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

Author: Alba Delgado Santiago
Advisor: Alejandro Bellogín Kouki

# 1. Introduction & Motivation

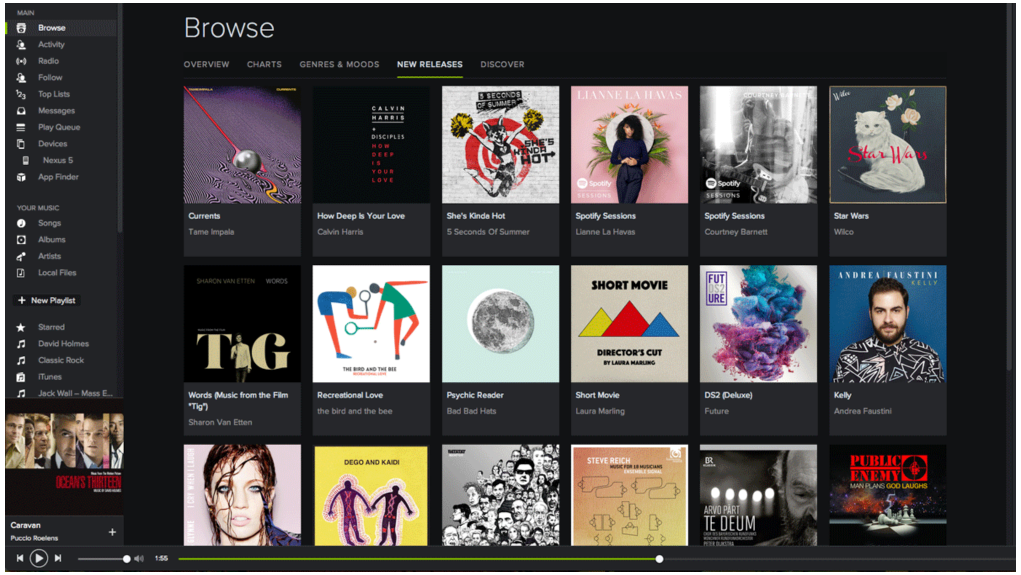
Why RecipeTime?

# 1. Introduction & Motivation

Recommenders Systems


*Netflix Library*

*Spotify Library*



# 2. Project Goals

Our **main objective** is to help users cook more efficiently and sustainably with personalized recipe suggestions

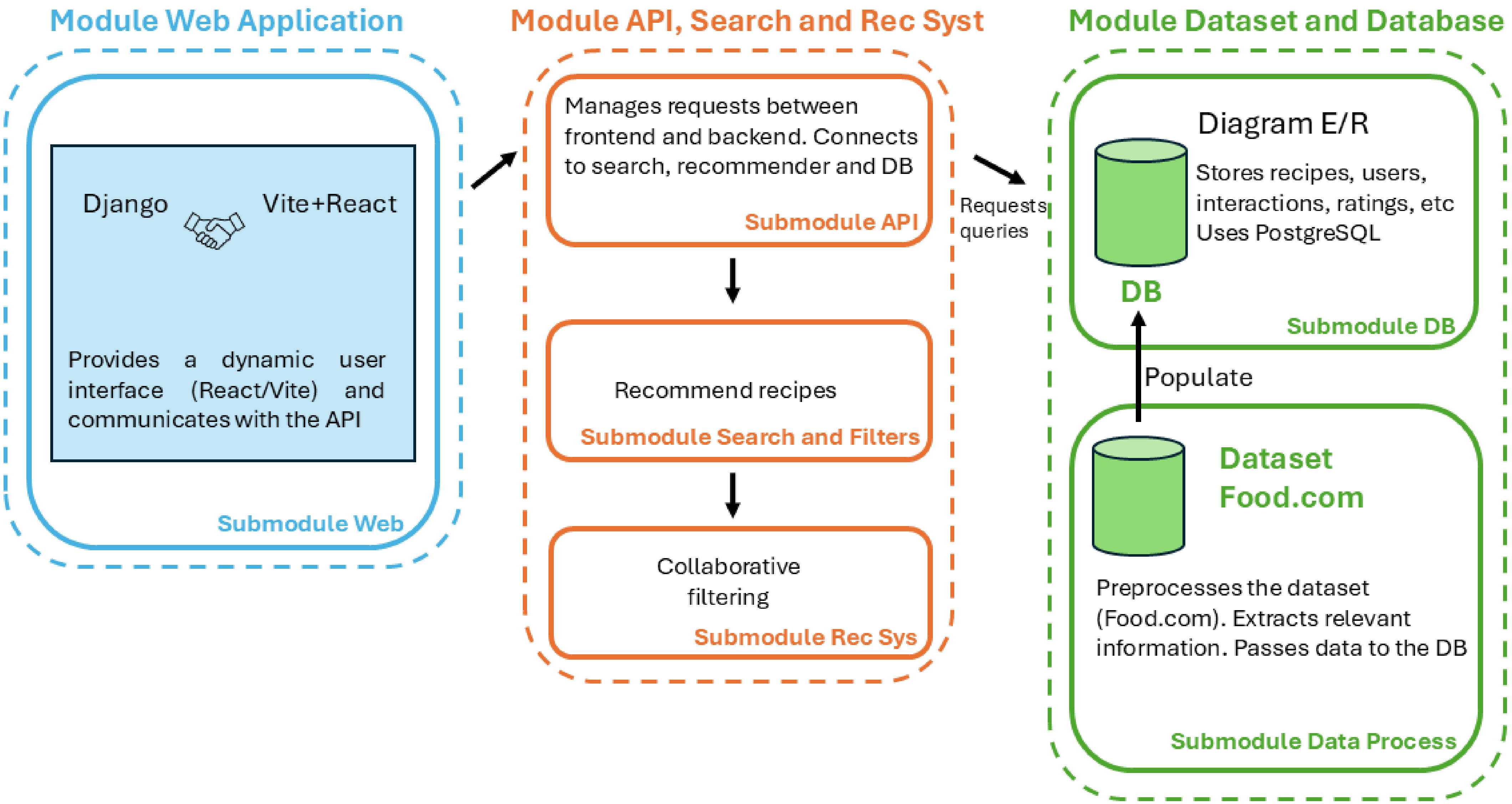
Maximize use of existing kitchen ingredients

