved-based page.

General Search

In the General Search task, users went to Recommendations, selected “I want to discover new recipes,” searched for “pasta,” applied the filters (minimum rating 5, max time 45 min, max ingredients 15, vegan tag), adjusted nutritional values, and clicked the Search button. Ease remained high: 81.3 % (13/16) rated it “Very easy” and 18.8 % (3/16) rated it “Easy.” When asked if anything was confusing or unexpected, 45.5 % of respondents (5/11) reported no issues, and the others offered individual suggestions. One asked for an explanation that filters require clicking “Add,” another wanted more filter options to apply automatically as they type, a third proposed clearer drop down icons, a fourth suggested listing example values for the tags, and a fifth recommended a combined nutrition filter rather than separate ones. As shown in Figure 4.16, all difficulty ratings lie at the easy end of the 1–5 scale.

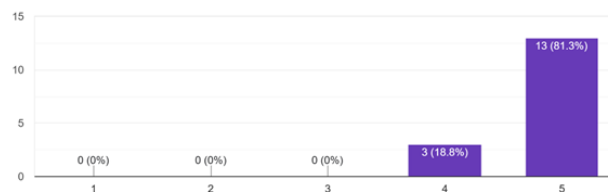


Figure 4.16: General search task.

Recipe Detail

In the Recipe Detail step, participants selected a recipe and had to find its reviews. The vast majority found this straightforward 87.5 % (14/16) rated it “Very easy” and 12.5 % (2/16) rated it “Easy.” When asked if anything was confusing or unexpected, half of respondents reported no issues (writing “No” or leaving the field blank). The other five users offered feedback: one asked for a dedicated “Details” button to access full review text, another pointed out that you must click the recipe title to open the recipe detail, a third said all the text was in lowercase making it hard to read. One user commented that they did not understand the names of the users that were already in the reviews, this is because the dataset we used did not contain specific usernames, so IDs like user_1, user_2, etc, were assigned. A fifth complimented the thoroughness of the reviews and the visual aspect of this page. As shown in Figure 4.17, all difficulty scores sit at the top end of the 1–5 scale.

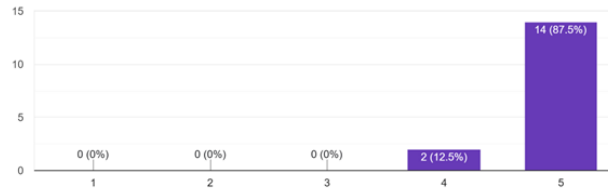


Figure 4.17: Recipe detail task.

4.1.5. Post-Test Experience & Feedback

After completing the task walkthrough, we asked participants to reflect on their overall interaction with RecipeTime. In this section, users rated key aspects of the app like ease of use, navigational clarity, overall satisfaction, usefulness of recommendations, visual design, and clarity of the various score metrics on a 1–5 star scale. They also indicated whether the Kitchen Score, Recipe Score, and Similarity Score were clear, and were invited to share any further improvement ideas. Their responses here (summarized in Table 4.1) provide a comprehensive view of the user experience and help pinpoint areas for refinement before moving on to the formal SUS questionnaire.

Ease of use	Clarity of navigation	Overall Satisfaction	Usefulness of recommendations	Application aesthetics
4.94	5.00	4.94	4.69	4.88

Table 4.1: Post-test experience summary.

Ease of use

Participants found the app remarkably easy to use: 15 out of 16 respondents (93.8 %) awarded it the maximum 5-star rating, and the remaining user gave it 4 stars (6.3 %). There were no ratings below 4, yielding an overall average for the ease of use score of 4.94 out of 5.

Clarity of navigation

Participants unanimously agreed that the app's navigation was crystal clear: 100 % of respondents (16/16) awarded it a 5-star rating, resulting in a perfect average score of 5.00 out of 5.

Overall Satisfaction

Participants expressed very high overall satisfaction with the app: 15 out of 16 respondents (93.8 %) awarded it 5 stars, while the remaining user (6.3 %) gave it 4 stars. Resulting in an average rating of 4.94 out of 5.

Usefulness of recommendations Participants found the recipe suggestions highly valuable: 75 % (12/16) awarded 5 stars, 18.8 % (3/16) gave 4 stars, and 6.3 % (1/16) rated it 3 stars. This led to an average usefulness score of 4.69 out of 5.

Application aesthetics

Participants rated the application's visual design very highly: 87.5 % (14/16) awarded it 5 stars, and the remaining 12.5 % (2/16) gave 4 stars. This resulted in an average aesthetics score of 4.88 out of 5.

Clarity of Scoring Metrics

We asked whether the Kitchen Score, Recipe Score, and Similarity Score were clear to users. As shown in Figure 4.18, 87.5 % of participants (14/16) answered Yes, indicating they understood all three metrics, while 12.5 % (2/16) answered No. Of those who found them unclear, one respondent commented that the Similarity Score was confusing, specifically, they were not sure how it was calculated or how to interpret differences between values.

