Final Project Documentation: EasyCal

SE/COM S 3190 – Construction of User Interfaces
Spring 2025

Brian Craciun – bcraciun@iastate.edu
Muhammad Blal – mblal@iastate.edu

Contents

1 Introduction

EasyCal is a full-stack nutrition and recipe planning website that helps users become their healthier selves. It focuses on providing tools like popular recipes, and restaurants to get users going with their plans. Also, we have recently included the recipeAI to help deliver quick and easy meal ideas to all consumers. The target audience includes fitness enthusiasts, students, and anyone tracking their diet. Inspired by the fact that there is no secure and safe feeling nutrition website, EasyCal provides a clean UI with AI assistance and CRUD functionality for users to find their best recipes and restaurants.

2 Project Description

- **Home Page:** Displays featured content, images, and calls-to-action. The homepage also leads to all other popular features.
- **Food Search:** Nutritionix-powered API search. Returns amazing nutritional breakdowns of popular foods.
- **Popular Recipes/Restaurants:** MongoDB-integrated views with full CRUD.
- **RecipeAI:** AI-assisted recipe generator + nutrition lookup (using Nutritionix).
- **Login Page:** Simple dev login for test access as well as editing current recipes and restaurants. This is hidden in the footer as to not distract the user from the important features of our actual website.

CRUD Entities:

- Recipes — `GET`, `POST`, `PUT`, `DELETE`
- Restaurants — `GET`, `POST`, `PUT`, `DELETE`
- Team/devLogins — `GET`, `POST`

  Allows for developers who are signed in to edit both recipes and restaurants using CRUD style backend calls.

3 File and Folder Architecture

We kept everything compact and organized in the frontend, backend, and Documents folders. In the front end we organized components and services to keep our workflow clean. On the backend, we only had a few files so organization was not as key, but naming conventions were very important. Lastly, our Documents folder had everything from our mini assignments as well the final project videos that were recorded.

We had no static files, but we did have two service files. One was a python file aiding in API calls to run ollama ai, and then we had a nutritionix API service to help make calls to the API to generate nutrition labels for both the AI's post processing result, and the Food Search function on our site.]

Below is a tree of our overall file structure:

```
project_JK_13/
|
+-- frontend/
|   +-- components/
|   |   +-- Authors.jsx
|   |   +-- FoodSearch.jsx
|   |   +-- Footer.jsx
|   |   +-- Home.jsx
|   |   +-- LoginPage.jsx
|   |   +-- Navbar.jsx
|   |   +-- PopularRecipes.jsx
|   |   +-- PopularRestaurants.jsx
|   |   \-- RecipeAI.jsx
|   +-- services/
|   |   \-- nutritionApi.jsx
|   +-- App.css
|   +-- index.css
|   +-- App.jsx
|   \-- main.jsx
|
+-- backend/
|   +-- server.js
|   +-- aiServer.py
|   +-- routes/
|   |   \-- devLogins.js
|   +-- package.json
|   \-- README.md
|
\-- Documents/
    +-- Mini-Assignments/
    +-- FinalVideos/
    +-- JK13_Final_Proposal.pdf
    \-- JK_13_Technical_Report.pdf
```

4 Code Explanation and Logic Flow

Frontend uses React with routing and component state via hooks. Backend uses Express to handle MongoDB and AI endpoints.

We use useState and props in our components by managing things like input fields, modal visibility and form state, dynamic rendering of data from our database, and using props like setStep for redirection in something like Home.jsx

React Component loading recipes from the backend using useEffect() from MongoDB

```
useEffect(() => {
  fetch("http://localhost:3000/api/recipes")
    .then((res) => res.json())
    .then((data) => setRecipes(data))
    .catch((err) => console.error("Error fetching recipes:", err));
}, []);
```

Backend Route Snippet: server.js (POST New Recipe)

```
app.post('/api/recipes', async (req, res) => {
  try {
    const result = await recipesCollection.insertOne(req.body);
    res.status(201).json(result);
  } catch (err) {
    res.status(500).json({ error: 'Failed to create recipe' });
  }
});
```

Ollama and Uvicorn API Integration Snippet: server.js (AI-Powered Recipe Generator) (also uses Axios)

```
app.post('/api/generate-recipe', async (req, res) => {
  const { age, weight, height, exercise, meal } = req.body;
  const prompt = `You are a professional AI recipe assistant...`;

  const response = await axios.post('http://localhost:11434/api/generate', {
    model: "tinyllama:1.1b-chat",
    prompt,
    stream: false
  });
```

```
  const output = response.data.response?.trim() || '';
  res.json({ output });
});
```

5 Web View Screenshots

Below are annotated screenshots of each major view in the EasyCal application. These illustrate the functionality and user interface design across key features.

