Build a recipe management app to be used primarily on mobile devices.  
  
The App is called “Recipe Chef”

The format of all recipes will use the common industry standard: Recipe Schema.org JSON-LD. This will enable easy sharing and saving into the Recipe Chef Application from other website.  
The system will require Authentication using Supabase authentication.

The app will have a one time price of $9.99 and an optional monthly fee of $0.99 for the Recipe Finder AI feature. Both prices are variables that can be set on the Admin Page.

## User Interface

User will first arrive at the login screen. This will include a signin, signup and forgot password feature. A banner will appear at the top offering a 14 day free trial that, if clicked, will take the user to the Signup/payment flow.

The sign-up page will take users to a Paddle integrated page allowing them to pay the one time purchase fee and choose whether to include the RecipeSearchAI feature or not.

On the top left will be the Avatar symbol (an O in a circle to match Giuseppe and Marley).

### Chef OuiOui

Once signed in the users will see Chef OuiOui, a French chef avatar who is boisterous, playful and like to tell jokes. He will greet everyone at login with a rolling greeting that is either:

* Mix greetings and jokes (e.g., 1 greeting + 1 joke combo)
* Add an occasional “chef tip of the day” or “OuiOui wisdom” line later for depth  
  use the table schema provided as Appendix A: Random greeting schema and logic

To the right of Chef OuiOui will be a search box to allow users to search for something in their cookbook or, if they subscribed to the monthly   
RecipeFInder AI service, they can also ask Chef OuiOui a question that can fallback to OpenAI if the database can not provide a 70% or better confidence answer.

Chef OuiOui will blink and sway.  
  
My Cookbook  
Under Chef OuiOui and the Search will be “My Cookbook” at the top then a list of the users saved recipes in a table style including Recipe Photo, Recipe Name, Cuisine, Rating, Total time (prep time + cook time). They will also see edit links for every recipe row in the table. A search box will appear on the top of the page allowing the user to search for any recipes with search terms entered returning a list of any recipes that match with fuzzy logic and confidence of 50%. users with the Recipe Search AI feature, will also see an open text box titled “Ask AI to find a recipe” prompting them to ask the system to find them a recipe with anything they list. If they enter a question and click search, the system will first try to find a matching recipe in the global recipe database (75% confidence) and present a list of matches if it finds any. If not it will call OpenAI and ask the user question and return a list of candidates which will be recipes found on the internet. If the user selects a candidate from either the database or AI search results, the recipe will be added to their cookbook.

Users can click on any recipe and be taken to the Recipe Page for that recipe. The recipe page will have the recipe photo on the top left, the recipe name, cuisine, rating and total time on the top right. Underneath that, for users with the Recipe Search AI feature, they will see an open text box prompting them to ask a question about the recipe. If they enter a question and click search, the system will try to answer the question with formatted prose and if it cannot, it will call OpenAI and ask the user question including the recipe name, cuisine and primary protein. For example “<user question> for <cuisine> with <primary protein>. User will also be able to edit or delete the recipe from the Recipe Page. The user question box should prompt for question like “What do I substitute for <ingredient> in this recipe?”, and “Can I use <ingredient> in this recipe”, “How many ounces in a cup”, How many grams in an ounce”.

Underneath that will be Ingredients and Instructions. The user can click a tab to see detailed ingredients will will show them all the checkboxes for protein, vegetable, fruits, sauce, etc… The items in this recipe will be checked.

**Users will see a “Print” button on every recipe card that allows them to** Generate shareable PDFs.

User Profile Pages  
Display personal cookbooks, favorite cuisines, ratings, and badges. Allows the user to opt out of their recipes becoming candidates for the global cookbook. Badgers will be earned by:

1. Adding 25, 50, 100, 250 recipes
2. Adding at least 1 recipe for every cuisine
3. Having 5,10,25,50 or 100 recipes accepted into the Global Cookbook

## Offline Mode

**A local cache will be created on the user device a**llowing the user to view the recipes in their Cookbook without internet connection.

