Smart Recipe Generator - Full Project Plan (Updated)

Project Overview

Project Overview

The Smart Recipe Generator is a full-stack web application that generates and recommends recipes based on ingredients users have. It supports dietary preferences and provides nutritional details. It integrates AI-based ingredient recognition and includes secure authentication, a relational database for recipes and users, and object storage for images.

Technology Stack

Technology Stack

- Frontend: React.js (or simple HTML/CSS/JS prototype)
- Backend: Node.js with Express
- Database: PostgreSQL (hosted on Render)
- Storage: Supabase Storage (for recipe images)
- AI Integration: Clarifai API (ingredient recognition)
- Deployment: Render (Web Service + Postgres)

- Version Control & CI/CD: GitHub + GitHub Actions

Phase 1: Setup and Local Development

Phase 1: Setup and Local Development

1. Initialize GitHub repository with .gitignore, package.json, and migrations/init.sql.
2. Run a local PostgreSQL instance using Docker for development: docker run --name recipe-pg –e POSTGRES_USER=dev -e POSTGRES_PASSWORD=dev -e POSTGRES_DB=recipes -p 5432:5432 -d postgres:15
3.Apply schema locally: export DATABASE_URL=postgres://dev:dev@localhost:5432/recipes psql "$DATABASE_URL" -f migrations/init.sql
4. Develop backend (Express API) with endpoints: - /auth/signup, /auth/login - /api/recognize (Clarifai integration) - /api/recipes (CRUD operations) - /api/recommend (suggest recipes based on input ingredients and preferences)

5. Implement JWT authentication (bcrypt + jsonwebtoken). 6. Store local environment variables in a .env file (do not commit .env).

Phase 2: Database Design

- Tables include users, recipes, ingredients, recipe_ingredients, ingredient_nutrition_100g, user_recipe_actions, dietary_preferences, and user_preferences.
- Images stored in Supabase Storage, only URLs in Postgres (recipes.image_url).
- Indexes on search_vector and foreign keys for performance.

PHASE 3: AI Integration

- Use Clarifai food-item-recognition model for ingredient detection.
- Backend sends image to Clarifai API and parses top results.

- Recognized ingredients matched against Postgres data.

Phase 4: Recipe Generation and Recommendation (Updated)

Phase 4: Recipe Generation and Recommendation (Updated)

This project uses PostgreSQL full-text search (tsvector) with a GIN index for fast text-based retrieval of recipes.

- Maintain a search_vector column on recipes that combines title and ingredients_text. Use a triggeror a GENERATED column to keep it up to date.

- When Clarifai returns a list of detected ingredient names, construct a text query using plainto_tsquery or websearch_to_tsquery.
- Use the search_vector match to find candidate recipes and rank them by relevance using ts_rank or websearch_to_ts_rank.
- Example SQL for search and ranking by text relevance: SELECT id, title, ts_rank(search_vector, websearch_to_tsquery($1)) AS rank FROM recipes WHERE search_vector @@ websearch_to_tsquery($1) ORDER BY rank DESC LIMIT 20;
- Combine the text rank with personalization boosts in the application layer or via weighted SQL

scoring. Example final score formula: final_score = text_rank + 0.2 * cook_count + 0.1 * avg_rating

- Use recipe_nutrition_cache to fetch precomputed nutrition values and return them with reciperesults.

- For strict ingredient overlap filters or advanced filtering, normalized joins onrecipe_ingredients are still available but the primary fast retrieval method is tsvector-based search.

Phase 5: Image Handling with Supabase

Phase 5: Image Handling with Supabase

- Host recipe images in Supabase Storage (or Cloudinary/S3). Mark the bucket public for demo or use signed URLs for private content.

- Store only the image URL in recipes.image_url in Postgres. Example upload flow: upload file to Supabase via SDK or admin UI, get the public URL, save URL in the recipe record.

Phase 6: Deployment Workflow

Phase 6: Deployment Workflow

1. CI/CD Pipeline (GitHub Actions):
 - On push or PR: run tests, build, and optionally run migrations on a test DB.
 - Option: use a manual workflow dispatch to apply migrations to the Render production DB.
2. Render Deployment:
- Create a Render Web Service for the Node backend, connect to the GitHub repo.
- Create Render Postgres for production data and copy its DATABASE_URL to Render environment variables.

3. Environment variables:
- Add CLARIFAI_KEY,JWT_SECRET, SUPABASE_URL, SUPABASE_KEY, and DATABASE_URL to Render environment variables.
4. Trigger deployment after migrations are applied and tests pass.

Phase 7: Testing and Verification

- Unit tests for business logic (matching and scoring).

- Integration tests for API endpoints; mock Clarifai responses with nock or similar.

- CI uses a Postgres service for DB-backed tests and runs migrations against the test DB.

- Optional smoke tests that call the deployed URL after Render deploy.

Database Design Highlights

Database Design Highlights

- Normalized tables: users, dietary_preferences, user_preferences, recipes, recipe_steps,ingredients, recipe_ingredients, units, ingredient_nutrition_100g, ingredient_tags, ingredient_tag_map, user_recipe_actions, recipe_nutrition_cache.

