Primal Fit

By

404 NOT FOUND

<Undergraduate>

- Abdelaziz Saleh
- David Tincher
- Jinho Nam

<Graduate>

- Dilean Munoz
- Revanth Reddy Banala

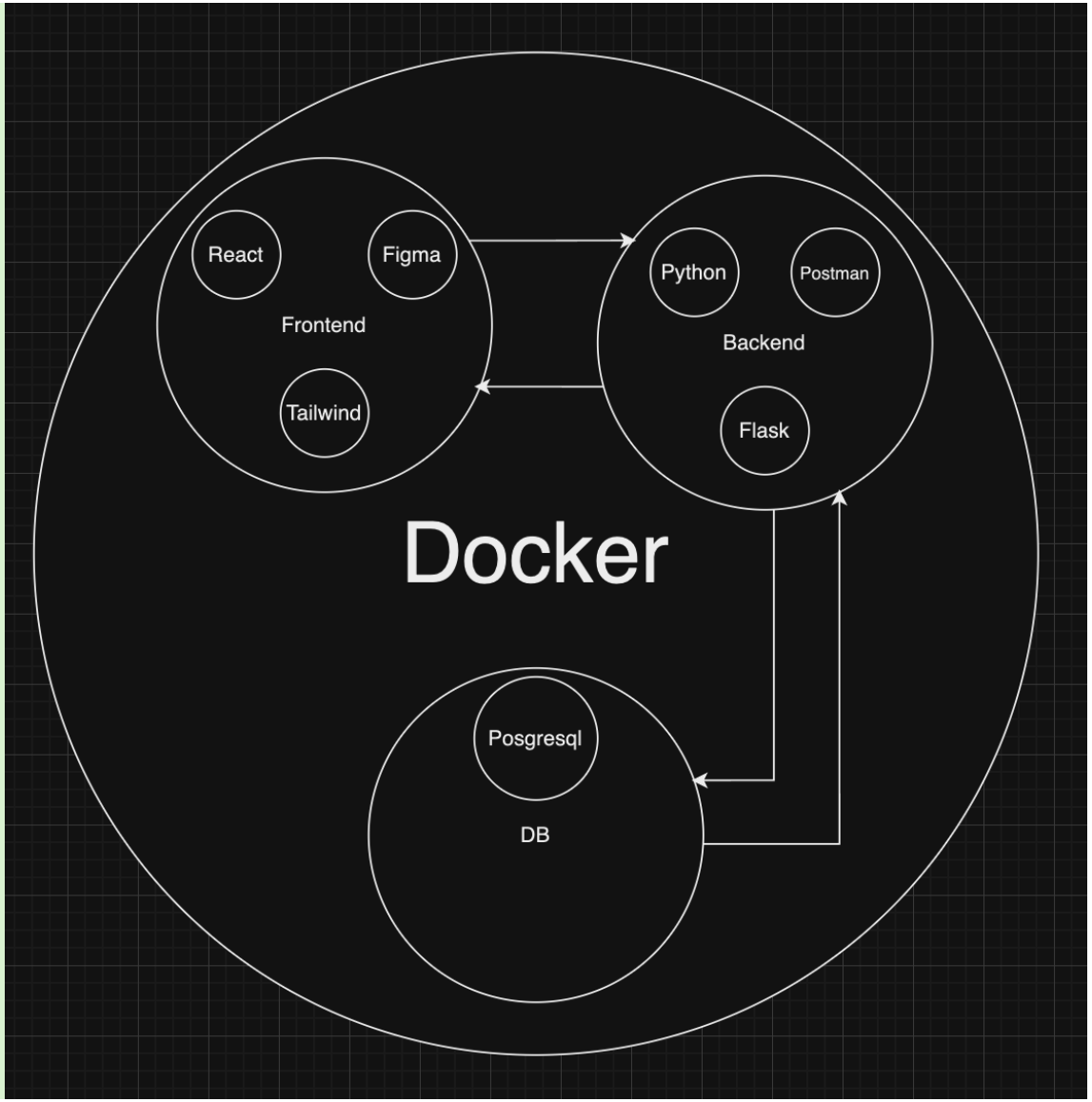
Introduction

- Primal Fit is a fitness app that helps users achieve their goals toward being healthy.

- The app has some features, including tracking calories, nutrition intake, customizing routines with user's choice of exercises.