Simplify meal planning and encourage dietary variety

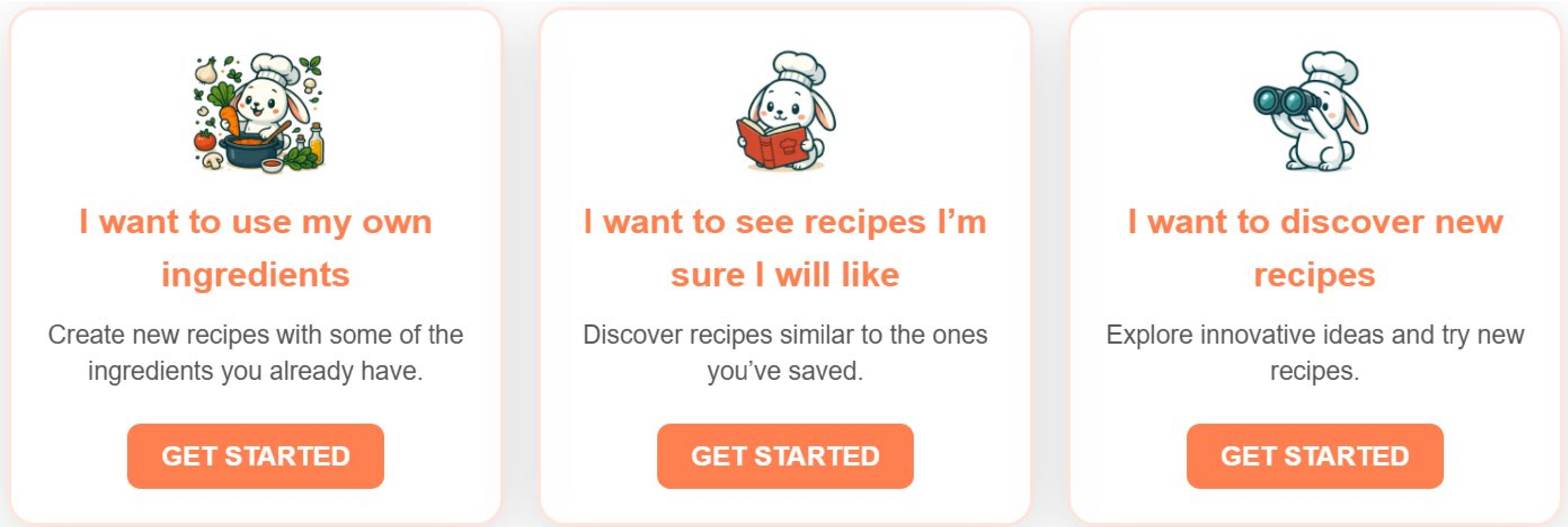
Adapt to individual tastes and dietary needs

**RecipeTime's Approach:** A full-stack web application integrating three core recommendation strategies.

# 3. Project Structure



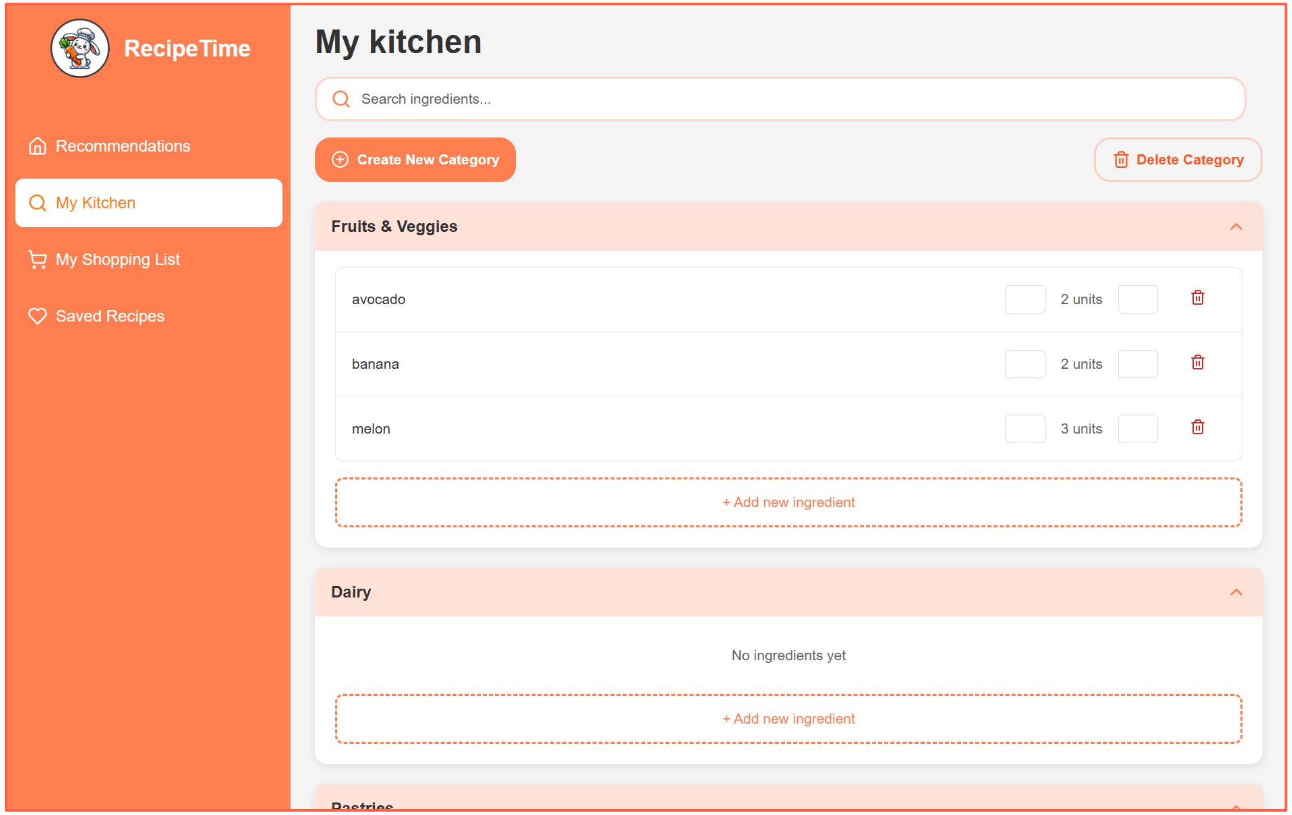
# 4. Core Recommendation Features



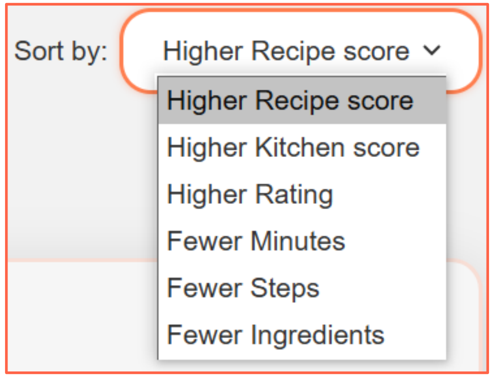

*Recommendations Page in RecipeTime*

# 4.1 Kitchen-based recommendation

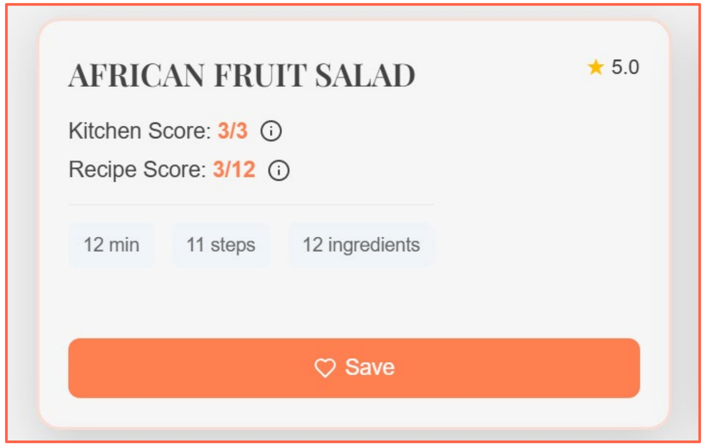
Prioritizes Recipes based on ingredients in the users' "virtual kitchen"