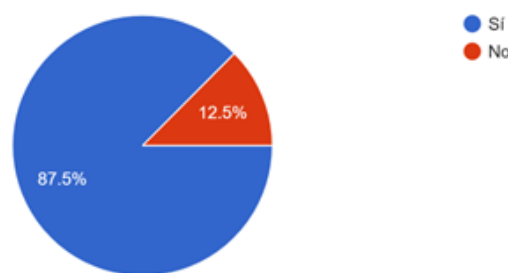


Figure 4.18: Clarity of scoring metrics.

Improvements for RecipeTime

Throughout the walkthrough, participants flagged many opportunities for refinement. The following suggestions are additional ideas gathered in the final survey. Participants suggested adding more food imagery (recipes and ingredient photos), an FAQ/help page for score definitions, an Easy/Pro mode switch to simplify or expand metrics, a background music toggle, online availability, and small UI touches such as nutrition emoticons.

Additional Feature Ideas

When asked what they would love to see in any recipe app, users again asked for photos of recipes, plus ideas like a user profile section, recipe difficulty labels, the ability to add personal recipes, and an interactive “fridge” view to filter by ingredient storage (e.g., clicking on the freezer to only show frozen items).

4.1.6. Measuring Usability with the SUS

To quantify overall usability we employed the System Usability Scale (SUS). It was originally developed by John Brooke in 1986 and described in detail in his 1995 article [43]. Brooke designed SUS to be a rapid, reliable, context-independent tool for assessing global usability. He derived its ten items

by having expert users rate two prototype systems (one very easy and one very difficult) and selecting the statements that produced the most extreme, consistent responses. By balancing five positively worded items (e.g., “I thought the application was easy to use”) with five negatively worded items (e.g., “I found the application unnecessarily complex”), SUS counters response bias while covering key aspects of effectiveness, efficiency, and satisfaction. Each statement is rated on a 5 point Likert scale from 1 (“Strongly disagree”) to 5 (“Strongly agree”), yielding a single score from 0–100 for overall usability.

Scoring Procedure:

After collecting each participant’s raw ratings, we converted them into a 0–4 range. For positively worded items (1, 3, 5, 7, 9), we subtracted 1 from the raw score. For negatively worded items (2, 4, 6, 8, 10), we computed 5 minus the raw score. This produced ten adjusted scores between 0 and 4, which we summed and then multiplied by 2.5 to yield a 0–100 SUS score.

Example calculation:

A participant’s raw responses might be (5, 1, 5, 1, 4, 1, 4, 2, 5, 1). After converting each item, the adjusted scores become (4, 4, 4, 4, 3, 4, 3, 3, 4, 4). Summing these gives 37, and multiplying by 2.5 yields a SUS score of 92.5 for that participant.

Once all sixteen scores were calculated, we computed the mean and standard deviation to assess central tendency and variation. Our sample produced an average SUS score of 92.5 with a standard deviation of approximately 4.3. According to the adjective ratings developed by Bangor et al [44], our average SUS score of 92.5 falls into the “Excellent” usability category, confirming that RecipeTime delivers an outstanding user experience and a low SD in our case 4.3 means most participants gave very similar scores, indicating a consistent experience across our sample.

4.2. Analysis of results

Integrating usability testing into our development process proved exceptionally enriching. Usability is a complex discipline, where even minor interface details (often overlooked in early development) can significantly affect user engagement. Keeping our users’ needs at the forefront and striving to make the interface both intuitive and enjoyable has been essential to our application. We are proud to report that this emphasis on user centered design yielded a System Usability Scale score of 92.5, an excellent result that met our expectations and affirmed the application’s overall usability.

We found that observing actual users interact with RecipeTime was both highly informative and deeply reassuring of our work. From the very first questions, we gained valuable insight into who our users really are: their ages, cooking habits, and self-rated skill levels helped us assess whether RecipeTime should aim for a broad audience or target a specific demographic. Discovering that “saving time on meal planning” and “optimizing ingredients I already have” were the top motivations of ours users to

use a recipe application validated the importance of our “Fewer Minutes” sort option and the ingredient-based recommendation feature. Likewise, offering users the ability to sort by Kitchen Score (maximizing use of existing ingredients) or Recipe Score (minimizing additional purchases) proved directly aligned with their needs.

The guided task walkthrough proved equally enriching for us. By breaking the experience into small, focused steps, participants felt comfortable experimenting and offering creative suggestions. We originally assumed that toggling a recipe in and out of the user’s saved recipes collection via the same “Save” and “Saved” button (just like social media interactions with the “like” toggle) would feel intuitive for both adding and removing items. However, during testing many participants hesitated when there was no trash bin icon or an explicit “Delete” label. Younger participants completed this task with ease, perhaps reflecting their familiarity with social media, while other users strongly preferred for a clearly marked “Delete” button for removal. Similarly, the dark checkboxes in the shopping list, while aesthetically pleasing, lacked sufficient contrast. We will replace them with lighter, more distinct controls to improve clarity.

