Random Recipe Generator - Design Document

Project Overview:

The Random Recipe Generator is a Python-based application aimed at inspiring users to try new dishes by providing randomly selected recipes. Users can specify preferences such as cuisine, ingredients, or meal type, and the system will generate options that match their criteria. Recipes will include comprehensive details like ingredients, step-by-step instructions, preparation time, and cuisine type. The application is designed to be lightweight, responsive, and user-friendly.

Requirements:

Functional Requirements:

1. Generate Recipes: Users can generate a random recipe at the click of a button or command.
2. Filters: Users can filter recipes by:
   - Cuisine: Select from various cuisines such as Italian, Indian, Mexican, Chinese, etc.
   - Ingredients: Specify ingredients to include or exclude.
   - Meal Type: Choose from meal categories such as breakfast, lunch, dinner, or snacks.
3. Recipe Details: Each recipe will display:
   - List of ingredients
   - Step-by-step preparation instructions
   - Preparation time
   - Cuisine type
4. Favorites List: Users can save their favorite recipes for future reference.
5. View Favorites Users can view and manage their list of saved recipes.
6. Search Favorites: Allow users to search their saved recipes using keywords.

Non-Functional Requirements:

1. Performance: Ensure that recipe generation and filtering are completed within 2 seconds.
2. Data Accuracy: Recipes must be complete and formatted for readability.
3. Scalability: Support future integration with additional recipe APIs or datasets.
4. User Experience: Provide a clear, intuitive interface that is accessible to users with minimal technical expertise.

Research Notes:

1. Data Sources:
   - APIs: Use platforms like Spoonacular, Edamam, or TheMealDB to fetch diverse recipes.
   - Local Dataset: Include a preloaded JSON or CSV dataset as a fallback.
2. Technology Stack:
   - Language: Python
   - Libraries:

- `requests` for API integration
- `json` and `pandas` for data handling
- `tkinter` or `PyQt` for graphical user interface (optional)

3. Design Considerations:
   - Modular code structure for easy maintenance and scalability.
   - Efficient filtering algorithms to handle large datasets.

Design Documents:

Flowchart:

[Start] -> [User Chooses Filters] -> [Fetch Recipes from Source]
-> [Apply Filters] -> [Randomly Select Recipe] -> [Display Recipe]
-> [Save Recipe to Favorites?] -> [Exit or Repeat]

Detailed Workflow:

1. Initialization:
   - Load recipes from a local dataset or fetch them from an API.
   - Parse and store data for quick access.

2. User Interaction:
   - Present options for filtering (cuisine, ingredients, meal type).
   - Accept user inputs and validate them.

3. Recipe Filtering:
   - Apply filters sequentially based on user preferences.
   - Narrow down the recipe pool to relevant options.

4. Random Selection:
   - Use Python's `random.choice` to pick a recipe from the filtered list.

5. Output Display:
   - Format the selected recipe with clear instructions and details.
   - Allow users to save the recipe to their favorites list.

6. Favorites Management:
   - Enable users to view, search, and delete saved recipes.

Pseudocode:

```
import random
import json

# Load recipe dataset or integrate with API
def load_recipes(source):
    if source == "local":
        with open('recipes.json') as f:
```

```
            return json.load(f)
    elif source == "api":
        # Make API calls to fetch recipes
        pass

# Filter recipes based on user inputs
def apply_filters(recipes, cuisine=None, ingredients=None, meal_type=None):
    filtered = recipes
    if cuisine:
        filtered = [r for r in filtered if r['cuisine'].lower() == cuisine.lower()]
    if ingredients:
        filtered = [r for r in filtered if all(ing.lower() in r['ingredients'] for ing in ingredients)]
    if meal_type:
        filtered = [r for r in filtered if r['meal_type'].lower() == meal_type.lower()]
    return filtered

# Select a random recipe
def get_random_recipe(filtered_recipes):
    return random.choice(filtered_recipes)

# Save recipe to favorites
def save_to_favorites(recipe, favorites):
    favorites.append(recipe)
    with open('favorites.json', 'w') as f:
        json.dump(favorites, f)

# Main program function
def main():
    recipes = load_recipes("local")
    try:
        with open('favorites.json') as f:
            favorites = json.load(f)
    except FileNotFoundError:
        favorites = []

    while True:
        # User input for filters
        cuisine = input("Enter cuisine (or leave blank): ")
        ingredients = input("Enter ingredients (comma-separated or leave blank): ").split(',')
        meal_type = input("Enter meal type (or leave blank): ")

        # Apply filters and select a recipe
        filtered = apply_filters(recipes, cuisine, ingredients, meal_type)
        if not filtered:
            print("No recipes found. Please adjust your filters.")
            continue
```

```
    random_recipe = get_random_recipe(filtered)
    print("Recipe:", random_recipe)

    # Save to favorites
    save = input("Save to favorites? (y/n): ")
    if save.lower() == 'y':
        save_to_favorites(random_recipe, favorites)

    # Exit or continue
    cont = input("Generate another recipe? (y/n): ")
    if cont.lower() != 'y':
        break
```