## Roles

There will be 3 roles, Admin, Moderator and user. The User may be Active, Inactive or Trial.   
Include a 14-day free trial that allows full use of the product (without the Recipe Search AI feature).

Global Cookbook  
The system will be pre-loaded with 1000’s of recipes across many cuisines. These will be used to answer user searches for recipes. IN addition, every recipe any user adds will become a recipe candidate unless the User opts out of their recipes being considered for the global Cookbook. A Moderator will decide whether to include it in the Global Cook book or reject it. The Global Recipe table will hold all the data on the recipes in the Global Cookbook.

## Recipe Chef Recipe Finder

The unique feature of this app will be that user can click “Find a recipe” and be brought to the Recipe Discovery section. They can choose any of the following search criteria: Cuisine drop down select 1 (list of all major cuisines which can be edited on the Admin page by Admin users), Meal Type drop down select 1 (Appetizer, Entrée, Side Dish, Dessert, Soup, Cocktails), Protein (checkboxes, multiple ok), Veggies (checkboxes, multiple ok), Fruits (checkboxes, multiple ok), Grains (checkboxes, multiple ok), Sauces (checkboxes, multiple ok), etc. Once the user makes all their selections, the system will look up existing recipes in the system and present a list of options. To lookup, the system will find the recipes in the systems Global Cookbook that have the most matches on the user selections and the ingredients, cuisine and meal type of the recipes stored in the database.  
Matching criteria  
1. Cuisine – must match  
2. Meal Type – must match  
3. Protein – must match at least one meat selected by the user  
3. Other ingredients – the more matches between a recipe and the users selections will increase the match confidence.  
Any recipes with 50% confidence or better (can be changed by an Admin on the Admin page) will be presented in a list to the user as a candidate. Each candidate listed will have an ”+ADD” button next to it. If the user clicks the recipe, it will be added to their personal cookbook. For any recipe in the Global Cookbook, whenever a user adds it to their User Cookbook, a count will be kept so the system knows at all times, for every recipe, how many times it has been added.

If no recipe in the database matches at a high enough confidence level a call to OpenAI will be made ONLY IF the user has subscribed to RecipeFinderAI. If the user does not have that functionality then no results will be returned.  
**AI Feedback Loop** – Allow users to mark AI results as “useful/not useful” to refine model prompts.

Recipe Table  
The system will store recipes for user search. Each ingredient in a recipe will belong to a category including: Proteins (Beef, Pork, Chicken, Lamb, Goat, Duck, Tofu, Venison, Shrimp, Clams, Cod, Trout, Salmon, Tuna, Bacon, Pancetta, Guanciale, etc…. all other meat types used in recipes), vegetables (all common vegetables, spices (common spices from all around the world), baking (flour, brown sugar, baking soda, baking powder, corn starch, corn meal, chocolate chips, etc..), sauces (Salsa, Yum Yum, Oyster Sauce, Hoisin, etc), fruits (all known fruits), Grains (Pasta, Basmati Rice, Arborio Rice, Sushi Rice, White Rice, Carneroli Rice, all other rices, Cous Cous, Israeli Cous Cous, Polenta, Grits, etc ).   
  
Every recipe will have the following Fields:

1. Recipe Image (upload of a pic of the recipe completed)
2. Recipe Video link (optional)
3. Recipe Name
4. Recipe rating (1 to 5 chef hats) users can select the number for their personal rating
5. Favorite – users can mark any recipe as a favorite
6. Servings
7. Difficulty (Easy, Medium, Hard, Very Hard)
8. Prep Time
9. Cooking Time
10. Source (varchar)
11. Source URL (varchar) the URL of the recipe if it was found on the web
12. Cuisine
13. Meal Type
14. Vegetables – check box pick list
15. Proteins - check box pick list
16. Dairy - check box pick list
17. Fruits - check box pick list
18. Grains - check box pick list
19. Baking - check box pick list
20. Sauces - check box pick list
21. Spices – check box pick list
22. Ingredients – a list of all ingredients with the amount of each as a separate field to enable scaling
23. Instructions
24. Equipment – a list of equipment needed for the recipe.
25. Notes
26. Diet type: Vegan, Vegetarian, Keto, Gluten-Free, Dairy-Free, Low-Carb, etc.
27. Tags – user defined tags that can be added to any recipe