Toward the end, we invited participants to share their thoughts, allowing users to give suggestions without limits for future development. Several participants requested a “Show Password” toggle on the login screen, a dedicated FAQ explaining the metric Scores and the functionality behind the recommendations, and the ability to “Delete All” purchased items in the shopping list with a single action, and deployment of the application as an accessible online service. Others recommended to facilitate viewing the recipe details by enlarging the area on recipe cards where you can click or adding a “Details” button. More ambitious requests such as adding recipe imagery, personalized user profiles, difficulty badges, an original soundtrack that embodies the app and plays softly in the background and a visually interactive kitchen where users could click on the “fridge” for example and only see frozen ingredients.

We are very grateful for all the time and cooperation of all of the participants in our study, thanks to them we collected incredible recommendations and user feedback that could be implemented in future versions.

CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

This thesis aimed to reduce food waste and enable more efficient home cooking by delivering personalized recipe suggestions that balance taste, nutrition, and individual dietary restrictions. We wanted to address a clear gap in the current recommender system landscape.

To bring this vision to life, we developed **RecipeTime**, a full-stack application featuring three complementary recommendation strategies. The **Kitchen-Based Recommender** identifies recipes that make the most of ingredients you already have, cutting down on unnecessary purchases and spoilage. Meanwhile, the **Collaborative-Filtering Recommender** leverages users' saved recipes and the preferences of similar users to surface new dishes they are likely to enjoy, promoting culinary exploration. The **Filter-Driven Recommender** offers fine-grained control through a variety of parameters like minimum rating, maximum preparation time in minutes, maximum number of steps, maximum number of ingredients, chosen dietary tags (e.g., diabetic), and set nutritional thresholds (for example, capping sugar at 50 g) enabling highly customized meal planning. Underlying these features is a rich dataset of over 180 000 recipes and 700 000 implicit user interactions, which has been instrumental in enhancing both the quality and diversity of its recommendations.

Throughout development and informal user testing, RecipeTime demonstrated rapid response times and users reported that the tailored suggestions streamlined their meal planning process. Leveraging such a large, high quality dataset proved invaluable, yet we also uncovered several areas for refinement. Unfiltered tag searches can become sluggish when scanning 180 000+ recipes. Dataset limitations prevented the addition of useful filters like recipe difficulty or explicit "healthy" labels. We recognized that the absence of recipe photographs detracted from the user experience and may have limited engagement and diminished visual appeal. Additionally, because of time limitations, RecipeTime's scope was confined to recommending existing recipes rather than allowing users to contribute their own. Although we endeavored to create a polished, intuitive interface, users who are long habituated to the seamless experiences of large commercial platforms held high expectations. Nevertheless, every participant found RecipeTime useful and was able to navigate it with ease.

In completing this project, I was able to consolidate the wide array of skills I acquired throughout my degree from academic research and dataset curation to backend API development and frontend integration. The end-to-end system design process strengthened my expertise in data engineering, machine learning, and user centered design. Above all, RecipeTime represents a tangible step toward more sustainable food practices by encouraging users to cook with what they already own and thereby reduce waste. It is my hope that this work contributes meaningfully to both home cooking efficiency and the broader goal of planetary stewardship.

5.2. Future Work

Future work could explore automated inventory tracking by letting users upload grocery receipts or photos to keep ingredient quantities up to date. Uploading a receipt [45] or a simple photo [46] of kitchen shelves would update which items are on hand and how much remains. A new recommendation mode could then generate full recipes based entirely on the current stock, optimizing for taste, nutrition, and minimal waste.

To create a true cooking community, RecipeTime could introduce user profiles that let cooks submit their own recipes and then rate, review, and upload photos of the dishes they make. During recipe submission, contributors would complete required fields like tagging with difficulty level, healthy or allergy-friendly labels, and child-friendly or adult-only labels so every recipe is easy to discover and well matched to individual needs. App messaging and discussion forums would give cooks a space to ask questions, swap tips, and connect over a shared love of cooking [47].

Additional application improvements could involve an integrated meal planner calendar, allowing recipes to be assigned to specific days or mealtimes and automatically generate shopping lists [48]. Accessibility features such as text to speech recipe narration for hands free cooking and themes designed for users with color blindness would ensure that everyone can interact comfortably. A dedicated section could include an embedded Google Map to pinpoint nearby supermarkets and, when available, display the price, helping users plan shopping trips more efficiently. Alongside that, an “Insights & News” panel could curate timely articles on sustainable eating, food waste reduction tips, and seasonal produce guides, raising awareness and inspiring users to cook and shop more responsibly [49].

Future studies could assess graph-based, transformer-driven, or context-aware recommendation approaches to identify which best serve new users, changing dietary needs, and sustainability goals. Finally, presenting different feature designs to separate user groups and analyzing their choices could provide quantitative data on RecipeTime’s real world impact on reducing food waste, improving nutrition, and boosting overall satisfaction [50].