- With all these tools, and our mascot, Primal Fit will make you keep motivated through your fitness journey.

Architecture Overview



User Stories

“I want to be able to schedule routines for each day with workout options to choose from and be displayed with an instructional video, because I will benefit from watching someone do the exercise with the correct form.”

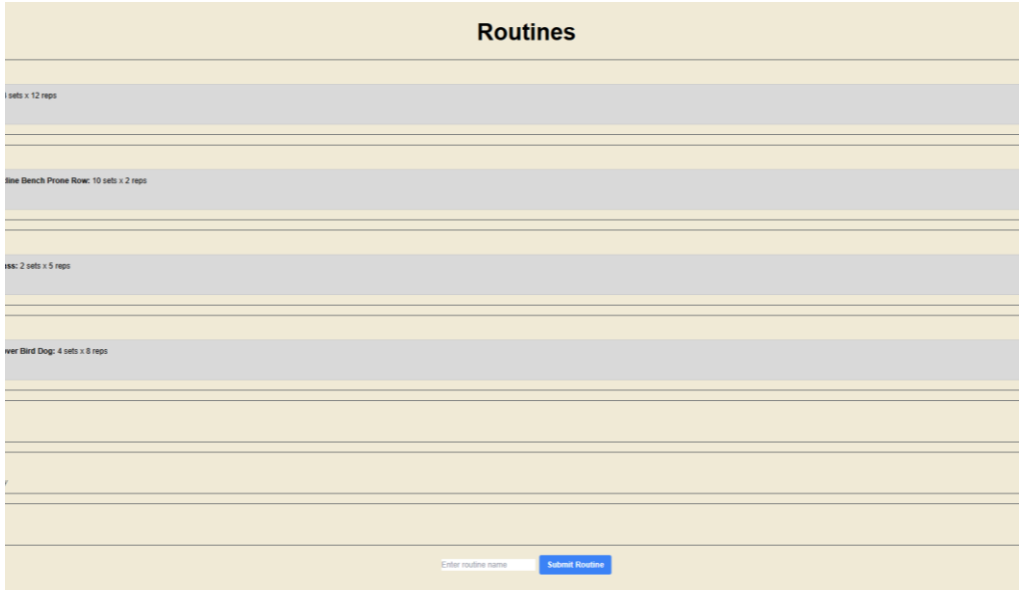
“I want to be able to track how many calories I have burned as I perform my workout, and get updates as I exercise to have motivation to not give up while I work out.”

“when visiting the website for the first time I would like to be able to create an account so my progress can be saved for later use.”

“I want to be able to search common foods and input them into my nutritional goals for the day so that I can track my daily nutrition.”

