[Overview 2](#_7nh13eb1s3na)

[I. Backend Assignment: Meal Calorie Count Generator 2](#_1v4i2hxpw6br)

[Objective 2](#_xn1smclwvoyw)

[Functional Requirements 2](#_4bkuyrj85r9a)

[Tech Stack 3](#_ea7u5c4jz72v)

[API Endpoints 3](#_w5o8qse5swlx)

[Implementation Notes 4](#_yqqajsl49e8l)

[Testing 4](#_1kkut5pf4scc)

[Security & Limits 5](#_8zqvtmta6l0r)

[Extension Ideas 5](#_hvjgi6mucfyn)

[Grading Criteria - 5](#_8l9vgwbnfop6)

[II. Frontend Assignment: Meal Calorie Count Generator – Web App 6](#_f12a1mllh0tm)

[Objective 6](#_m1j02gcut2t3)

[Tech Stack Requirements 6](#_t04tvwd7g8n1)

[Required Features - 6](#_3ymqy5djkhm3)

[Sample API Contracts (Backend Application) 7](#_1ta5nlrutdu4)

[Bonus Features (Optional) 8](#_iv76j96diwr8)

[Suggested Folder Structure (Next.js App Router) 8](#_4plg1u8ek0s4)

[Authentication Flow 9](#_3ozgf932ytxf)

[Testing Expectations (Optional) 9](#_62pu9nihcfxs)

[Submission Instructions 9](#_ibdkzhje4paq)

[Evaluation Criteria 10](#_1jb8qtlj706p)

[Deadline 10](#_tuvf53mrzfll)

[Questions? 10](#_729vu9uwm9bi)

**Guidelines**

# Overview

This document has an assignment to evaluate

* Backend
* Frontend

Important Notes -

* Attention to detail
* Reading and understanding the submission instructions

# *Backend Assignment: Meal Calorie Count Generator*

## Objective

Create a backend service that allows users to input a dish name and number of servings and returns the total calorie count using the free USDA FoodData Central API.

## Functional Requirements

| Feature | Description |
| --- | --- |
| Dish Search | Users input the name of a dish (e.g., 'pasta alfredo') and the system searches for it using the USDA API. |
| Serving Count | Users specify the number of servings. |
| Calorie Calculation | The system calculates total calories based on the nutrition data retrieved and the number of servings. |
| API Response | Returns structured JSON with total calories, calories per serving, and ingredient breakdown if possible. |

## Tech Stack

| Component | Technology |
| --- | --- |
| Language | Node.js (Express) or Python (FastAPI) |
| External API | USDA FoodData Central API |
| Hosting (Optional) | Render, Railway, Vercel, or local server |
| Rate Limiting | Optional: Use middleware like expressrate-limit or FastAPI dependencies to ratelimit beyond **15** rapid requests from the same IP |

## API Endpoints

***POST `/get-calories`***



| Parameter | Type | Description |
| --- | --- | --- |
| dish\_name | string | Name of the food/dish |
| servings | number | Number of servings |

Sample Request Body:

***{***

***"dish\_name": "chicken biryani",***

***"servings": 2***

***}***



Sample Response:

***{***

***"dish\_name": "chicken biryani", "servings": 2,***

***"calories\_per\_serving": 280,***

***"total\_calories": 560,***

***"source": "USDA FoodData Central"***

***} Integration with USDA API***



* API Endpoint: https://api.nal.usda.gov/fdc/v1/foods/search
* Method: GET
* Params: query, api\_key, pageSize
* API Key: Obtain from https://fdc.nal.usda.gov/api-key-signup.html

## Implementation Notes

* Use fuzzy matching to select the best-matching food item from USDA API results.
* Fetch calories per 100g or per serving depending on availability.
* Multiply calories by number of servings.
* Cache frequent queries for performance (optional).
* Implement authentication
* User to input first name, last name, email, and password
* Persist authentication data in encrypted format along with best practices for validation of email and password and security protocols.
* Choice of Database – MongoDB or Postgres

Authentication should be implemented using the following nested routes only-->

**/auth/register - for new user signUps**

**/auth/login - for logins**

## Testing

* Test with common dishes: 'macaroni and cheese', 'grilled salmon', 'paneer butter masala'.
* Include tests for:
* - Non-existent dishes
* - Zero or negative servings
* - Multiple similar matches

## Security & Limits

* Store API key securely (e.g., in .env)
* Rate-limit API to prevent abuse
* Return friendly errors (e.g., 'Dish not found', 'Invalid servings')

## Extension Ideas

* Add macronutrients (carbs, fat, protein) to output.
* Support partial matching and user feedback for food selection.
* Add a front-end interface (React or simple HTML form).
* Add a meal log or nutrition tracker feature.

## Grading Criteria -

Your submission will be graded on the folowing criteria -->

OOP Practices

Code Modularity

API Accuracy

Algorithmic Sophistication – Data Processing Logic; Performance Optimizations, etc.