BIBLIOGRAPHY

- [1] G. FAO *et al.*, “Global food losses and food waste—extent, causes and prevention,” *SAVE FOOD: An initiative on food loss and waste reduction*, vol. 9, p. 2011, 2011.
- [2] K. Jaglo, S. Kenny, and J. Stephenson, “From farm to kitchen: The environmental impacts of us food waste,” *US Environmental Protection Agency Office of Research and Development*, pp. 1–113, 2021.
- [3] Unicef *et al.*, “The state of food security and nutrition in the world 2024,” 2024.
- [4] S. S. Gropper, “The role of nutrition in chronic disease,” *Nutrients*, vol. 15, no. 3, 2023.
- [5] F. Ricci, L. Rokach, and B. Shapira, eds., *Recommender Systems Handbook*. Springer US, 2022.
- [6] Y. Koren, R. M. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [7] A. van den Oord, S. Dieleman, and B. Schrauwen, “Deep content-based music recommendation,” in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States* (C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2643–2651, 2013.
- [8] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001* (V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, eds.), pp. 285–295, ACM, 2001.
- [9] D. Elsweiler, H. Hauptmann, and C. Trattner, “Food recommender systems,” in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 871–925, Springer US, 2022.
- [10] J. N. Bondevik, K. E. Bennin, Önder Babur, and C. Ersch, “A systematic review on food recommender systems,” *Expert Systems with Applications*, vol. 238, p. 122166, 2024.
- [11] C. Trattner and D. Elsweiler, “An evaluation of recommendation algorithms for online recipe portals,” in *Proceedings of the 4th International Workshop on Health Recommender Systems co-located with the 13th ACM Conference on Recommender Systems 2019 (RecSys 2019), Copenhagen, Denmark, September 20, 2019* (D. Elsweiler, B. Ludwig, A. Said, H. Schäfer, H. Torkamaan, and C. Trattner, eds.), vol. 2439 of *CEUR Workshop Proceedings*, pp. 24–28, CEUR-WS.org, 2019.
- [12] C. Trattner and D. Elsweiler, “Investigating the healthiness of internet-sourced recipes: implications for meal planning and recommender systems,” in *Proceedings of the 26th international conference on world wide web*, pp. 489–498, 2017.
- [13] B. P. Majumder, S. Li, J. Ni, and J. J. McAuley, “Generating personalized recipes from historical user preferences,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-*

- IJCNLP 2019, Hong Kong, China, November 3-7, 2019* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), pp. 5975–5981, Association for Computational Linguistics, 2019.
- [14] C.-Y. Teng, Y.-R. Lin, and L. A. Adamic, “Recipe recommendation using ingredient networks,” in *Proceedings of the 4th annual ACM web science conference*, pp. 298–307, 2012.
- [15] J. Freyne and S. Berkovsky, “Intelligent food planning: personalized recipe recommendation,” in *Proceedings of the 15th international conference on Intelligent user interfaces*, pp. 321–324, 2010.
- [16] Y.-Y. Ahn, S. E. Ahnert, J. P. Bagrow, and A.-L. Barabási, “Flavor network and the principles of food pairing,” *Scientific reports*, vol. 1, no. 1, p. 196, 2011.
- [17] S. Hausmann, O. Seneviratne, Y. Chen, Y. Ne’eman, J. V. Codella, C. Chen, D. L. McGuinness, and M. J. Zaki, “Foodkg: A semantics-driven knowledge graph for food recommendation,” in *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II* (C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. F. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, eds.), vol. 11779 of *Lecture Notes in Computer Science*, pp. 146–162, Springer, 2019.
- [18] Y. Hu, Y. Koren, and C. Volinsky, “Collaborative filtering for implicit feedback datasets,” in *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pp. 263–272, IEEE Computer Society, 2008.
- [19] X. He, H. Zhang, M. Kan, and T. Chua, “Fast matrix factorization for online recommendation with implicit feedback,” *CoRR*, vol. abs/1708.05024, 2017.
- [20] S. Li, “Food.com recipes and interactions.” <https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions>., 2019. Last accessed: Feb 2025.
- [21] P. Mooney, “Recipenlg.” <https://www.kaggle.com/code/paultimothymooney/explore-recipe-nlg-dataset>, 2021. Last accessed: Feb 2025.
- [22] R. Figueroa, “Epicurious - recipes with rating and nutrition.” <https://www.kaggle.com/datasets/hugodarwood/epirecipes>, 2017. Last accessed: Feb 2025.
- [23] Yummly, “Recipeingredients dataset.” <https://www.kaggle.com/datasets/kaggle/recipe-ingredients-dataset>, 2017. Last accessed: Feb 2025.
- [24] onzie9, “Allrecipes.com dataset.” https://github.com/onzie9/all_recipes_data, 2019. Last accessed: Feb 2025.
- [25] A. Morales-Garzón, O. A. Rocha, S. Benel Ramirez, G. Tuco Casquino, and A. Medina, “Recetasdelaabuela dataset.” <https://huggingface.co/datasets/somosnlp/RecetasDeLaAbuela/blob/main/README.md?code=true%23L26>, 2024. Last accessed: Feb 2025.
- [26] S. (international community), “Recetascocina dataset.” <https://huggingface.co/datasets/somosnlp/recetas-cocina>, 2024. Last accessed: Feb 2025.
- [27] P. A. Crystal, David, “Spoonacular food api.” <https://spoonacular.com/food-api>, 2024. Last accessed: Feb 2025.