*User's virtual kitchen in RecipeTime*



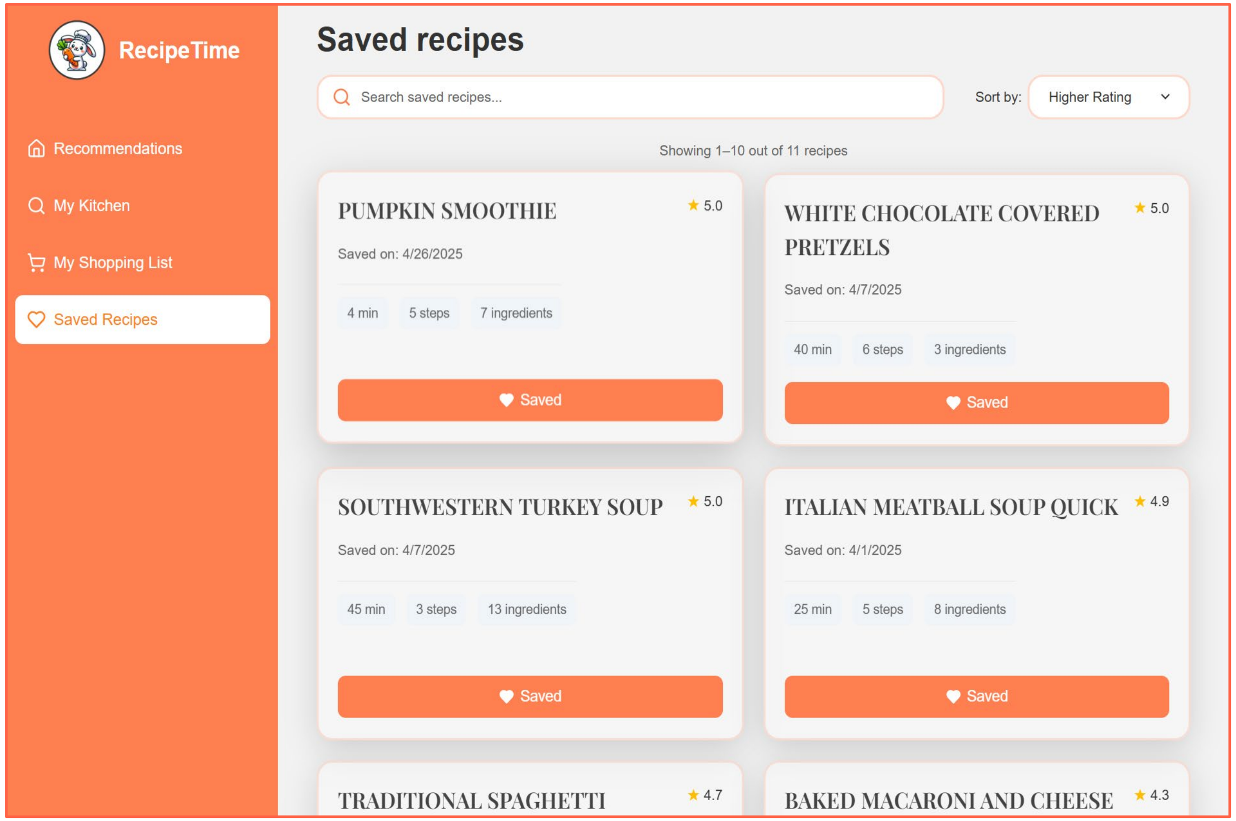

*Sorting Options*




*Recipe Card*

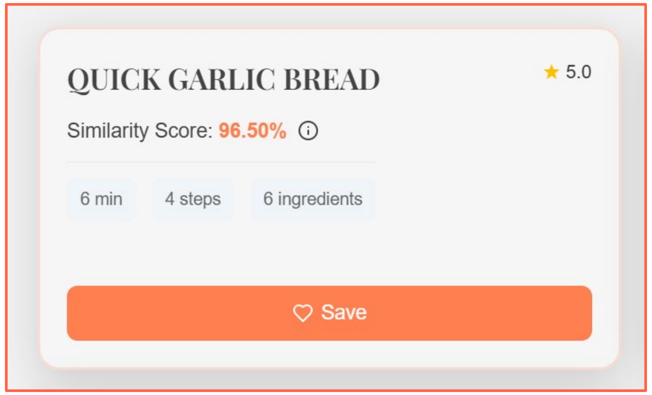
# 4.2 Collaborative filtering

Suggests recipes favored by users with similar tastes, based on *saved* recipes.