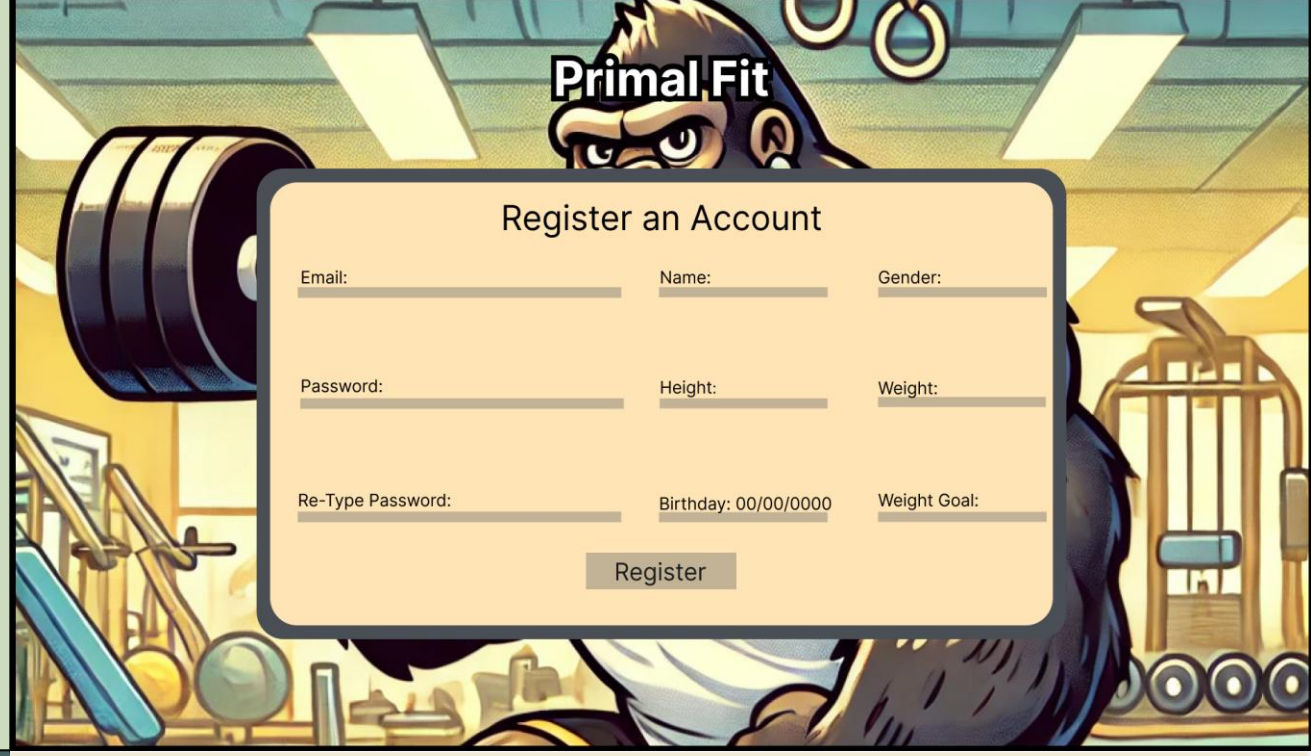
Requirements

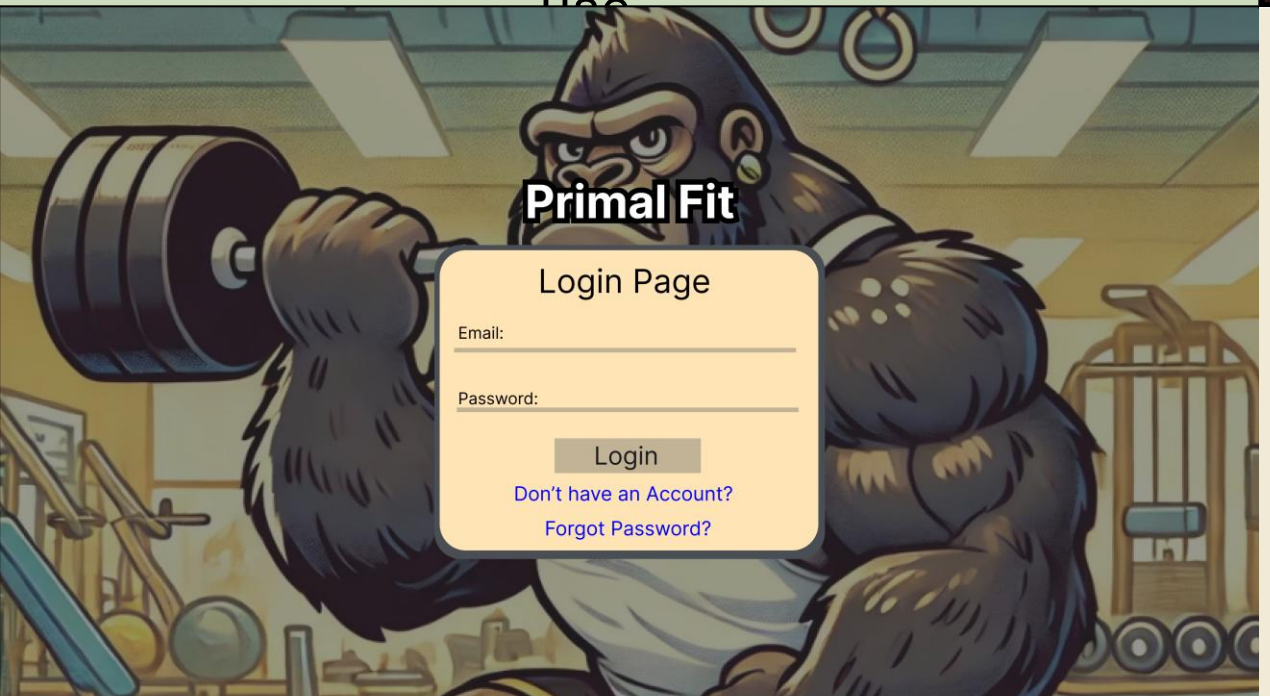
R1. The user shall be able to schedule routines for each day with workout options to choose from and be displayed with an instructional video.