-
- [28] P. A. Crystal, David, "Edamam. recipe search api." <https://www.edamam.com/>, 2024. Last accessed: Feb 2025.
- [29] E. Schacht, "Food.com eda and text analysis." <https://www.kaggle.com/code/etsc9287/food-com-eda-and-text-analysis/report>, 2019. Last accessed: Feb 2025.
- [30] R. Decal, "Modulenotfounderror: No module named 'pandas.core.indexes.numeric' using metaflow." <https://stackoverflow.com/questions/75953279/modulenotfounderror-no-module-named-pandas-core-indexes-numeric-using-met>, 2019. Last accessed: Dec 2024.
- [31] T. Christie, "Django rest framework." <https://www.django-rest-framework.org/>, 2019. Last accessed: Feb 2025.
- [32] Y. Zhou, D. M. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *Algorithmic Aspects in Information and Management, 4th International Conference, AAIM 2008, Shanghai, China, June 23-25, 2008. Proceedings* (R. Fleischer and J. Xu, eds.), vol. 5034 of *Lecture Notes in Computer Science*, pp. 337–348, Springer, 2008.
- [33] B. Frederickson, "Implicit library documentation." <https://benfred.github.io/implicit/>, 2022. Last accessed: March 2025.
- [34] G. Dhama, "Tuning hyper parameters als model." <https://stackoverflow.com/questions/48779687/tuning-hyper-parameters-als-model/49161421#49161421>, 2018. We used this reference for the default latent factors and regularization parameters. Last accessed: May 2025.
- [35] B. Frederickson, "Value for regularization parameter, and other hyperparameters." <https://github.com/benfred/implicit/issues/100>, 2018. We used this reference for the default number of iterations. Last accessed: May 2025.
- [36] B. Frederickson, "Faster implicit matrix factorization." <https://www.benfrederickson.com/fast-implicit-matrix-factorization/>, 2016. Last accessed: May 2025.
- [37] E. Bernhardsson, "Spotify annoy." <https://github.com/spotify/annoy>, 2016. Last accessed: May 2025.
- [38] Z. Ye, "time complexity of popular ann." <https://milvus.io/ai-quick-reference/what-is-the-typical-time-complexity-of-popular-ann-approximate-nearest-ne>, 2025. Last accessed: May 2025.
- [39] S. Alpert, D. Abramov, and R. Florence, "Built-in react hooks." <https://react.dev/reference/react/hooks>, 2019. Last accessed: Feb 2025.
- [40] T. Christie, "Pagination." <https://www.django-rest-framework.org/api-guide/pagination/>, 2019. Last accessed: May 2025.
- [41] D. Arias and S. Bellen, "What are refresh tokens and how to use them securely." <https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/>, 2021. Last accessed: May 2025.

- [42] T. Sigaev and O. Bartunov, "Gin index." <https://www.postgresql.org/docs/current/gin.html>, 2006. Last accessed: Feb 2025.
- [43] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.
- [44] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.
- [45] J. Antonio, A. R. Putra, H. Abdurrohman, and M. S. Tsalasa, "A survey on scanned receipts ocr and information extraction," in *Proceedings of the International Conference on Document Analysis and Recognit, Jerusalem, Israel*, pp. 29–30, 2022.
- [46] Y. Ji, H. Plourde, V. Bouzo, R. D. Kilgour, and T. R. Cohen, "Validity and usability of a smartphone image-based dietary assessment app compared to 3-day food diaries in assessing dietary intake among canadian adults: randomized controlled trial," *JMIR mHealth and uHealth*, vol. 8, no. 9, p. e16953, 2020.
- [47] S. Gross, A. Toombs, J. Wain, and K. Walorski, "Foodmunity: designing community interactions over food," in *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Extended Abstracts Volume, Vancouver, BC, Canada, May 7-12, 2011* (D. S. Tan, S. Amershi, B. Begole, W. A. Kellogg, and M. Tungare, eds.), pp. 1019–1024, ACM, 2011.
- [48] C. E. Mauch, T. P. Wycherley, R. A. Laws, B. J. Johnson, L. K. Bell, and R. K. Golley, "Mobile apps to support healthy family food provision: systematic assessment of popular, commercially available apps," *JMIR mHealth and uHealth*, vol. 6, no. 12, p. e11867, 2018.
- [49] R. A. Khot and F. F. Mueller, "Human-food interaction," *Found. Trends Hum. Comput. Interact.*, vol. 12, no. 4, pp. 238–415, 2019.
- [50] Y. Zhang and X. Chen, "Explainable recommendation: A survey and new perspectives," *CoRR*, vol. abs/1804.11192, 2018.