User-Facing Design:

1. Console Interface:
   - Menu Options:
     - Generate a Recipe
     - View Favorites
     - Exit
   - Input prompts for filtering options (cuisine, ingredients, meal type).

2. Sample Output:
Recipe: Spaghetti Carbonara
Cuisine: Italian
Ingredients: spaghetti, eggs, pancetta, Parmesan
Instructions: Cook spaghetti. In a separate pan, fry pancetta. Mix eggs and Parmesan. Combine all ingredients.
Preparation Time: 20 minutes

3. Favorites Display:
Saved Recipes:
1. Spaghetti Carbonara
2. Vegetable Stir Fry
3. Chicken Tikka Masala

Implementation Details:

-Modules:
1. data_loader.py: Handles loading and fetching of recipes from local or API sources.
2. filters.py: Contains filtering logic.
3. favorites.py: Manages the favorites list (add, view, delete).
4. main.py: Integrates all modules and serves as the entry point for the application.

- Error Handling:
  - Handle invalid user inputs gracefully.
  - Notify users when no recipes match their filters.

- File Structure:

```
project_directory/
  |-- data/
      |-- recipes.json
      |-- favorites.json
  |-- src/
      |-- data_loader.py
      |-- filters.py
      |-- favorites.py
      |-- main.py
  |-- tests/
      |-- test_filters.py
      |-- test_data_loader.py
```

Testing:

1. Unit Testing:
   - Test filter logic for accuracy.
   - Validate random recipe selection.
2. Integration Testing:
   - Test end-to-end functionality, including filtering and favorites management.
3. Performance Testing:
   - Ensure recipes are displayed within the required time frame.

UI Design:

The user interface concept for the **Random Recipe Generator** is designed to be intuitive and visually engaging, focusing on simplicity and functionality. Here's a breakdown of the design elements:

1. **Navigation Bar:**
   - Located at the top, this bar includes menu options such as:
     - **Home:** Takes users back to the main page.
     - **Favorites:** Allows users to view saved recipes.
     - **Settings:** Provides options to customize preferences or manage the app.
2. **Filter Section:**
   - Positioned below the navigation bar, users can apply filters for:
     - **Cuisine Type:** A dropdown or multi-select list for choosing cuisines like Italian, Mexican, or Indian.
     - **Ingredients:** A text input or checkboxes to include or exclude specific ingredients.

        - **Meal Type:** Radio buttons or dropdowns for breakfast, lunch, dinner, or snacks.
3. **Generate Recipe Button:**
    - A prominent button in the center encourages interaction. Users click this to randomly generate a recipe based on selected filters.
4. **Recipe Display Area:**
    - Located below the button, this area showcases:
        - **Recipe Name and Image:** Visually appealing title and image to catch the user's attention.
        - **Ingredients List:** Clearly formatted with bullet points.
        - **Instructions:** Step-by-step preparation details.
        - **Additional Info:** Preparation time, cuisine type, and dietary information.
5. **Color Scheme and Style:**
    - A vibrant mix of **green**, **orange**, and **white** conveys freshness and creativity.
    - Rounded buttons, clear typography, and responsive layouts enhance usability.

Future Enhancements:

1. Add filters for dietary preferences such as vegan, keto, or gluten-free.
2. Expand the database with more recipes and cuisines.
3. Develop a full-fledged GUI for enhanced user experience.
4. Implement a recommendation engine based on user preferences.
5. Add multilingual support for recipe instructions and UI elements.