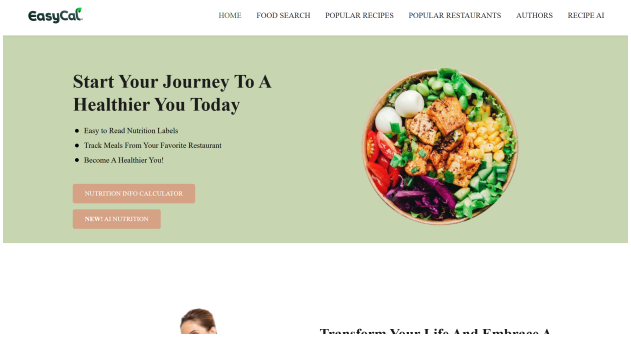

Figure 1: Home Page: User can see what EasyCal has to offer as well as be redirected to many of our other more useful pages.

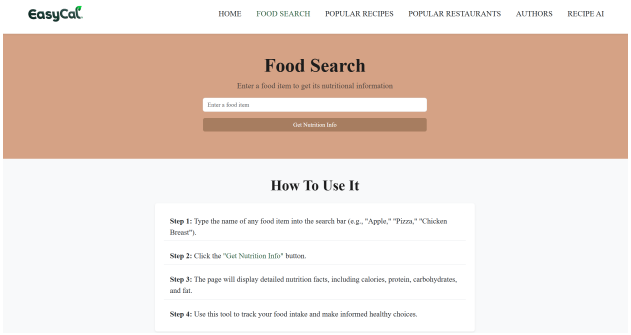

Figure 2: Food Search: Allows the user to enter any meal and receive the nutritional information for that food.

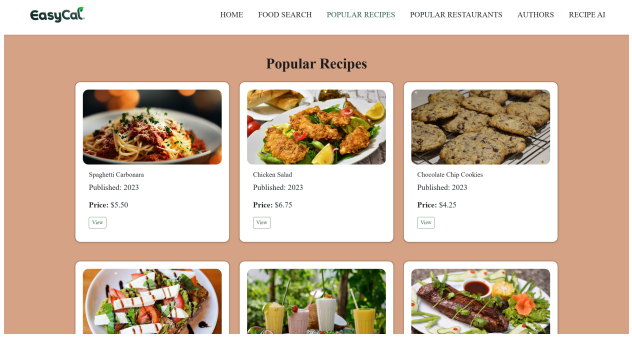

Figure 3: Popular Recipes: A list of popular recipes users can try out. Pressing on view opens a modal revealing the recipe, and the price of each recipe is listed on the front.

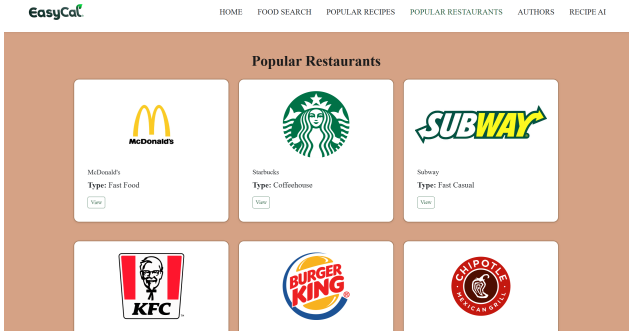

Figure 4: Popular Restaurants: A list of popular restaurants users can see. There is also a modal that shows some popular meal choices.

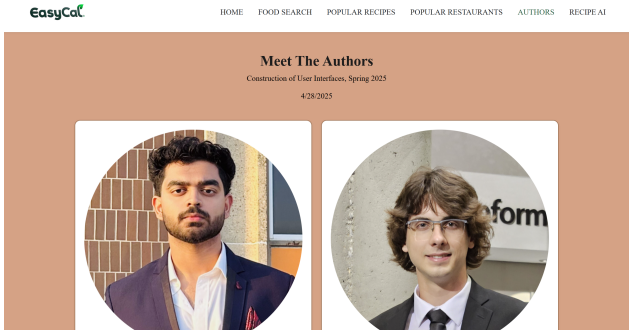

Figure 5: Authors: Displays the Author's of the site and their contact information.

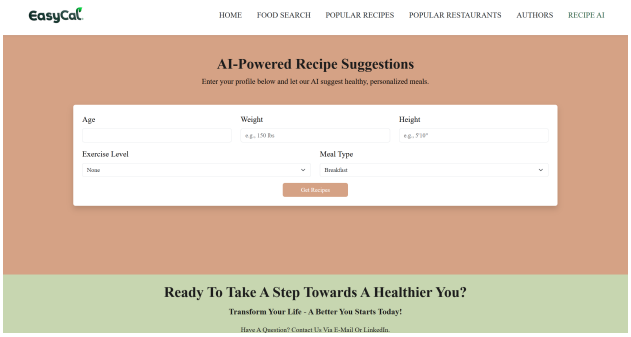

Figure 6: recipeAI: AI generated meals for breakfast, lunch, and dinner where the user can see immediate nutritional facts after generation.

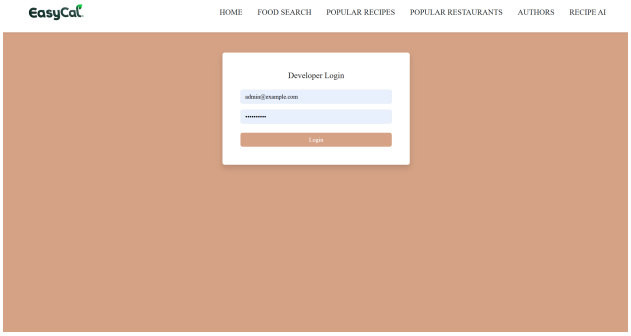

Figure 7: Developer Login: Login for developer to gain access to CRUD functionality.