APPENDICES

CODE FRAGMENTS

Code A.1: Applying the ALS algorithm from the implicit library

```
1 def train_als_model(sparse_ratings, factors=64, iterations=15, regularization=0.1):
2     """
3     Train an ALS model using the implicit library to obtain low-dimensional embeddings.
4     Returns the trained ALS model.
5     """
6     # The implicit library expects a (users x items) matrix.
7     # Our matrix is (recipes x users), so we transpose it.
8     model = implicit.als.AlternatingLeastSquares(
9         factors=factors,
10        regularization=regularization,
11        iterations=iterations,
12        calculate_training_loss=True
13    )
14    model.fit(sparse_ratings.transpose())
15
16    return model
```

Code A.2: Building an Annoy Index

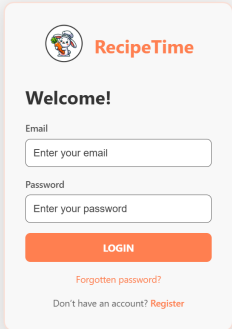
```
1 def build_annoy_index(item_factors, n_trees=10):
2     """
3     Build an Annoy index from the given item embeddings (item_factors).
4     item_factors shape: (num_recipes, embedding_dim)
5     Returns the built Annoy index.
6     """
7
8     num_items, dim = item_factors.shape
9     ann_index = AnnoyIndex(dim, 'angular')
10
11     for i in range(num_items):
12         ann_index.add_item(i, item_factors[i])
13         if i > 0 and i % 10000 == 0:
14
15     ann_index.build(n_trees)
16
17     return ann_index
```

Code A.3: Retrieve top k nearest neighbors

```
1 def compute_topk_similarities_annoy(ann_index, recipe_ids, k=200):
2     """
3     Compute the top-k similar recipes for each recipe using the Annoy index.
4     Returns a dict: recipe_id -> list of (similar_recipe_id, similarity_score).
5     """
6
7     num_recipes = len(recipe_ids)
8     top_k_similarities = {}
9
10    for i in range(num_recipes):
11        # get_nns_by_item returns indices of the nearest neighbors
12        neighbor_indices = ann_index.get_nns_by_item(i, k + 1)
13
14        neighbor_indices.remove(i)
15
16        similar_list = []
17
18        for neighbor_idx in neighbor_indices:
19            dist = ann_index.get_distance(i, neighbor_idx)
20            sim_score = 1.0 - dist
21            similar_list.append((int(recipe_ids[neighbor_idx]), sim_score))
22
23        top_k_similarities[int(recipe_ids[i])] = similar_list
24
25    return top_k_similarities
```

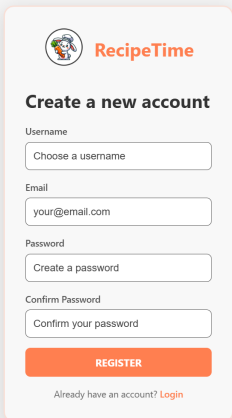
WEB APPLICATION

B.1. Authentication



The login page features the RecipeTime logo at the top left. Below it is a "Welcome!" heading. The form includes an "Email" field with the placeholder "Enter your email", a "Password" field with the placeholder "Enter your password", and an orange "LOGIN" button. At the bottom, there are two links: "Forgotten password?" and "Don't have an account? Register".

Figure B.1: Login page.



The register page features the RecipeTime logo at the top left. Below it is a "Create a new account" heading. The form includes a "Username" field with the placeholder "Choose a username", an "Email" field with the placeholder "your@email.com", a "Password" field with the placeholder "Create a password", and a "Confirm Password" field with the placeholder "Confirm your password". At the bottom, there is an orange "REGISTER" button and a link "Already have an account? Login".

Figure B.2: Register page.

B.2. Recommendations Flows

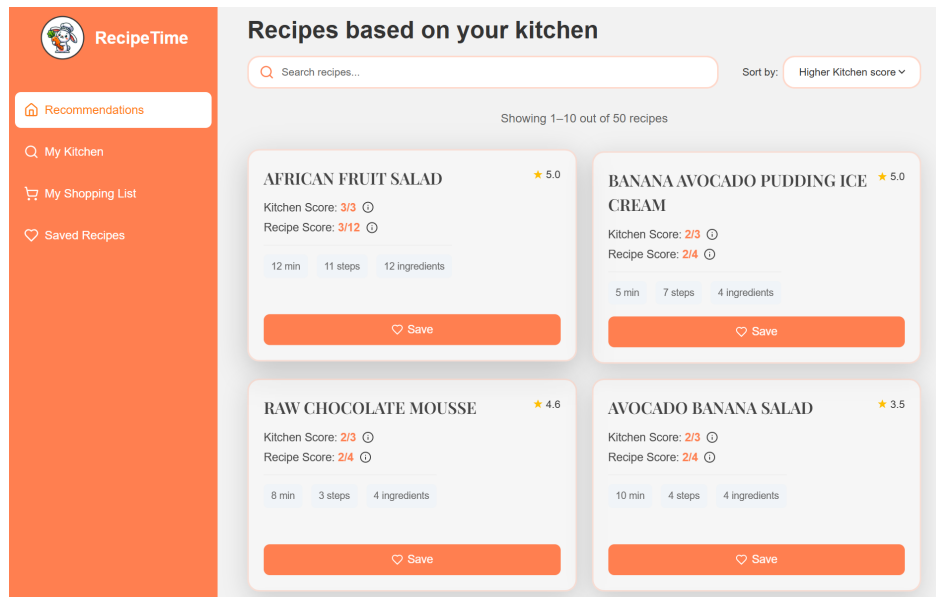


Figure B.3: Recipes based on your kitchen page.