*User's saved recipes in RecipeTime*

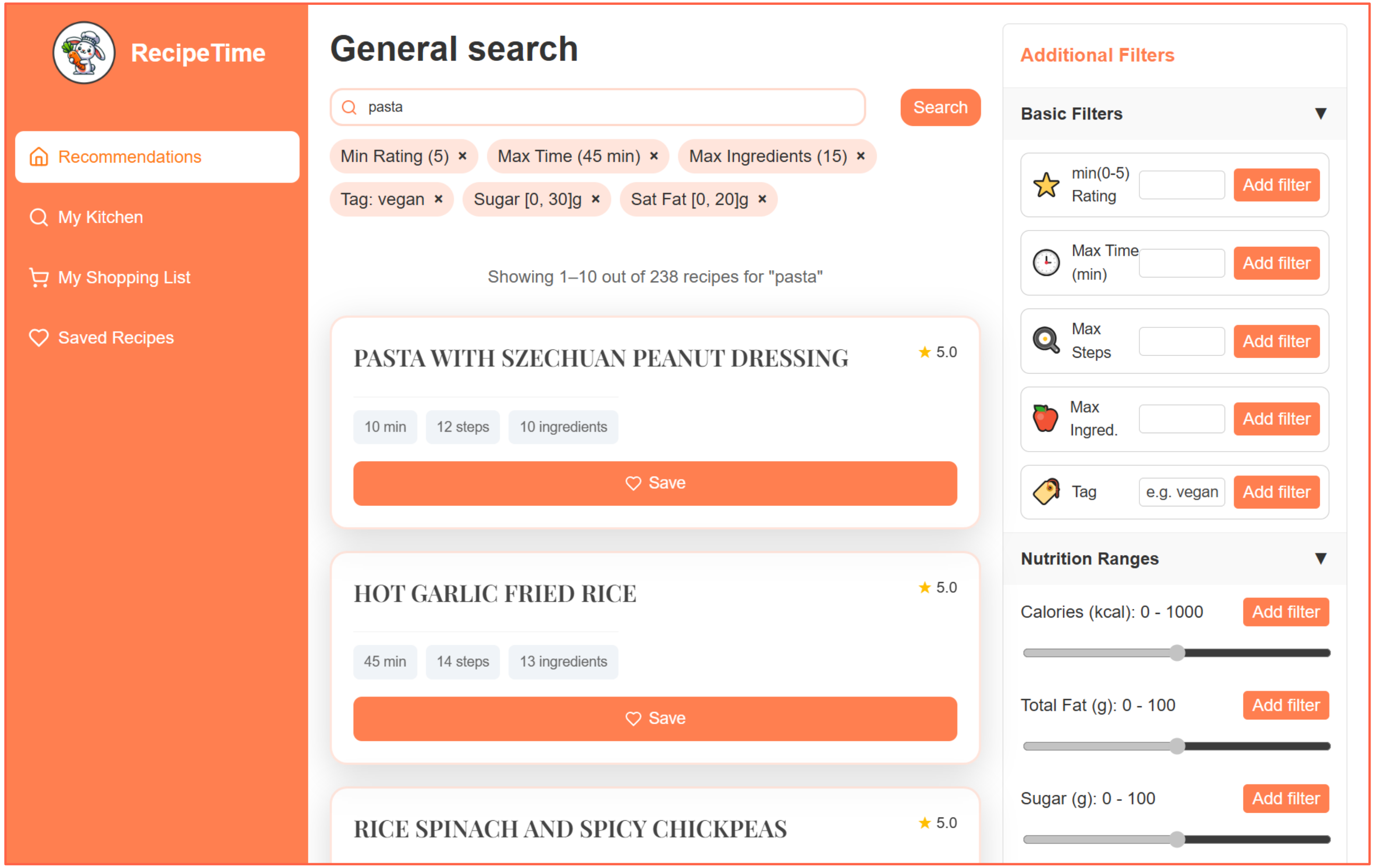

*Sorting Options*




*Recipe Card*

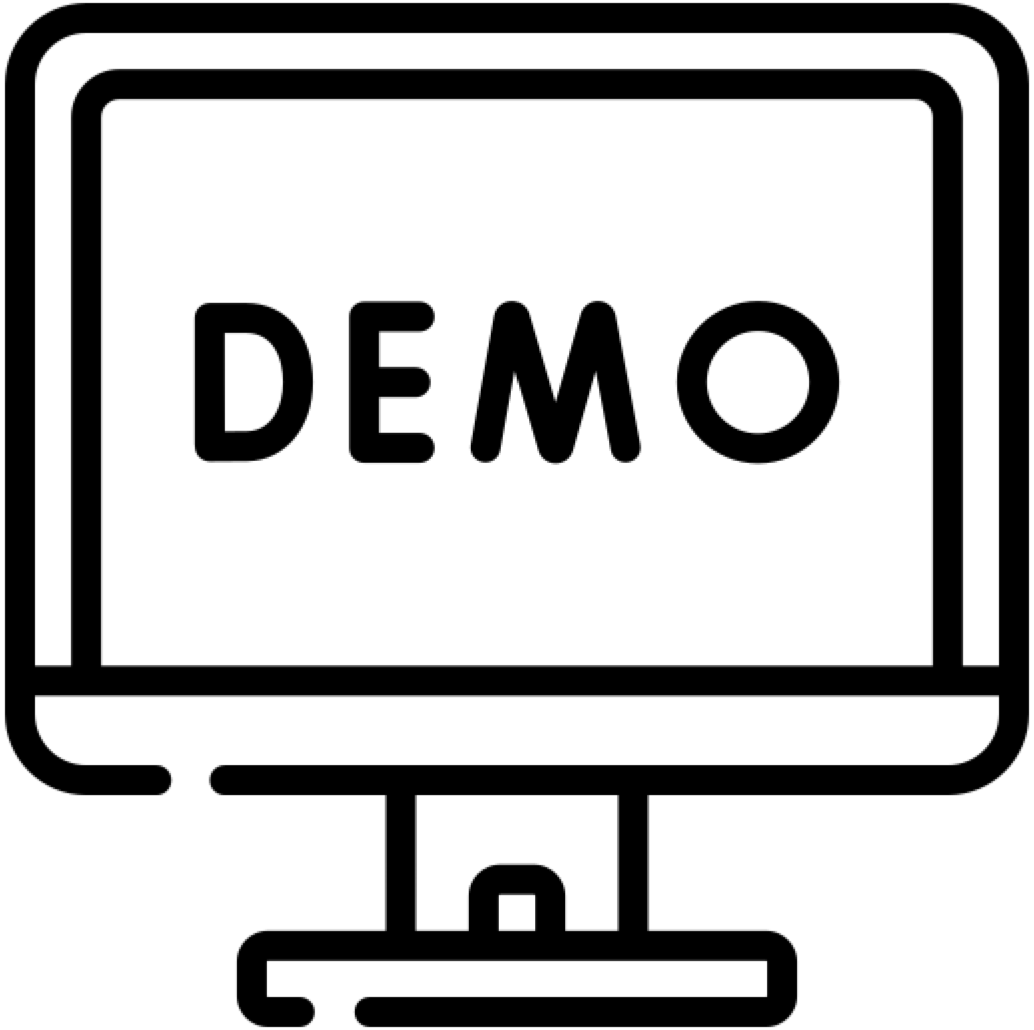
# 4.3 Filtered-driven recommendation

Dynamic search with fine-grained controls.




*General Search in RecipeTime*

# 5. Live Demonstration



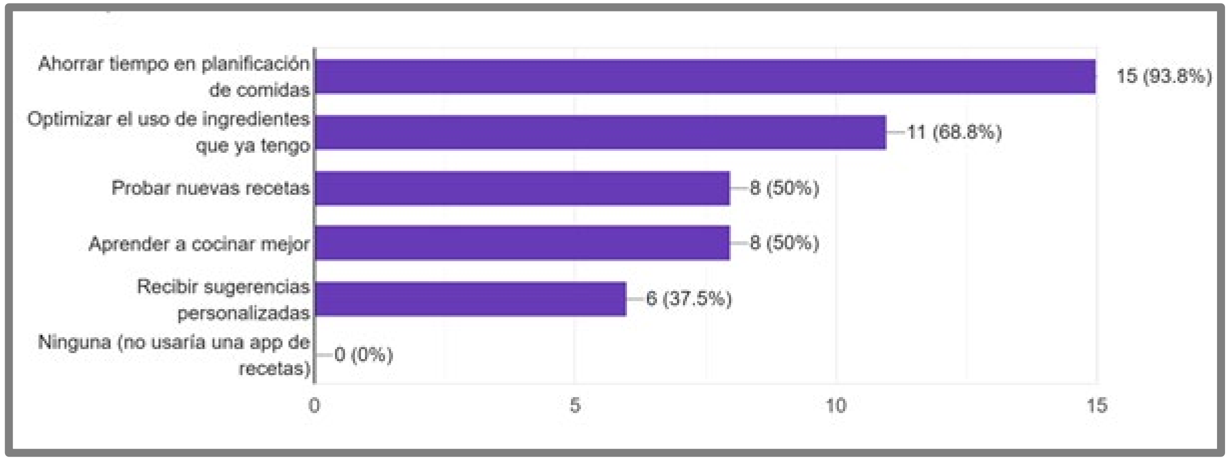
# 6. User study and Results

- 16 participants
- They performed in my laptop guided tasks covering all core functionalities while answering questions in a google form
- All the participants followed this process:

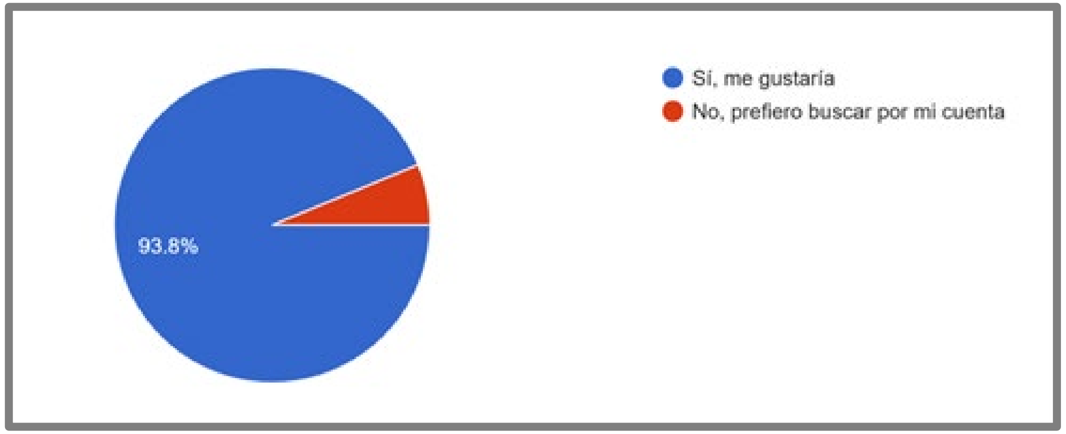
Questions regarding demographics & cooking background, motivations & preferences → Guided tasks covering core features → Post-Test Experience & Feedback → Rate 10 statements to measure usability with the SUS

# 6. User study and Results

### User motivations for using a recipe recommendation app




### Interest in ingredient-based recommendations




**System Usability Scale (SUS) score: 92.5% (“Excellent” Usability)**

# 7. Discussion and Challenges

**Achievements:** Successfully integrated 3 recommendation strategies into a functional full-stack application. Achieved excellent usability. Addressed core motivations.