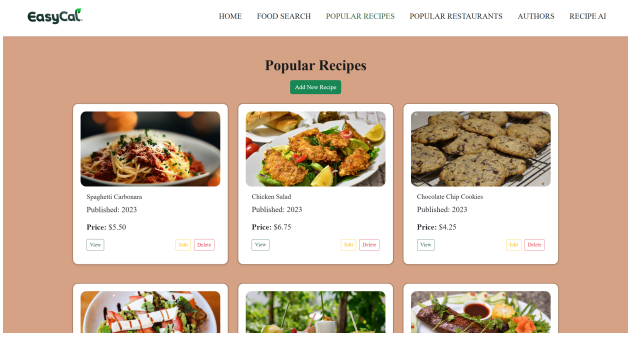

Figure 8: CRUD Popular Recipes: Developer view and crud function for popular recipes.

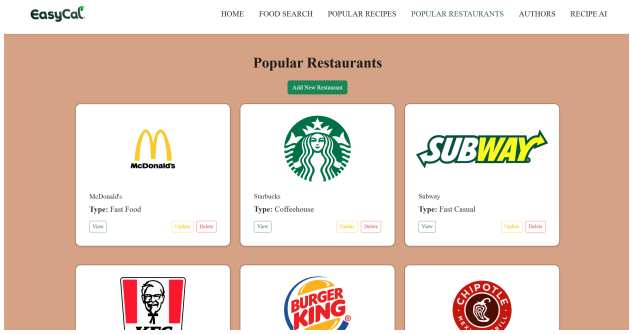

Figure 9: CRUD Popular Restaurants: Developer view and crud function for popular restaurants.

6 Installation and Setup Instructions

EasyCal Project Setup

Welcome to **EasyCal** — an AI-powered nutrition and recipe app! This guide explains how to set up and run the full stack locally.

Prerequisites

Before beginning, ensure the following tools are installed on your system:

- Node.js (v16 or higher)
- Python 3.9+
- `pip` (Python package manager)
- MongoDB (local or remote)
- Ollama — download from: `https://ollama.com/download`

Installation

Backend Dependencies

In the `backend/` folder, run:

```
npm install
pip install fastapi uvicorn requests
```

Frontend Dependencies

In the `frontend/` folder, run:

```
npm install
```

How to Run the Project

You will need **4 terminal windows** to run the system concurrently.

React Frontend

```
cd frontend
npm run dev
```

Runs at: `http://localhost:5173/`

Node.js Express Backend

```
cd backend
npm run dev
```

Runs at: `http://localhost:3000/` Connects to MongoDB.

AI Python FastAPI Server

```
cd backend
uvicorn aiServer:app --host 127.0.0.1 --port 5005
```

Runs at: `http://127.0.0.1:5005/`

Ollama AI Model

First-time setup:

```
ollama pull tinyllama:1.1b-chat
```

To run every time:

```
ollama run tinyllama:1.1b-chat
```

Runs at: `http://localhost:11434/`

Summary

- Frontend → React App at `http://localhost:5173`
- Backend → Node.js Express API at `http://localhost:3000`
- AI Server → Python FastAPI at `http://localhost:5005`
- Ollama Model → LLM server at `http://localhost:11434`

7 Contribution Overview

- Brian Craciun — Initializing all Frontend Components, completing the footer and navbar for the site. Fully applying and integrating the Nutritionix API, as well as integrating and filtering results for RecipeAI. Equal contribution to server.js.
- Muhammad Blal — Worked on finalizing all components. Linked MongoDB components, and created the developer login for access to CRUD operation on both PopularRecipe.jsx and PopularRestaurants.jsx. Equal contribution to server.js.

8 Challenges Faced

- Problem: Handling AI output formatting for consistent UX.
- Solution: Strictly Remove and reformat AI output, and recall the AI if the output is completely undesired.
- Problem: Bootstrap modal behavior across multiple CRUD routes.
- Solution: Read into Bootstrap documentation and call certain imports that were missing from the project.
- Problem: MongoDB document validation and ID mismatch bugs.
- Solution: Check and Validate all ID's for them to be uniform, and then modofiy how they are being stored if needed.

9 Final Reflections

This project taught us to integrate multiple technologies like React, Node.js, MongoDB, Python, and AI models like Ollama into one application. We also learned to pace ourselves to meet deadlines and produce the best product in the given period.

Next time, we would organize our project ideas differently. We were too fragmented with our

ideas and goals that it became hard to materialize them when it came to implementation. It is essential to understand the time frame and the work your team can accomplish.

In the future, Muhammad and I might look into deploying this app to increase the number of public open projects we have for our Resumes and Portfolios. We also couldack into it and refactor our code to make it a possible product in the future where users can pay to receive exclusive services that no other site may offer.