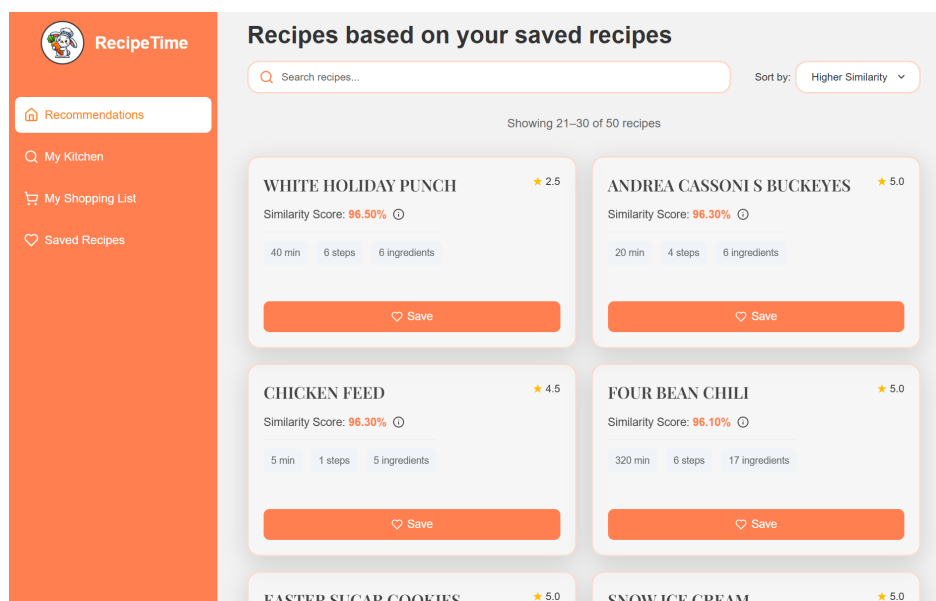


Figure B.4: Recipes based on your saved recipes page.

B.3. User Personal Space

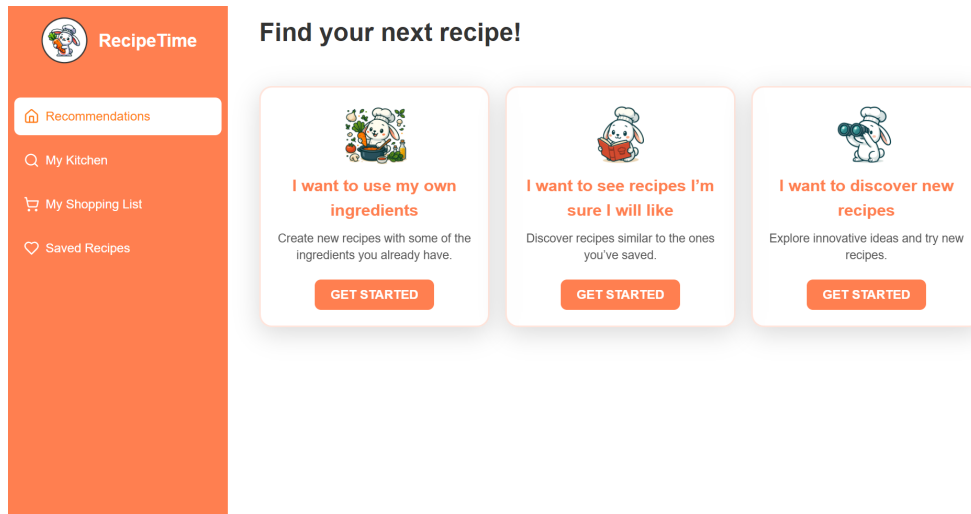


Figure B.5: Recommendations page.