- ✓ Managing and preprocessing the large Food.com dataset.
- ✓ Implementing and tuning the ALS algorithm for collaborative filtering.
- ✓ Ensuring responsive search and filtering across 180,000+ recipes.
- ✓ Designing an intuitive UI for complex functionalities.
- ✗ Absence of recipe photos (user feedback).
- ✗ Scope limited to recommending (not user-contributed recipes).

# 8. Conclusions

RecipeTime effectively helps users cook more efficiently and sustainably.

It successfully addresses food waste, meal planning complexity, and a personalized dietary needs through its unique combination of recommendation strategies.

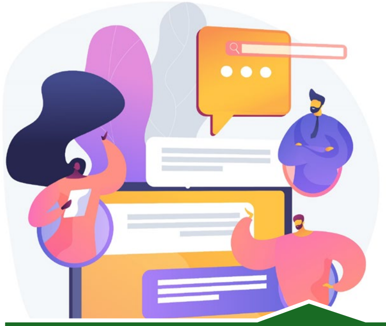
The development consolidated skills in data engineering, machine learning, full-stack development, and user-centered design.

The high SUS score (92.5) validates its intuitive design and effectiveness.

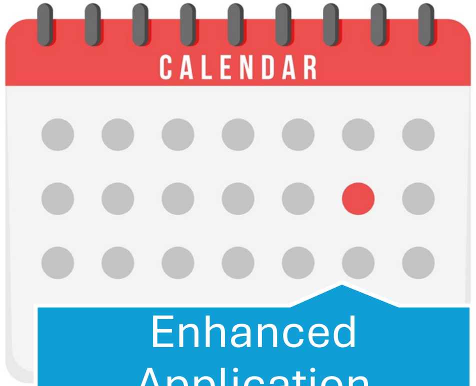
Recipe Time is a tangible step towards more sustainable food practices.

# 9. Future Work

Automated Inventory Tracking



Building a Cooking Community




Enhanced Application Features & Accessibility

Advanced Recommendation Research & Impact Assessment

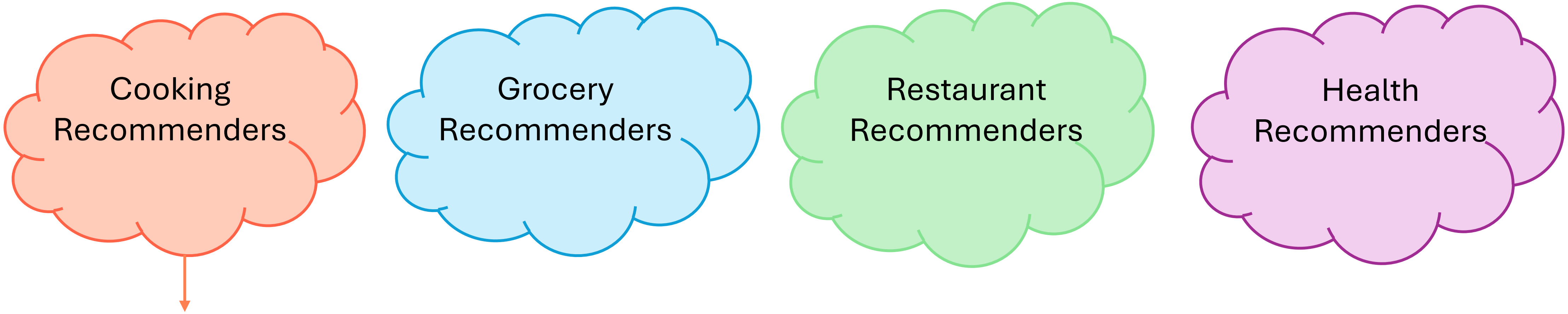
# 10. Acknowledgements

# 11. Questions and Answers

***Questions?***

# State of the Art

- “Food Recommender Systems” chapter in the Recommender Systems Handbook by Elsweiler et al.




RecipeTime’s focus

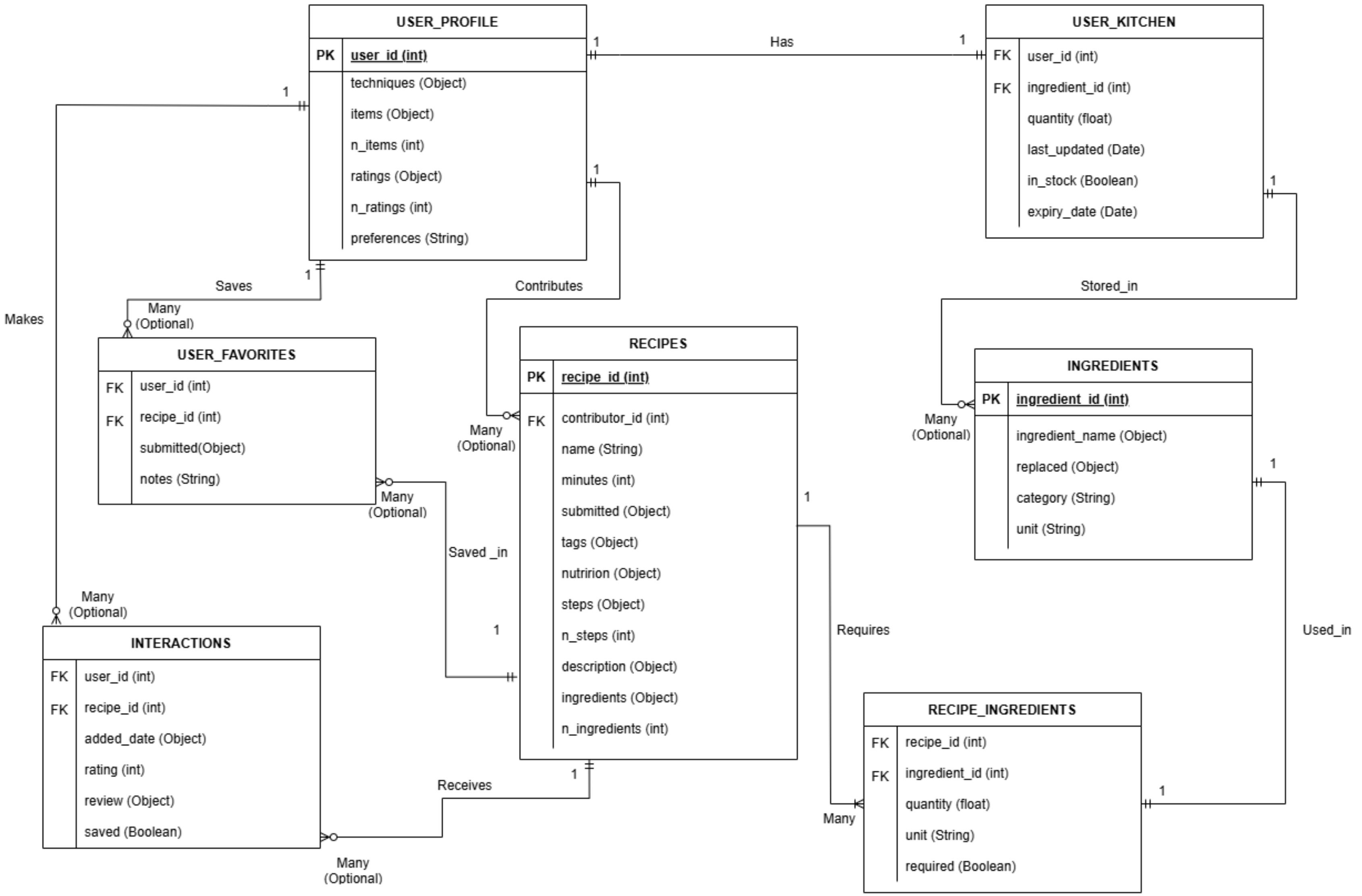
- Key Algorithm for Collaborative Filtering is **Alternating Least Squares (ALS):**