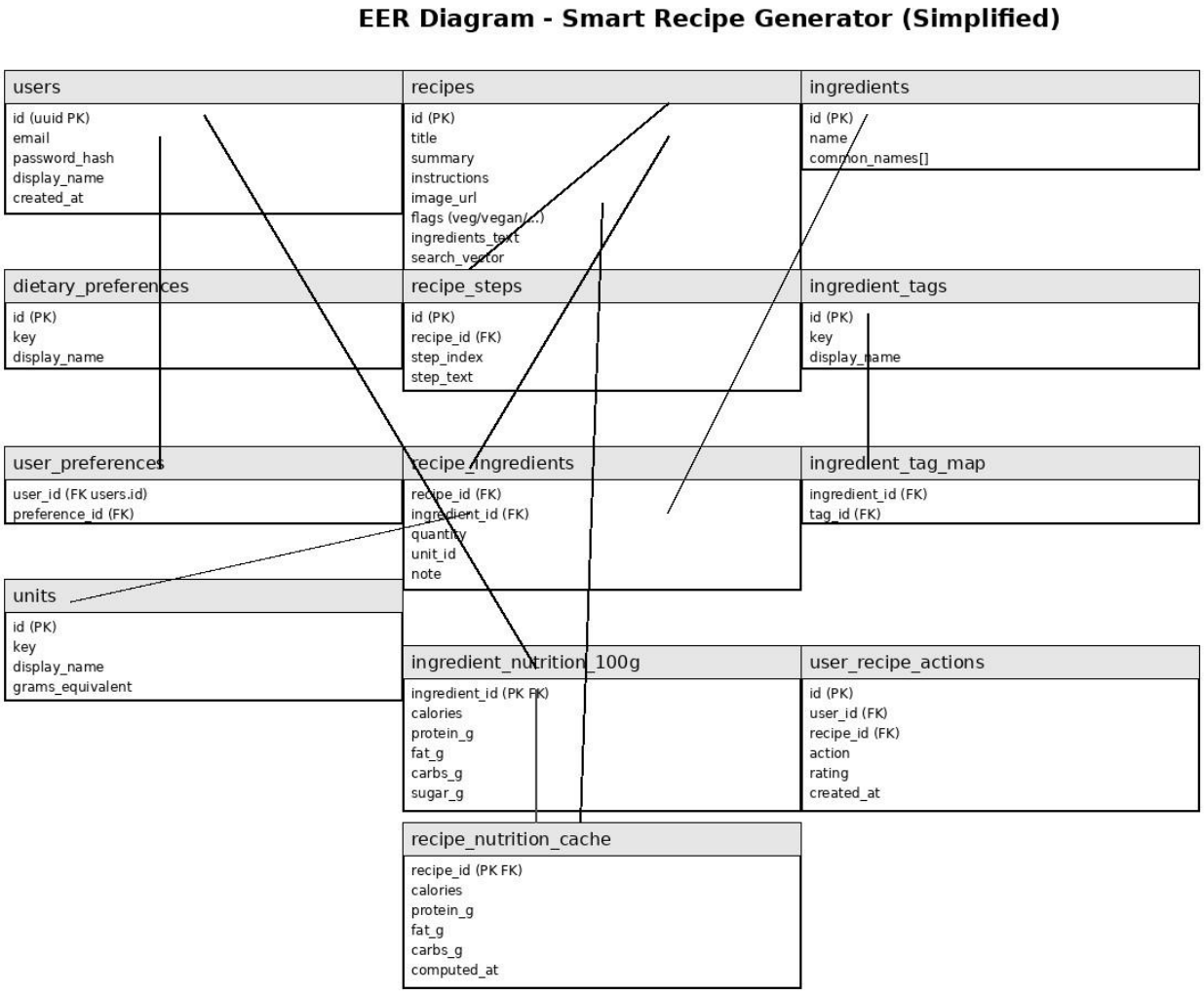
- Use tsvector + GIN index on recipes.search_vector for fast text search. - Store images in Supabase and only save image URLs in recipes.image_url.

- Use recipe_nutrition_cache for fast retrieval of recipe-level nutrition values.

Key Features

-Ingredient recognition via Clarifai API (multi-label)
- Fast recipe retrieval using PostgreSQLfull-text search with tsvector and GIN index
- Dietary filtering (vegetarian, vegan, gluten-free, dairy-free, etc.)
- Nutrition calculation via ingredient nutrition data and recipe_nutrition_cache User accounts, history, favorites, and ratings
- Images hosted on Supabase Storage - Secure backend with JWT authentication
- CI/CD with GitHub Actions and deployment on Render

EER Diagram - Database Structure



Legend: PK = Primary Key, FK = Foreign Key. search_vector uses tsvector + GIN index.
Use tsvector on recipes.title and recipes.ingredients_text to perform text search (plainto_tsquery / websearch_to_tsquery) and rank via ts_rank.

Deployment Checklist

Deployment Checklist

- [ ] Migrations tested locally and in CI

- [ ] Render Postgres created and tested

- [ ] Environment variables configured in Render (CLARIFAI_KEY, JWT_SECRET, SUPABASE_URL, SUPABASE_KEY)

- [ ] GitHub Secrets added for CI (RENDER_DATABASE_URL, CLARIFAI_KEY, JWT_SECRET) - [ ] Migrations applied to Render DB (manual or CI)

- [ ] Web service deployed on Render - [ ] Smoke tests passed on deployed app

Future Enhancements

Future Enhancements

- Add collaborative filtering for personalized recommendations

- Add direct image upload from frontend to Supabase with signed URLs

- Implement caching (Redis) for popular searches

- Add analytics to track feature usage and recommendation quality

Notes and Implementation Tips

Notes and Implementation Tips

- Keep .env out of Git; use .env.example for documentation.

- Use a small connection pool (max 5) for Render Postgres free tier.

- Mock Clarifai in tests to avoid API usage in CI.

- Run migrations from CI or a single admin workflow to avoid race conditions.