**🧠 Recipe GPT Chatbot Project Summary**

**🗂️ Overview**

This document captures the essential context, technical architecture, design decisions, known issues, and todos from the development of a custom GPT-powered recipe chatbot.

The goal is to create a web-based assistant that:

* Accepts user input on recipe preferences (e.g. dish type, flavor, cuisine)
* Queries live online Whole30-compliant recipes
* Filters recipes using a pantry/commissary ingredient list (Google Sheet)
* Only shows recipes where 90%+ ingredients are available
* Returns selected recipe name + link

**🛠️ Tech Stack**

* **Frontend**: Gradio (Python)
* **LLM options**: OpenRouter.ai, Together.ai, Mistral, etc.
* **Backend**: Flask (or FastAPI optional)
* **Data sources**: Google Sheets (CSV published), Spoonacular API
* **Hosting options**: NAS/local dev, Hugging Face Spaces, Render.com

**✅ Key Features & Flow**

1. Gradio chatbot UI handles user input
2. Sends query to LLM to interpret intent (e.g., "spicy side dish")
3. Backend Flask API fetches 3 Whole30 recipes from Spoonacular
4. Each recipe’s ingredients are fuzzy matched against pantry/commissary list
5. Recipes with ≥90% pantry match are returned

**🔧 Backend Details**

**File: app.py**

* Flask server with /get\_recipes endpoint
* Reads pantry/commissary list from Google Sheet (CSV)
* Uses rapidfuzz.partial\_ratio to match ingredients
* Fetches recipes from Spoonacular using random offset + shuffling

**Key functions:**

* fuzzy\_match(ingredient, threshold) — Fuzzy string matching
* get\_matching\_recipes(dish\_type, threshold) — Loops through batches of recipes until 3 meet criteria
* @app.route('/get\_recipes') — Accepts dish\_type query param, returns JSON

**File: gradio\_ui.py**

* Gradio chatbot interface
* Function chatbot\_response(user\_message, history):
  + Sends message to LLM (via OpenRouter)
  + Extracts dish type (for now uses raw input)
  + Calls /get\_recipes API endpoint
  + Returns formatted recipe list or fallback

**⚠️ Known Issues**

* Colab environment cannot bind Flask to a public port directly
* flask-ngrok fails in Colab due to missing system daemon
* Port 5000 conflict errors if Flask not shut down properly
* jq missing in Colab when parsing ngrok info

**✅ Decisions Made**

* Use Gradio over Streamlit due to simpler chatbot integration
* Use Flask for backend (easier to set up and sufficient)
* Use OpenRouter for LLM API to avoid OpenAI API cost
* Use Spoonacular as recipe source with live HTTP queries
* Use rapidfuzz to allow fuzzy pantry ingredient matching

**🔜 Next Steps**

**🧩 Backend (Flask)**

Refactor Flask app to:

- Accept `cuisine`, `flavor` as optional query params

- Return available + missing ingredients per recipe

- Allow configurable match threshold (query param)

- Wrap into Docker container for NAS deployment

**💬 Gradio Frontend**

Update Gradio app to:

- Parse LLM reply for structured intent (dish type, cuisine, flavor)

- Allow user to pick one of 3 suggested recipes

- On selection, fetch and show full recipe + URL

**🔌 Hosting**

Create a deploy script to:

- Launch Flask server on NAS or public host

- Start ngrok tunnel from Colab if needed

- Persist pantry sheet link + API key securely

**🧾 API Examples**

**GET /get\_recipes?dish\_type=side%20dish**

{

"recipes": [

{"title": "Spicy Tuna Cakes", "url": "https://..."},

{"title": "Roasted Cauliflower", "url": "https://..."},

{"title": "Garlic Zoodles", "url": "https://..."}

]

}

**📁 File Index**

| **File** | **Purpose** |
| --- | --- |
| app.py | Flask backend for filtering recipes |
| gradio\_ui.py | Gradio chatbot interface |
| requirements.txt | Python dependencies (flask, gradio, rapidfuzz, etc.) |
| pantry.csv | Published Google Sheet of ingredients (read via CSV link) |

**Reboot instructions:** Copy this doc to a new GPT, run Dockerized backend or use Colab/ngrok, then relaunch Gradio.