  -Based on Hu et al.’s “Collaborative Filtering for Implicit Feedback Datasets”

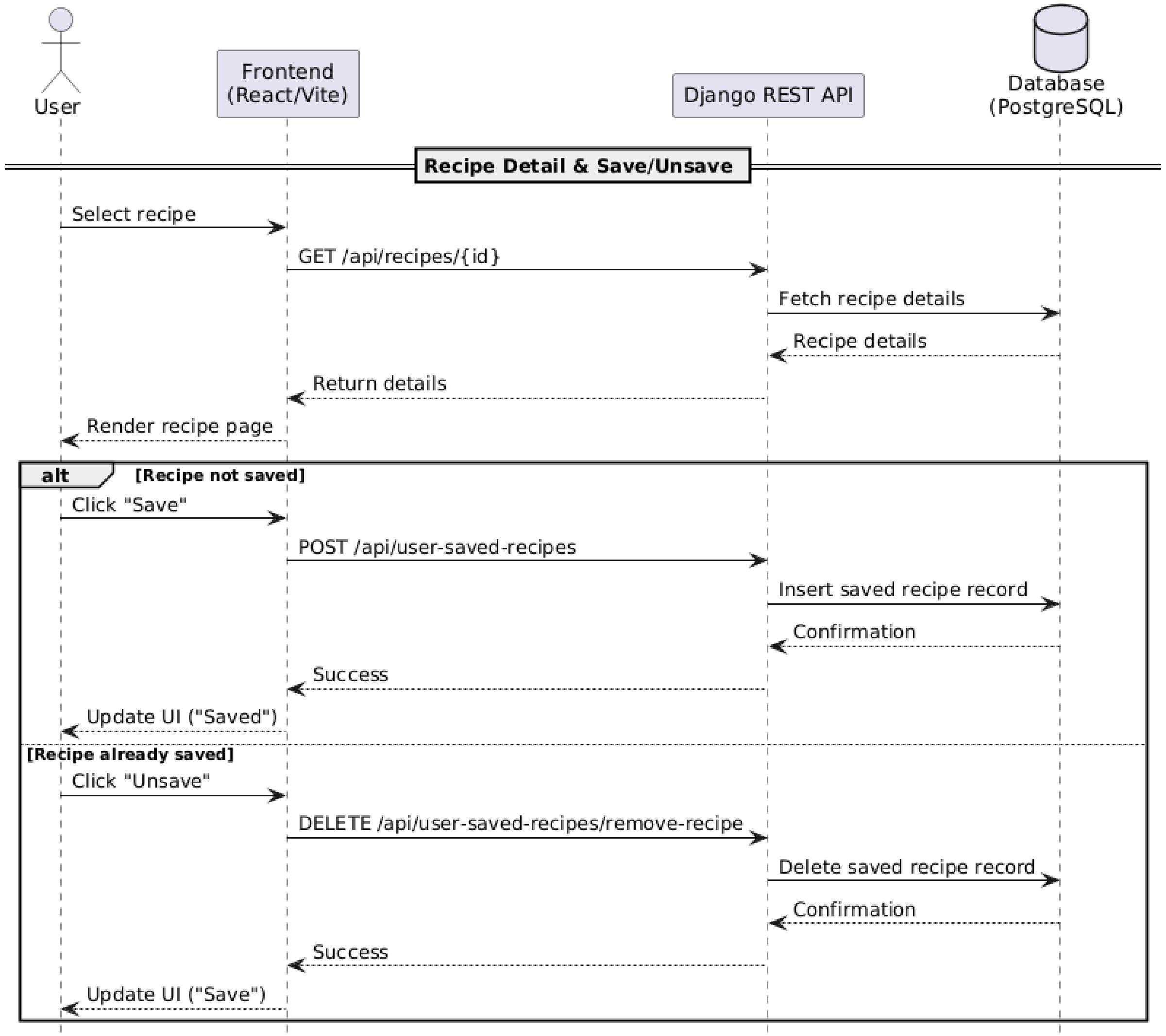
  -Ideal for implicit signals like 'saves' (our primary interaction data).

