Supplemental Implementation Document

This document provides an overview of the Recipe Generator system implementation, detailing architecture, data models, API endpoints, prompt strategy, error handling, and integration with the React frontend.

---

1. Architecture Overview

- **Frontend**: React app with `react-router-dom` for three slides (input, saved list, landing). Communicates with backend via HTTP (`axios`).
- **Backend**: FastAPI service exposing four endpoints. Uses Uvicorn for ASGI hosting.
- **AI Integration**: OpenAI API (GPT-4o-mini) for recipe generation, returning JSON.
- **Database**: MongoDB (local or Atlas) storing recipes in a `recipes` collection.
- **CORS**: Configured to allow requests from `http://localhost:3000`.

---

2. Data Models (Pydantic)

```
class Ingredient(BaseModel):
    name: str
    quantity: Optional[str] = None

class RecipeIn(BaseModel):
    ingredients: List[Ingredient]
    dishName: Optional[str] = None

class RecipeOut(BaseModel):
    id: str
    dishName: str
    ingredients: List[Ingredient]
    instructions: List[str]

class SaveRecipeIn(BaseModel):
    title: str
    ingredients: List[Ingredient]
```

instructions: List[str]

- `RecipeIn` used for `/generate` payload.
- `RecipeOut` used as response model for both generate and save.
- `SaveRecipeIn` used for explicit saving.

---

3. API Endpoints

Method	Path	Description
OPTIONS	`/generate`	Preflight for CORS.
POST	`/generate`	Accepts `RecipeIn`, calls GPT-4o-mini, returns `RecipeOut`.
POST	`/recipes`	Accepts `SaveRecipeIn`, checks duplicates, saves recipe.
GET	`/recipes`	Returns list of all saved `RecipeOut`.
DELETE	`/recipes/{id}`	Deletes a recipe by its MongoDB `_id`.

---

4. Prompt Strategy

- Build a single JSON prompt to GPT: list ingredients and dish name.
- Request output strictly as JSON with two arrays: `ingredients` (string list) and `instructions` (string list).
- On return, parse JSON, split each ingredient string on `" of "` to reconstruct `{ name, quantity }`.

---

5. Error Handling

- **Missing API Key**: Raises `RuntimeError` at startup.
- **Input Validation**: Raises `HTTPException(400)` if no ingredients provided.
- **JSON Parsing**: Catches `JSONDecodeError`/`KeyError`, returns `HTTPException(500)` for bad AI output.
- **Duplicate Save**: Raises `HTTPException(409)` if same recipe already exists.
- **Delete Not Found**: Raises `HTTPException(404)` if `_id` not in DB.
- **Quota Fallback** (optional): Could catch `insufficient_quota` and retry with `gpt-3.5-turbo`.

---

6. Frontend Integration

- **Slide1.jsx**: Maintains `ingredients` list, calls `POST /generate`, displays resulting `RecipeOut`.
- **Save Button**: Calls `POST /recipes` to persist a recipe.
- **Slide2.jsx**: Calls `GET /recipes`, maps over returned array, rendering each recipe's title, ingredients, and instructions. Includes Delete action calling `DELETE /recipes/{id}`.
- **Slide3.jsx**: Static landing page with navigation.

---

7. Running the Application

Backend

```
python3 -m venv .venv
source .venv/bin/activate
```

```
pip install fastapi uvicorn python-dotenv pymongo openai
uvicorn main:app --reload
```

1.

Frontend

```
npm install
npm start
```

2.