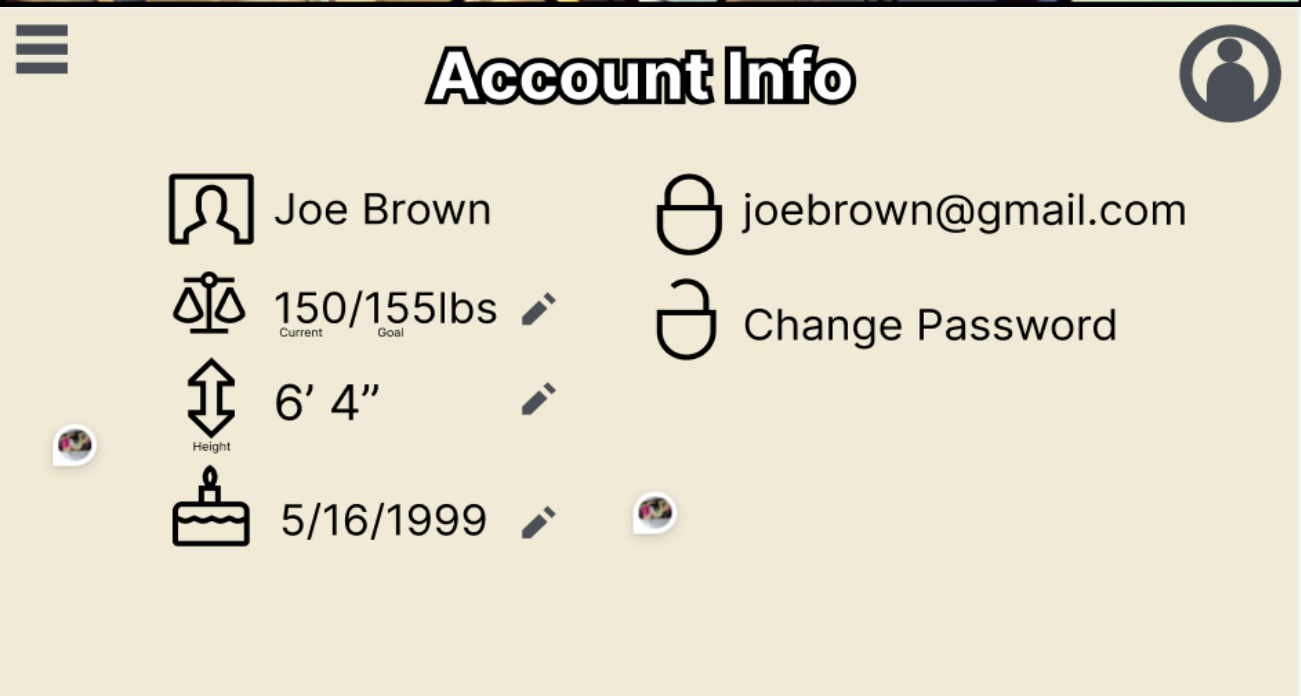


R2. The user shall be required to create an account when accessing the website for the first time.