  -Scalable and efficient for large datasets

# Entity-Relationship Diagram

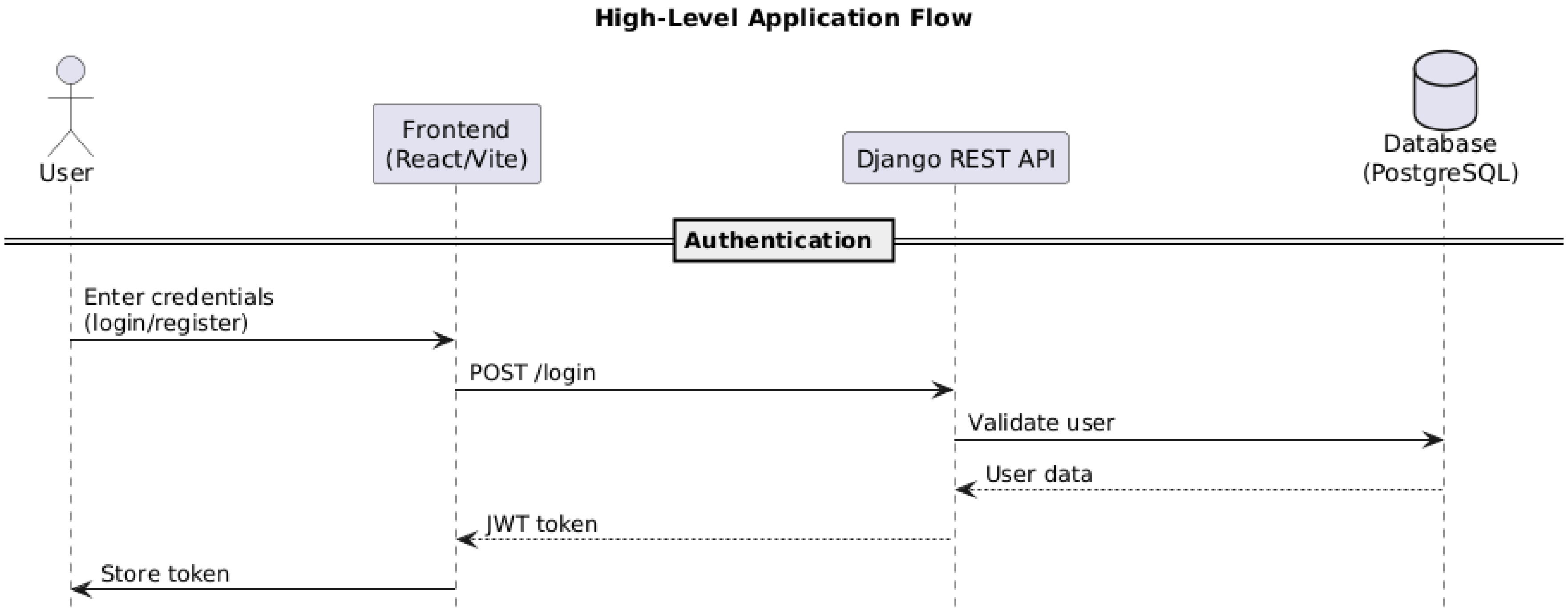


# Sequence Diagram



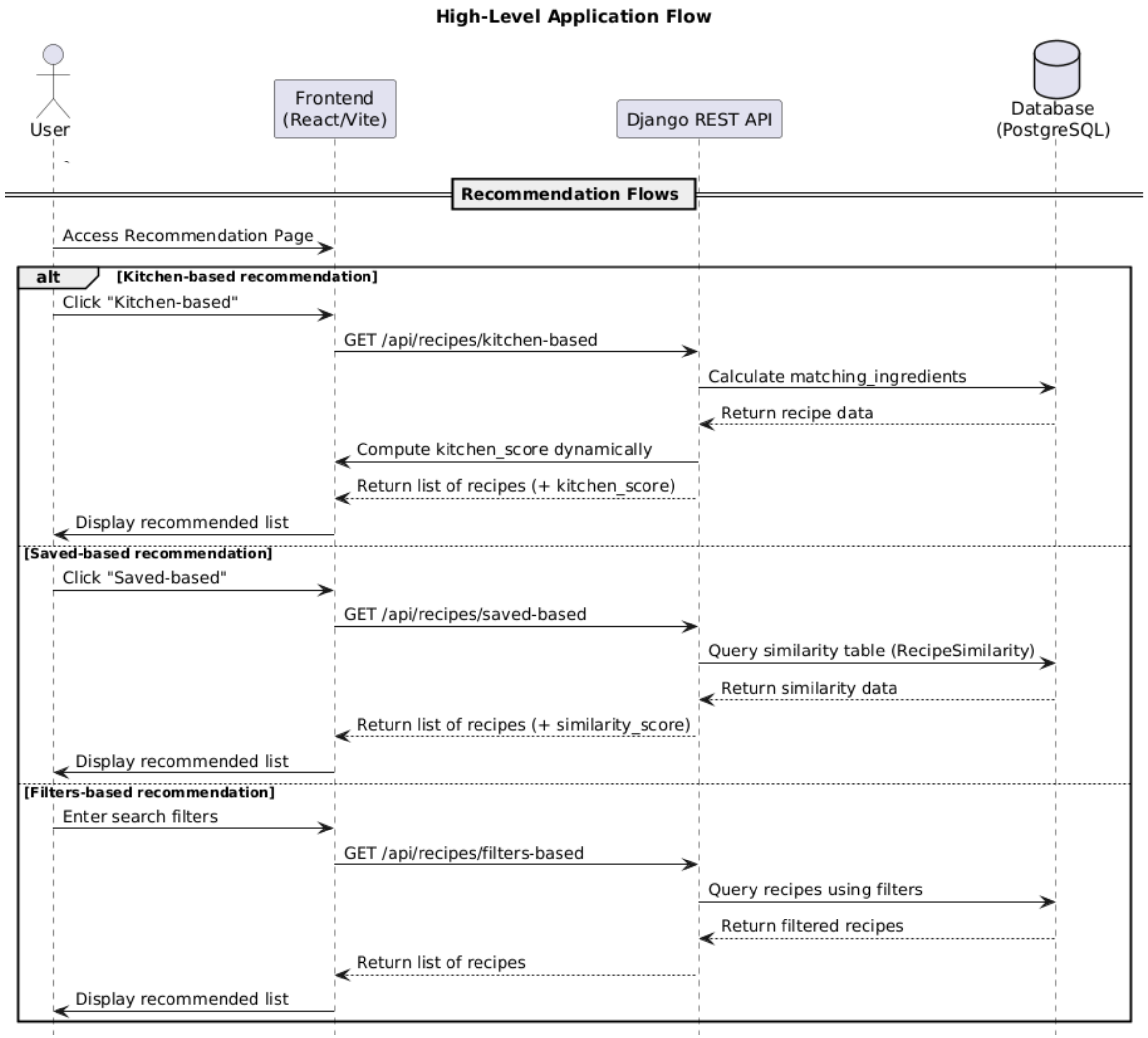

*Recipe Detail & Save/Unsave Recipe*

# Sequence Diagram



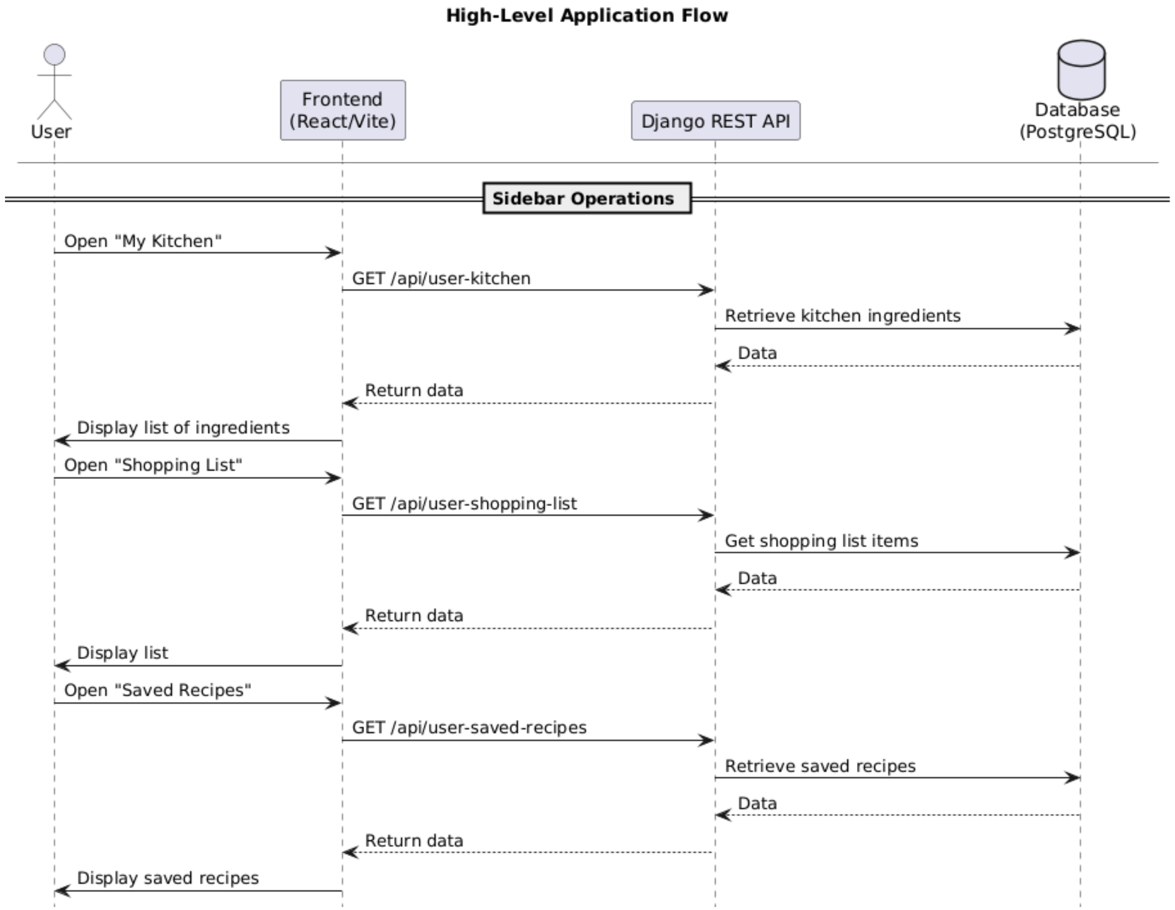

*Authentication: Login or register*

# Sequence Diagram



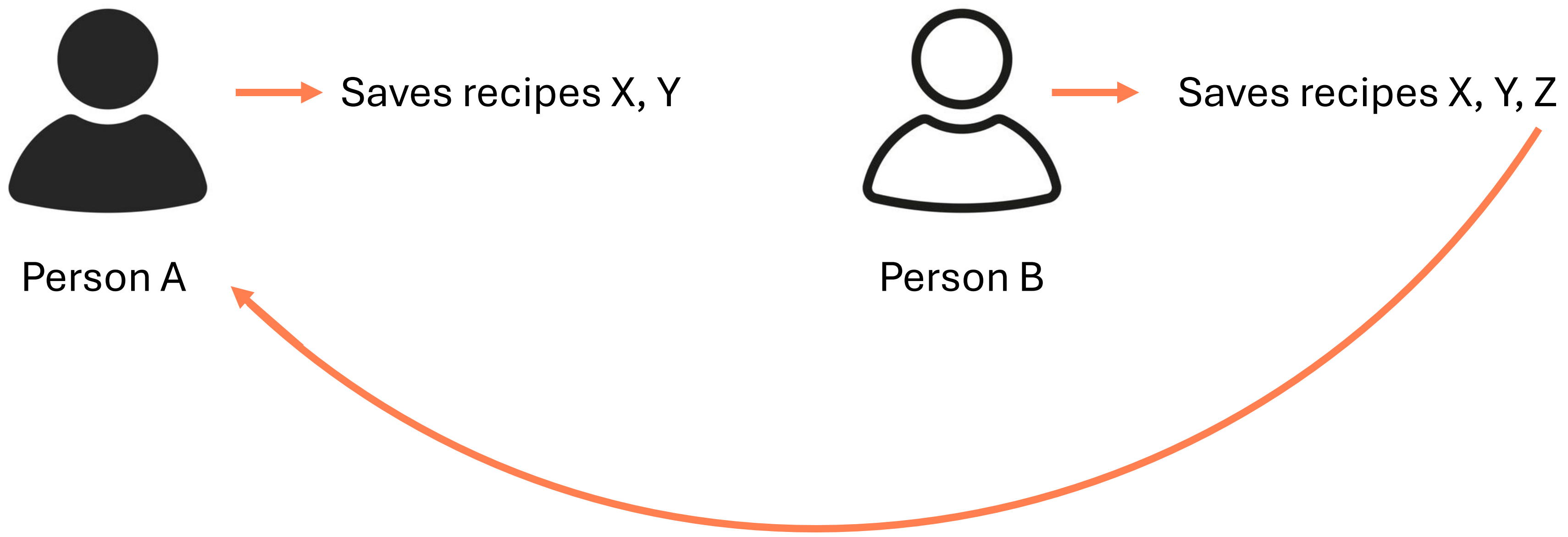

*Recommendation Flows*

# Sequence Diagram




*Sidebar Operations*

# 4.2 Collaborative filtering




RecipeTime recommends Recipe Z to Person A

**How it works:**

1. Learning “Taste Profiles” with ALS (Alternating Least Squares)

2. Finding Similar Recipes Quickly (with Annoy)

# Explanation of scores

Kitchen Score: This score shows **how much of your total food inventory this single recipe will use up**, telling you the impact it will have on your kitchen.

Recipe Score: This score shows you **how ready you are to cook a recipe**, telling you what percentage of its required ingredients you already have.

Similarity Score: This score shows **how well a new recipe matches your personal taste**, based on the recipes you've already saved. A higher score means it's a stronger recommendation for you.

# Final Registers loaded in the DB

| 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 |

*Every table in the Database with its corresponding number of loaded registers.*

# ALS Algorithm

```python
def train_als_model(sparse_ratings, factors=64, iterations=15, regularization=0.1):
    """
    Train an ALS model using the implicit library to obtain low-dimensional embeddings.
    Returns the trained ALS model.
    """
    # The implicit library expects a (users x items) matrix.
    # Our matrix is (recipes x users), so we transpose it.
    model = implicit.als.AlternatingLeastSquares(