.**env Variables -**

Don’t forget to include an .env.example file with relevant placeholders for .env variables referenced in your codebase

# *Frontend Assignment: Meal Calorie Count Generator – Web App*

## Objective

Build a fully responsive, production-ready **front end interface** for the **Meal Calorie Count Generator** backend. For documentation on the backend application, please refer to the readme.MD file in the supplied folder titled “meal-calorie-count-generator-main". The UI should allow users to register, login, input dish details, and view calorie data returned by the backend API.

The frontend must reflect thoughtful **UX, state management, component structure**, and **SSR-friendly rendering**.

## Tech Stack Requirements

| **Layer** | **Tooling/Framework** |
| --- | --- |
| **Framework** | **Next.js (App Router)** or **React 18 + Vite** with SSR |
| **Styling** | Tailwind CSS + **shadcn/ui** |
| **State** | Zustand |
| **Validation** | Zod (optional: react-hook-form) |
| **Language** | TypeScript only |
| **Testing** | (Optional but encouraged) Vitest / Playwright |
| **Package Manager** | pnpm or npm |

## Required Features

| **Feature** | **Description** |
| --- | --- |
| **User Registration** | Auth form (First Name, Last Name, Email, Password). Calls /auth/register endpoint. |
| **User Login** | Auth form (Email + Password). Stores token securely (localStorage or HTTP-only cookie preferred). |
| **Calorie Lookup Form** | Form to input dish name & servings. Sends data to /get-calories and renders the calorie breakdown. |
| **Calorie Results Card** | Shows: dish name, servings, calories per serving, total calories, and data source. |
| **User Feedback** | Show loading spinner, success alert, error alerts (dish not found, invalid servings, etc.). |
| **Responsive Layout** | Mobile-first layout that also looks great on desktop. |
| **State Management** | Use Zustand to persist auth token, user info, and optionally previous meals. |
| **Dark/Light Mode** | Use shadcn's theme toggle for styling consistency. |

## Sample API Contracts (Backend Application)

**Register a User** -> POST /register:

json CopyEdit

{

"first\_name": "John",

"last\_name": "Doe",

"email": "john@example.com",

"password": "secure123"

}

**Login** -> Log the user in and **get the necessary authorization payload** | POST /login:

{

"email": "john@example.com",

"password": "secure123"

}

**Get Calories -> The main functionality of the application |**  POST /get-calories

{

"dish\_name": "chicken biryani",

"servings": 2

}

Sample Response:

{

"dish\_name": "chicken biryani",

"servings": 2,

"calories\_per\_serving": 280,

"total\_calories": 560,

"source": "USDA FoodData Central"

}

### 🧠 Bonus Features (Optional)

These are **not required**, but completing them will give extra credit:

* **Meal History Log** — Display a table of past calorie searches (in memory or persisted via backend if extended).
* **Dockerized Dev Environment** — Wrap your frontend in a Dockerfile + dockercompose config.
* **SEO + Meta Tags** — Add <Head> metadata for calorie lookup page.
* **Basic Testing** — Write 2–3 test cases for form and result display using Vitest or Testing Library.
* **Guarded Routes** — Prevent unauthenticated users from accessing the dashboard or calorie form.

### 🗂️ Suggested Folder Structure (Next.js App Router)

bash /src

/app

/login

/register

/dashboard

/calories

/components

|  | AuthForm.tsx |
| --- | --- |
|  | MealForm.tsx |
| /lib | ResultCard.tsx |
|  | api.ts |
|  | auth.ts |

/stores

authStore.ts mealStore.ts

/types

/styles

.env.example

README.md

If using **Vite**, structure under src/pages, src/components, src/lib, etc., and implement SSR using Vite SSR adapter.

### 🔐 Authentication Flow

* Use **Zustand** to manage and persist authStore with the JWT token.
* After login or registration, **redirect** the user to /dashboard.
* For protected pages (like dashboard), use a useAuthGuard() hook to redirect unauthenticated users.

### 🧪 Testing Expectations (Optional)

| **Scope** | **Tool** | **Description** |
| --- | --- | --- |
| Compone  nt | Vitest + React Testing  Library | Test MealForm logic |
| Integrati on | Playwright | Test full login → search → result rendering |

### 🧾 Submission Instructions

1. **Create a GitHub repository**: meal-calorie-frontend-{your-name}.
2. Include:
   1. README.md with:
      1. Setup instructions
      2. Any decisions or trade-offs
      3. Screenshot(s) iv. Hosted link (if deployed)
   2. .env.example with:
      1. NEXT\_PUBLIC\_API\_BASE\_URL=http://localhost:8000

1. Push your code and share the repo link.
2. Deploy on vercel (or something like that) and share app url

### 🧠 Evaluation Criteria

| **Area** | **Weight** |
| --- | --- |
| Code Quality (TS, folder structure, modularity) | 25% |
| API Integration Accuracy | 20% |
| UX, UI, Responsiveness | 20% |
| Zustand Implementation | 15% |
| Error Handling & Validation | 10% |
| (Bonus) Testing & Extras | 10% |

### 🕒 Deadline

Submit within **72 hours** of receiving the assignment. If you need extra time, communicate early.

#### 💬 Questions?

Feel free to ask for clarifications before you start.

Let us see what you've got. Happy building!