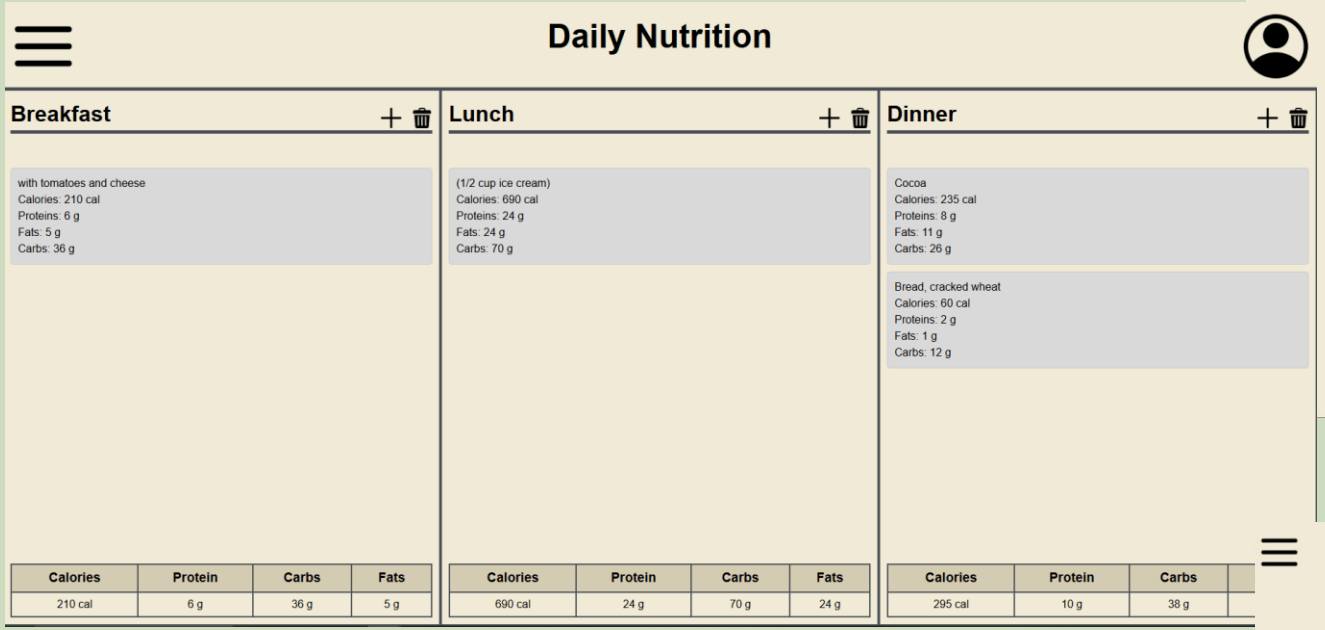
R3. The user shall be able to login to the website and have their data saved for later