        factors=factors,
        regularization=regularization,
        iterations=iterations,
        calculate_training_loss=True
    )
    model.fit(sparse_ratings.transpose())

    return model
```

*Applying the ALS algorithm from the implicit library*

# Annoy Index

```
def build_annoy_index(item_factors, n_trees=10):
    """
    Build an Annoy index from the given item embeddings (item_factors).
    item_factors shape: (num_recipes, embedding_dim)
    Returns the built Annoy index.
    """

    num_items, dim = item_factors.shape
    ann_index = AnnoyIndex(dim, 'angular')

    for i in range(num_items):
        ann_index.add_item(i, item_factors[i])
        if i > 0 and i % 10000 == 0:

    ann_index.build(n_trees)

    return ann_index
```

*Building an Annoy Index*

# Top k nearest neighbors

```
def compute_topk_similarities_annoy(ann_index, recipe_ids, k=200):
    """
    Compute the top-k similar recipes for each recipe using the Annoy index.
    Returns a dict: recipe_id -> list of (similar_recipe_id, similarity_score).
    """

    num_recipes = len(recipe_ids)
    top_k_similarities = {}

    for i in range(num_recipes):
        # get_nns_by_item returns indices of the nearest neighbors
        neighbor_indices = ann_index.get_nns_by_item(i, k + 1)

        neighbor_indices.remove(i)

        similar_list = []

        for neighbor_idx in neighbor_indices:
            dist = ann_index.get_distance(i, neighbor_idx)
            sim_score = 1.0 -dist
            similar_list.append((int(recipe_ids[neighbor_idx]), sim_score))

        top_k_similarities[int(recipe_ids[i])] = similar_list

    return top_k_similarities
```

*Retrieve top k nearest neighbors*

# Summary User-study

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

*Post-test experience summary*