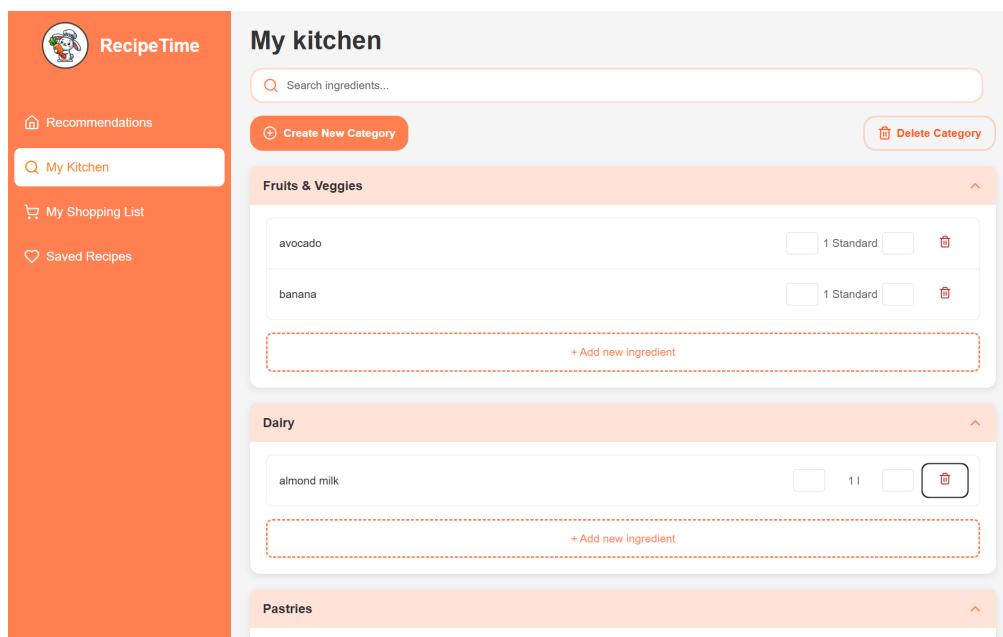


Figure B.6: My kitchen page.

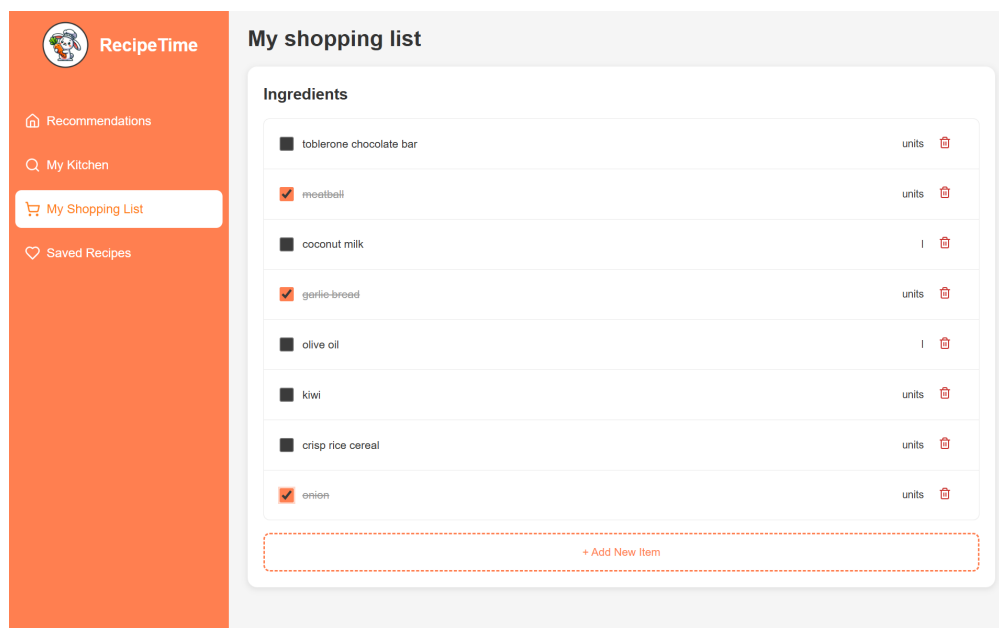


Figure B.7: My shopping list page.

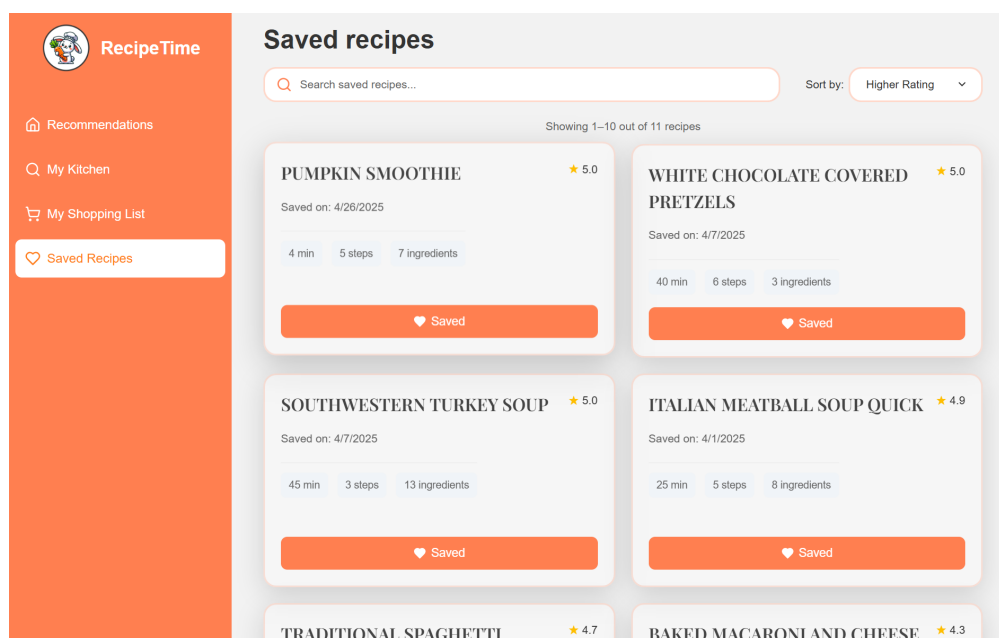


Figure B.8: Saved recipes page.



Universidad Autónoma
de Madrid