Recipes will also have a “Scale” setting that will auto edit the amount of all ingredients based on the factor chosen of: ¼, 1/3, ½, 2/3, 1,2,3,4,5,6,7,8,9,10,15,20. The Factor will be multiplied times the amount of each ingredient needed whenever a scale is chosen and the ingredient list will update with the scaled amounts.

## Search

All searches can be done by typing in text or with voice.

Moderator Page

The moderator page will display all Global Cookbook candidates and allow for anyone with an Admin or Moderator role to ADD the recipe into the Global cook book or reject it.

## Timers

The system will allow the user to set up any number of timers that will sound an alarm or the users choice (use phone sound options).

## Ratings System

All recipes will have a 1 to 5 rating system using chef hat icons. If the user clicks the third chef hat from the right, the system will store a rating of 3 and fill in the 1st, 2nd and 3rd chef hat on the screen.

Admin Page  
Only Admin users will have access to the Admin page. It will be available via a link in the top header and footer on the Home page only seen by Admins. It will allow management of: Users, roles, cuisines, meats, vegetables, fruits, sauces and all other ingredients list items. It will also allow the Admin to set the monthly and one time prices for the application. Search confidence level can also be adjusted on the Admin page. Manage Equipment – add equipment items to the list. The number of days of the free trial will be set on the Admin page defaulting to 14. **Analytics Dashboard** – Show recipe popularity, AI feature usage, user retention, etc.

Menu calendar and   
Provide a menu view and allow users to select a day then add a menu (using a search of their Cookbook) to any day. At any time the user can then click “generate shopping list”. When they do, the system will ask “For how many days from now?”. Once they answer, the system will generate a shopping list from all recipes on their calendar included in those number of days they selected. The shopping list will be separated by the same food categories (Vegetables, Fruits, Protein, Grains, Dairy, Other

Spices – check box pick list

Automatically create a grocery list from selected recipes; merge ingredients across recipes and categorize by section (produce, meat, dairy, etc.).

Shopping List Generator

Within the calendar, allow users to select a day then add any number of recipes to any day or days. At any time the user can then click “generate shopping list”. When they do, the system will ask “For the next how many days?” and “For how many people?”. Once they answer, the system will generate a shopping list from all recipes on their calendar included in those number of days they selected. All recipes will be adjusted based on the number of people provided. If a recipe does not have a number of servings, the default is 4. The shopping list will be separated by the ingredient categories. The list will have a print icon on top allowing it to be printed and a “push to Alexa” to connect to an Amazon Alexa list.

Spices – check box pick list

Automatically create a grocery list from selected recipes; merge ingredients across recipes and categorize by section (produce, meat, dairy, etc.).

Cookbook Upload  
CSV file available to populate and the upload to populate or update RecipeChef for the user.

## Recipe upload (shared from other sites)

A user can click share on any website and call Recipe Chef which will open the app then load the recipe (assuming it uses the standard format) and allow the user to save it.

**What you should support (in order)**

1. **Schema.org Recipe (JSON-LD)** – <script type="application/ld+json"> with @type: "Recipe".
2. **Schema.org Recipe (Microdata/RDFa)** – itemscope/itemprop in the HTML.
3. **h-recipe microformat** – older, still around.
4. **Best-effort DOM extraction** – run Readability, then parse headings “Ingredients/Instructions”.

**Minimal fields to map**

* name → title
* image → cover image URL
* recipeIngredient[] → ingredients (one per line)
* recipeInstructions[] or HowToStep[].text → steps (ordered)
* Optional: description, recipeYield/servings, totalTime, prepTime, cookTime, author.name, nutrition

**UX that makes sharing feel “instant”**

* **Share a URL**: a single input (“Paste recipe link”) → backend fetches & parses.
* **Browser Share**: implement **Web Share Target** so mobile “Share → RecipeChef” sends the page URL to your app.
* **Bookmarklet/Extension**: for desktop, send location.href to your import endpoint.
* **Paste Text**: accept Paprika-style text as an alternative import path.

