Heritage Recipes Lite

Project Architecture and Technical Documentation

Author: Paramveer Gupta

Date: 2025-11-17

1. Project Overview

Heritage Recipes Lite is a simple recipe-finder mobile application built with Flutter for Android and a Node.js + Express backend using MongoDB for data persistence. The app allows users to register and log in, browse and search recipes, view recipe details, save favorites, and add/edit recipes. The backend exposes RESTful APIs for authentication and recipe CRUD operations.

Main Goals

• Provide a minimal, reliable mobile UI for browsing and saving traditional recipes.
• Allow authenticated users to add, edit, and manage recipes.
• Keep the architecture simple and well-documented for the IB Computer Science IA.

2. Frontend (Flutter)

Tech stack & tools

• Flutter (Dart) for UI and app logic
• http package for REST API calls
• shared_preferences for token storage
• StatefulWidget-based screens for simplicity

Project structure (key files)

The Flutter project is in `heritage_recipes_lite/` with key paths:

• `lib/main.dart` - app entry and route definitions
• `lib/screens/` - pages (login, home, add recipe, recipe details)
• `lib/models/recipe_model.dart` - Recipe model and JSON parsing
• `lib/services/api_service.dart` - REST client and token management
• `lib/widgets/` - UI components (e.g., recipe card)

Data flow

The app communicates with the backend via JSON REST APIs. `ApiService` manages token storage and attaches the Authorization header. Responses are parsed into `Recipe` model objects. UI screens call service methods, await results, and update the widgets via setState.

Important implementation notes

• Recipe model handles populated userId objects and converts them into string IDs.
• Error handling in `ApiService` prints parsing errors to help debugging.
• Add/Edit recipe flow uses the same screen; route arguments indicate edit mode.

How to run (development)

Ensure the backend API is reachable (use 10.0.2.2 for Android emulator or a LAN/ngrok URL for devices). Then:

1. Install Flutter SDK and dependencies: `flutter pub get`
2. Run on emulator: `flutter run`
3. Build release APK: `flutter build apk --release`

3. Backend (Node.js + Express)

Tech stack & tools

• Node.js + Express for REST API
• MongoDB (Mongoose) for data storage
• bcryptjs for password hashing
• jsonwebtoken for JWT authentication
• nodemon for development auto-reload

Project structure (key files)

• `server.js` - starts server and mounts routes
• `config/db.js` - connects to MongoDB using `process.env.MONGODB_URI`
• `models/User.js` - user schema and password hashing
• `models/Recipe.js` - recipe schema and text index for search
• `routes/auth.js` - register/login endpoints
• `routes/recipes.js` - recipe CRUD and favorite endpoints
• `scripts/seedTestUser.js` / `seedRecipes.js` - helper scripts to create test data

Environment & configuration

Environment variables are stored in `.env` (not committed). Use `.env.example` as a template. Required variables include: `MONGODB_URI`, `JWT_SECRET`, `JWT_EXPIRE`, and `PORT`. For stable hosting, use MongoDB Atlas and set `MONGODB_URI` to the atlas connection string.

Key API endpoints

• POST /api/auth/register - register a new user
• POST /api/auth/login - login and receive JWT
• GET /api/recipes - list recipes (supports search and category query)
• GET /api/recipes/:id - get recipe details
• POST /api/recipes - create recipe (authenticated)
• PUT /api/recipes/:id - update recipe (authenticated, owner only)
• DELETE /api/recipes/:id - delete recipe (authenticated, owner only)
• POST /api/recipes/:id/favorite - toggle favorite

Data models

User schema fields: name, email, password (hashed), favorites array

Recipe schema fields: title, description, ingredients (array of strings), instructions (array of strings), imageUrl, category, prepTime, cookTime, servings, userId (ObjectId ref to User)

Seeding and developer scripts

The `scripts/` folder includes helpers:

• `seedTestUser.js` - creates `test@example.com` / `test1234` user
• `seedRecipes.js` - inserts sample recipes and links them to the test user
• `manageUsers.js` - interactive CLI to create/list/delete users
• `importCSV.js` - import recipes from CSV into the DB

How to run backend locally

1. Install dependencies: `npm install`
2. Copy `.env.example` to `.env` and set `MONGODB_URI`
3. Seed data: `npm run seed-test-user` and `npm run seed-recipes`
4. Start dev server: `npm run dev`

4. Deployment & Hosting Options

Backend hosting (simple)

Use Render, Railway, or Heroku-like services for easy Node.js deployment. Connect the GitHub repo, set environment variables in the platform, and deploy. Use MongoDB Atlas for the database and point `MONGODB_URI` to the cluster.

Mobile distribution

• Build a release AAB/APK and publish to Google Play (requires a developer account)
• Use Firebase App Distribution for private testing groups
• Use CI providers (GitHub Actions / Codemagic) to automate builds and releases

Security & production considerations

• Keep `.env` out of source control
• Use strong `JWT_SECRET` and rotate credentials if needed
• Enforce HTTPS in production
• Validate and sanitize user inputs on the backend
• Add rate limiting and logging for production readiness

5. Appendix - Important files and commands

Frontend quick commands

• flutter pub get
• flutter run
• flutter build apk --release
• Edit API URL: lib/services/api_service.dart

Backend quick commands

• npm install
• npm run dev
• npm run seed-test-user
• npm run seed-recipes
• npm run generate-ia-pdf
• npm run generate-arch-pdf

Generated files are placed in the `backend/` directory: `IA_Report.pdf` and `Architecture_Report.pdf`.

Generated by project tooling. Author: Paramveer Gupta