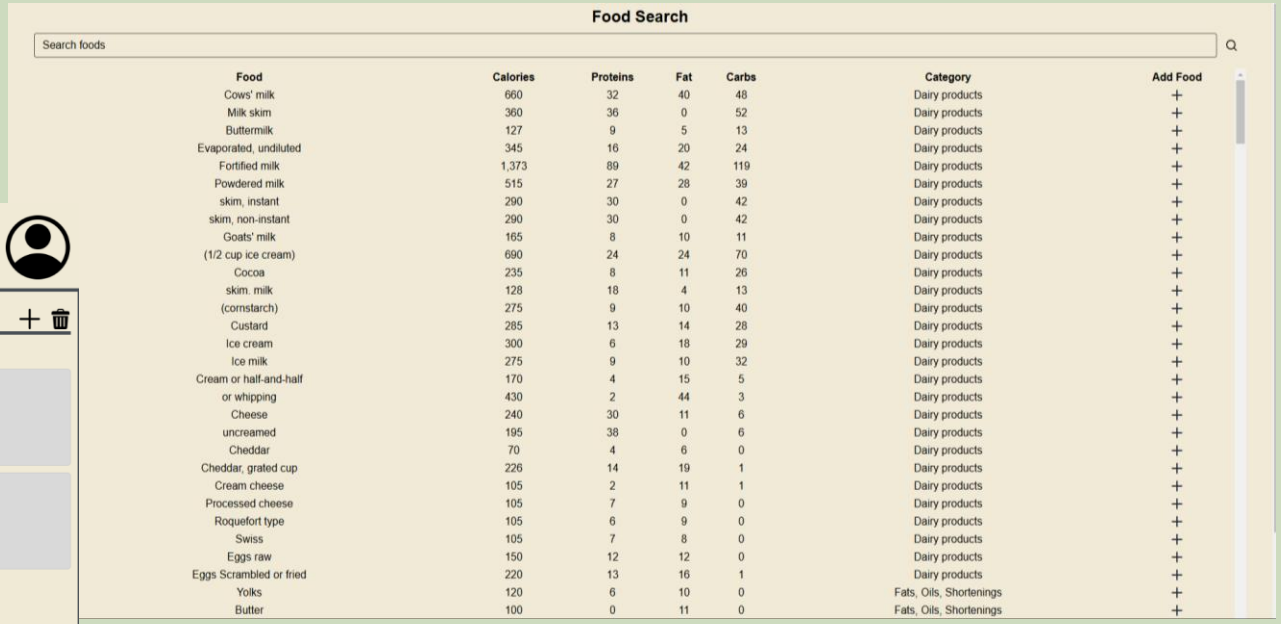




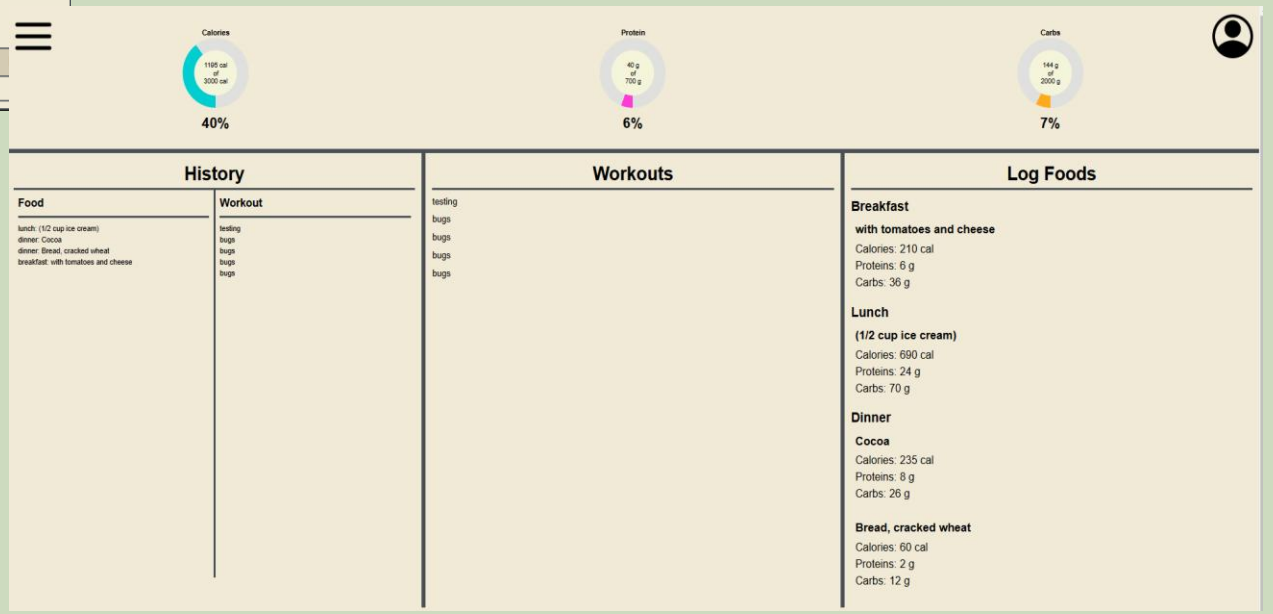
R4. A user shall be able to search common foods and input them into their meals for the day.