**Ingestion pipeline (recommended)**

1. **Fetch HTML (server)** with a real UA; follow one redirect.
2. **Extract structured data** (prefer JSON-LD, then Microdata/RDFa, then h-recipe).
3. **Normalize** to your RecipeChef schema.
4. **If none found**: run Readability → find “Ingredients”/“Directions” blocks with heuristics.
5. **Convert** to Paprika-style text (optional) and to your internal schema.
6. **Show edit screen** prefilled; user tweaks → save.

**Practical implementation notes**

* **Parsing libs**:
  + JS: @extractus/article-extractor (for Readability), cheerio, html-entities; JSON-LD is just parse+filter @type === 'Recipe'.
  + Python alt: extruct (JSON-LD/Microdata/RDFa), readability-lxml.
* **Multiple JSON-LD blocks**: choose the one with @type “Recipe” (or array containing it).
* **Instructions**: unify recipeInstructions whether it’s an array of strings or HowToStep objects.
* **Times**: ISO 8601 durations (PT35M) → human strings (“35 minutes”).
* **Images**: may be string or array; take first absolute URL.
* **Legal**: store only normalized fields + source URL; link back to original.

**Tiny mapping (JS, concept)**

function toRecipeChef(sd) {

const r = Array.isArray(sd) ? sd.flat() : [sd];

const recipe = r.find(x =>

x && (x['@type'] === 'Recipe' || (Array.isArray(x['@type']) && x['@type'].includes('Recipe')))

);

if (!recipe) return null;

const steps = Array.isArray(recipe.recipeInstructions)

? recipe.recipeInstructions.map(s => typeof s === 'string' ? s : s.text).filter(Boolean)

: (recipe.recipeInstructions?.text ? [recipe.recipeInstructions.text] : []);

return {

title: recipe.name || '',

description: recipe.description || '',

image: Array.isArray(recipe.image) ? recipe.image[0] : recipe.image || '',

servings: recipe.recipeYield || '',

times: {

prep: recipe.prepTime || '',

cook: recipe.cookTime || '',

total: recipe.totalTime || ''

},

ingredients: recipe.recipeIngredient || [],

directions: steps,

source: {

name: recipe.author?.name || '',

url: recipe.mainEntityOfPage || ''

}

};

}

**Endpoint contract (simple)**

* **POST** /api/import-recipe { url: string } → returns normalized recipe JSON and Paprika-text version.
* If no structured data found, return best-effort parse with a flag: {"confidence":"low"} and still open the editor.

**Format:** JSON-LD (Linked Data, using the Schema.org/Recipe vocabulary)

JASON FOR Recipe Files  
{

"@context": "https://schema.org/",

"@type": "Recipe",

"name": "Classic Spaghetti Carbonara",

"author": {

"@type": "Person",

"name": "Giuseppe Rossi"

},

"image": "https://example.com/images/carbonara.jpg",

"description": "A simple Italian pasta dish made with eggs, cheese, pancetta, and pepper.",

"recipeYield": "2 servings",

"prepTime": "PT15M",

"cookTime": "PT20M",

"totalTime": "PT35M",

"recipeCategory": "Dinner",

"recipeCuisine": "Italian",

"keywords": "pasta, carbonara, italian",

"recipeIngredient": [

"200g spaghetti",

"100g pancetta",

"2 large eggs",

"50g pecorino cheese",

"Black pepper"

],

"recipeInstructions": [

{

"@type": "HowToStep",

"text": "Boil spaghetti in salted water until al dente."

},

{

"@type": "HowToStep",

"text": "Cook pancetta until crisp."

},

{

"@type": "HowToStep",

"text": "Whisk eggs and cheese together in a bowl."