<Fig 5: Foods and Daily Nutrition intake>



<Fig 4: Food Search function>



<Fig 6: Daily Nutrition Progress in Graphs>

Design

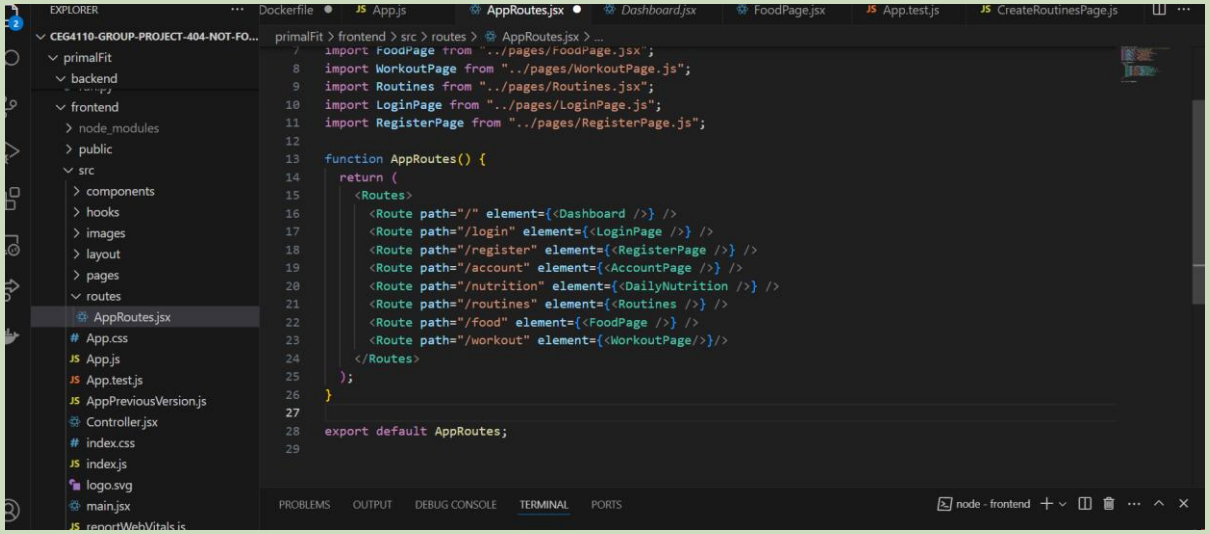
Complexity of design at front-end side

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@heroicons/react": "^2.1.5",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.7.7",
    "date-fns": "^4.1.0",
    "papaparse": "^5.4.1",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.27.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "devDependencies": {
    "@types/papaparse": "^5.3.15",
    "tailwindcss": "^3.4.14"
  },
```

```
/** @type {import('tailwindcss').Config} */
module.exports = {
    content: [
      "./src/**/*.{js,jsx,ts,tsx}",
    ],
    theme: {
      extend: {
        colors: {
          borderColor: '#494F55',
          borderColorBottom: '#494F5554',
          mainBackgroundColor: "#F0EAD6",
        },
      },
    },
    plugins: [],
  }
```

Design

Complexity of design at front-end side



Design

FoodPage.jsx

```jsx
import React, { useEffect, useState } from "react";
import SearchBar from "../components/SearchBar";

import useFetch from "../hooks/useFetch";
import FoodTable from "../components/FoodTable";
import { useLocation } from "react-router-dom";

export default function FoodPage() {
  const { state } = useLocation();

  const mealType = state.toLowerCase();

  const [foodList, setFoodList] = useState([]);
  const [query, setQuery] = useState("");

  const { fetchCsvData } = useFetch();

  const search = (data) => {
    return data.filter((item) => {
      if (item.food) {
        return item.food.toLowerCase().includes(query.toLowerCase());
      } else return false;
    });
  };

  // Get food List on load
  useEffect(() => {
    fetchCsvData("/data/nutrients.csv", setFoodList);
  }, []);

  return (
    <div className="mx-10 my-5">
      <h1 className="text-center text-2xl mb-2">Food Search</h1>
      <SearchBar onChange={(q) => setQuery(q)} />
      <FoodTable foodList={search(foodList)} mealType={mealType} />
    </div>
  );
}
```

AppRoutes.jsx

```jsx
import React from "react";
import { Route, Routes } from "react-router-dom";
import Dashboard from "../pages/Dashboard.jsx";
import AccountPage from "../pages/AccountPage.jsx";
import DailyNutrition from "../pages/DailyNutrition.jsx";
import FoodPage from "../pages/FoodPage.jsx";
import WorkoutPage from "../pages/WorkoutPage.js";
import Routines from "../pages/Routines.jsx";
import LoginPage from "../pages/LoginPage.jsx";
import RegisterPage from "../pages/RegisterPage.jsx";
import WorkoutVideos from "../pages/WorkoutVideos.jsx"; // Import the
import UserRoutines from "../pages/UserRoutines.jsx";

function AppRoutes() {
  return (
    <Routes>
      <Route path="/" element={<Dashboard />} />
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/account" element={<AccountPage />} />
      <Route path="/nutrition" element={<DailyNutrition />} />
      <Route path="/old-routines" element={<Routines />} />
      <Route path="/routines" element={<UserRoutines />}></Route>
      <Route path="/food" element={<FoodPage />} />
      <Route path="/workout-videos" element={<WorkoutVideos />} />
    </Routes>
  );
}

export default AppRoutes;
```

Implementation

Models.py

```python
# One to many with user
class Food(db.Model):
    __tablename__= 'foods'

    id:int = db.Column(db.Integer, primary_key=True)
    user_id:int = db.Column(db.Integer, db.ForeignKey('users.id'))
    name:str = db.Column(db.String, nullable=False)
    meal_type:str = db.Column(db.String, nullable=False, default='')
    calories:int = db.Column(db.Integer, nullable=False)
    protein:int = db.Column(db.Integer, nullable=False, default=0)
    carbs:int = db.Column(db.Integer, nullable=False, default=0)
    fats:int = db.Column(db.Integer, nullable=False, default=0)
    date = db.Column(db.DateTime, default=date.today)

    def __repr__(self):
        return f"<Food name {self.name}>"

    def to_json(self):
        return {
            "id": self.id,
            "userId": self.user_id,
            "name": self.name,
            "mealType": self.meal_type,
            "calories": self.calories,
            "protein": self.protein,
            "carbs": self.carbs,
            "fats": self.fats,
            "date": self.date
        }
```

Routes.py

```python
@bp.route("/routines/<int:rid>/exercises/<int:eid>", methods =["PATCH"])
def update_exercise(rid,eid):
    routine = Routine.query.filter(Routine.id == rid).first()
    exercise_update = db.session.get(Exercise, eid)

    if not routine:
        return jsonify({"message": "Rotine not found"}), 404
    if not exercise_update:
        return jsonify({"message" : "Exercise does not exist"}), 404

    data = request.json
    exercise_update.name = data.get("name", exercise_update.name)
    exercise_update.type = data.get("type", exercise_update.type)
    exercise_update.duration = data.get("duration", exercise_update.duration)
    exercise_update.video_url = data.get("videoUrl", exercise_update.video_url)
    exercise_update.calories_burned = data.get("caloriesBurned", exercise_update.caloric
    db.session.commit()
    json_values = exercise_update.to_json()
    return (json_values, 201)
```

Formulas.py

```python
# weight in kg, height in cm
def calculate_bmr(weight, height, age, is_male):
    if is_male:
        bmr = 88.362 + (13.397 *float(weight)) + (4.799 * float(height)) - (5.677 * flo
    else:
        bmr = 655 + (4.3 * float(weight)) + (4.7 * float(height)) - (4.7 * float(age))
    return bmr

def calculate_daily_calories(bmr, activity_level):
    if activity_level == 0:
        calories = bmr * 1.2
    elif activity_level == 1:
        calories = bmr * 1.375
    elif activity_level == 2:
        calories = bmr * 1.55
    elif activity_level == 3:
        calories = bmr * 1.725
    protein = calories/18
    carbohydates = calories/4 * .5
    return int(calories), int(protein), int(carbohydates)
```

Testing

Backend

Test-Driven Development (Pytest)

```python
def test_login(client):
    body = {"email":"test1@test1.com", "password":"password1"}
    response = client.post("/login", data=json.dumps(body), headers=headers)

    assert response.status_code == 200

    data:dict = response.get_json()

    assert data["id"] == 1
    assert data["name"] == "user1"
    assert data["email"] == "test1@test1.com"
    assert not "password" in data
    assert data["birthdate"] == str(date.today())
    assert data["weight"] == 11.1
    assert data["weightGoal"] == 111.1
    assert data["height"]== 11.1
    assert data["isMale"] == True
```

Frontend

Test-Driven Development (jest)
Jest framework

```javascript
import React from "react";
import { render, screen, fireEvent } from "@testing-library/react";
import '@testing-library/jest-dom'; // For extended matchers
import FoodTable from "./FoodTable";
import * as controller from "../_mocks_/Controller";
    any
jest.mock("../Controller.jsx");

describe("FoodTable Component", () => {
    const foodList = [
        { food: "Apple", calories: 95, protein: 0.5, fat: 0.3, carbs: 25, category: "Fruit" },
        { food: "Chicken", calories: 165, protein: 31, fat: 3.6, carbs: 0, category: "Meat" },
    ];

    beforeEach(() => {
        jest.clearAllMocks(); // Reset mocks before each test
    });

    test("renders table with food items", () => {
        render(<FoodTable foodList={foodList} mealType="lunch" />);

        expect(screen.getByText("Apple")).toBeInTheDocument();
        expect(screen.getByText("Chicken")).toBeInTheDocument();
        expect(screen.getByText("Fruit")).toBeInTheDocument();
        expect(screen.getByText("Meat")).toBeInTheDocument();
    });

    test("calls addFoodToUser on button click", async () => {
        const mockLogin = controller.login.mockResolvedValue({ id: 1 });
```

Conclusion

- It was a fun experience
- We should have choosen a smaller project
- We overestimated our abilities