},

{

"@type": "HowToStep",

"text": "Toss pasta with pancetta and egg mixture off the heat."

}

],

"nutrition": {

"@type": "NutritionInformation",

"calories": "550 calories"

}

}

**FUTURE Features:**

* 1. **Add cooking methods and allow for users to select Ingredients and the system generates a recipe with cooking method also generated. AI Recipe Generator (Pro Feature)** – Let users input available ingredients and desired cuisine; AI generates a complete recipe.
  2. **AI Cooking Assistant (Voice Mode)** – Hands-free step-by-step mode: read instructions aloud and respond to “Next step,” “Repeat,” “Timer,” etc.
  3. **AI Image Recognition** – Allow users to upload food photos; AI identifies dish and fetches the closest recipe.
  4. **Recipe Challenges or Collections** – Curated themes (e.g., “Summer Grilling,” “Holiday Baking”) users can explore or contribute to.
  5. **In-App Referral Rewards** – Encourage users to invite friends with a discount or free month of Recipe Finder AI.
  6. **Push Notifications** – Notify about new recipes, trial expiry, or AI suggestions.
  7. **In-App Upsells** – Promote Recipe Finder AI with contextual prompts (e.g., “Didn’t find what you wanted? Unlock AI Search.”).
  8. **Gift Purchases** – Allow users to gift the app or subscription to others.

# Appendix B – Random Greeting Schema and logic

**-- Schema: Rotating lines for Chef OuiOui**

**-- =====================================**

**-- 1) (Optional) enum for type**

**DO $$**

**BEGIN**

**IF NOT EXISTS (SELECT 1 FROM pg\_type WHERE typname = 'ouioui\_line\_type') THEN**

**CREATE TYPE public.ouioui\_line\_type AS ENUM ('greeting', 'joke');**

**END IF;**

**END$$;**

**-- 2) Main table**

**CREATE TABLE IF NOT EXISTS public.ouioui\_lines (**

**id bigserial PRIMARY KEY,**

**type public.ouioui\_line\_type NOT NULL,**

**text text NOT NULL,**

**locale text NOT NULL DEFAULT 'en', -- e.g., 'en', 'fr'**

**mood text, -- optional (e.g., 'boisterous','playful')**

**weight int NOT NULL DEFAULT 1 CHECK (weight > 0), -- for weighted random selection**

**is\_active boolean NOT NULL DEFAULT true,**

**created\_at timestamptz NOT NULL DEFAULT now()**

**);**

**-- Helpful indexes**

**CREATE INDEX IF NOT EXISTS idx\_ouioui\_lines\_type\_active ON public.ouioui\_lines(type, is\_active);**

**CREATE INDEX IF NOT EXISTS idx\_ouioui\_lines\_locale ON public.ouioui\_lines(locale);**

**-- 3) RLS (simple): allow all authenticated users to read; only service\_role can write**

**ALTER TABLE public.ouioui\_lines ENABLE ROW LEVEL SECURITY;**

**DO $$**

**BEGIN**

**IF NOT EXISTS (**

**SELECT 1 FROM pg\_policies**

**WHERE schemaname = 'public' AND tablename = 'ouioui\_lines' AND policyname = 'ouioui\_read\_all'**

**) THEN**

**CREATE POLICY ouioui\_read\_all**

**ON public.ouioui\_lines**

**FOR SELECT**

**TO authenticated**

**USING (true);**

**END IF;**

**END$$;**

**-- (For writes, use Supabase service\_role or run seeds via migrations—no public insert/update policy.)**

**-- 4) Weighted random selector**

**-- Usage: SELECT \* FROM public.get\_random\_ouioui\_line('greeting', 'en');**

**CREATE OR REPLACE FUNCTION public.get\_random\_ouioui\_line(p\_type public.ouioui\_line\_type, p\_locale text DEFAULT 'en')**

**RETURNS public.ouioui\_lines**

**LANGUAGE sql**

**STABLE**

**AS $$**

**SELECT l.\***

**FROM public.ouioui\_lines l**

**WHERE l.is\_active = true**

**AND l.type = p\_type**

**AND l.locale = COALESCE(p\_locale, 'en')**

**ORDER BY -LN(RANDOM()) / l.weight**

**LIMIT 1;**

**$$;**

**-- =====================================**

**-- Seed data: English greetings (type=greeting)**

**-- =====================================**

**INSERT INTO public.ouioui\_lines (type, text, locale, mood, weight) VALUES**

**('greeting', 'Bienvenue, mon ami! Chef OuiOui reporting for duty — and possibly dessert.', 'en', 'boisterous', 1),**

**('greeting', 'Ah, bonjour! You smell that? That’s the aroma of opportunity… or maybe garlic. Let’s cook!', 'en', 'playful', 1),**

**('greeting', 'Salut, salut! I hope you brought your appetite — and maybe a fire extinguisher, just in case.', 'en', 'playful', 1),**

**('greeting', 'Bienvenue! I am Chef OuiOui, your culinary sidekick. Together, we make ze magic!', 'en', 'friendly', 1),**

**('greeting', 'Bonjour, mon petit croissant! What delicious trouble shall we cook up today?', 'en', 'boisterous', 1),**

**('greeting', 'Ah, my favorite cook returns! I sharpened the knives and my wit — both very dangerous, non?', 'en', 'witty', 1),**

**('greeting', 'Bonjour! Cooking is like love — messy, exciting, and better with butter.', 'en', 'charming', 1),**

**('greeting', 'Bienvenue! Don’t worry, I only shout in excitement, not in judgment!', 'en', 'reassuring', 1),**

**('greeting', 'Ah, the kitchen! My stage, your adventure. Ready for another chef-d''œuvre?', 'en', 'dramatic', 1),**

**('greeting', 'Bonsoir or bonjour — time has no meaning when there’s food to be made!', 'en', 'playful', 1);**

**-- =====================================**

**-- Seed data: English jokes (type=joke)**

**-- =====================================**

**INSERT INTO public.ouioui\_lines (type, text, locale, mood, weight) VALUES**

**('joke', 'You know why I never trust omelettes? They’re always up to something eggs-tra.', 'en', 'punny', 1),**

**('joke', 'My secret ingredient is… accidental genius! (And maybe too much wine.)', 'en', 'boisterous', 1),**

**('joke', 'Remember: if it sticks to the pan, it means the pan loves you.', 'en', 'witty', 1),**

**('joke', 'Never cry over chopped onions. Cry over overcooked pasta — that’s tragedy.', 'en', 'dramatic', 1),**

**('joke', 'I told my baguette a joke… it was a little stale, but it still cracked up!', 'en', 'punny', 1),**

**('joke', 'I once tried to diet. It lasted until I smelled croissants.', 'en', 'playful', 1),**

**('joke', 'A recipe without butter? Sacrebleu! That’s not cuisine — that’s sadness!', 'en', 'boisterous', 1),**

**('joke', 'Cooking is art, baking is science, and eating… ah, eating is philosophy.', 'en', 'charming', 1),**

**('joke', 'I asked my sauce for advice — it told me to simmer down.', 'en', 'punny', 1),**

**('joke', 'The secret to happiness? One spoon for tasting, one for stirring, and one for wine.', 'en', 'witty', 1);**

**-- =====================================**

**-- Optional convenience views**

**-- =====================================**

**CREATE OR REPLACE VIEW public.ouioui\_greetings AS**

**SELECT \* FROM public.ouioui\_lines WHERE type = 'greeting' AND is\_active = true;**

**CREATE OR REPLACE VIEW public.ouioui\_jokes AS**

**SELECT \* FROM public.ouioui\_lines WHERE type = 'joke' AND is\_active = true;**

 **Random greeting:** SELECT \* FROM public.get\_random\_ouioui\_line('greeting','en');

 **Random joke:** SELECT \* FROM public.get\_random\_ouioui\_line('joke','en');