FITBITE

By: Group 21

01

Overview

Problem

- Struggle with tracking your meals?
- Not knowing what you’re eating?
- Need help with fitness goals?

We have the solution...

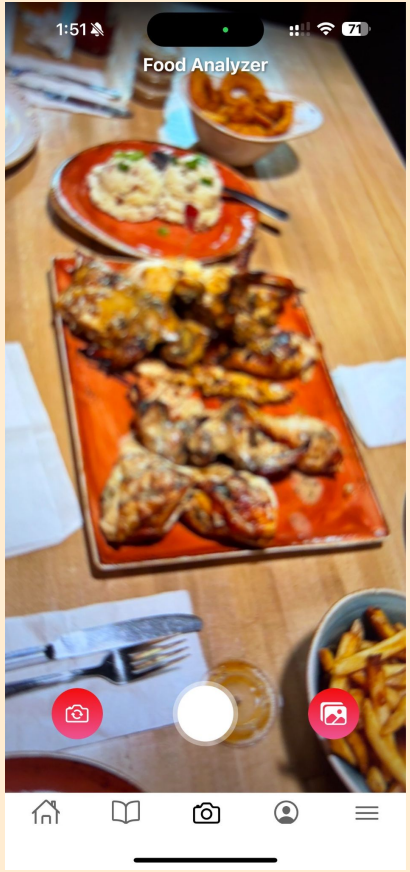
Introducing FitBite!

- A calorie tracker app that integrates AI (Google Gemini)

With just a simple click of a picture

Gives accurate macronutrient data:

- Calories
- Protein
- Fats
- Carbs
- Sugar



Our targeted users

- Fitness enthusiasts
- Wanting to gain/lose/maintain weight
- Anyone who wants to optimize their nutrition

Features

- Main Feature: Integrated camera
- Being able to add or delete meal
- Login System
- Get recommendations (Bulk or Cut)
- Get workout plans based on your goals
- Set goals for yourself and track your progress

Functional & Non-Functional Requirements

Requirement ID	Requirement Description	Design Reference	Test Case ID	Status
FREQ-01	Allow user to take a picture of their meal	Sequence Diagram	TC-01	Passed
FREQ-02	Analyze the provided image to output macronutrient information	Sequence Diagram	TC-02	Passed
FREQ-03	Allow users to track their protein and calorie intake	Sequence Diagram	TC-03	Passed
FREQ-04	Have a login system so user's information can be saved	Sequence Diagram	TC-04	Passed
FREQ-05	Allow users to input their weight, height, body fat percentage, and any other metrics to generate recommended goals	Class Diagram	TC-05	Passed

Requirements ID	Requirement Description	Design Reference	Test Case ID	Status
NFREQ-01	The system should provide an output within 3 seconds after the user uploads an image	N/A	TC-11	Failed
NFREQ-02	The system should be able to handle 100 users at once	N/A	TC-12	In Progress
NFREQ-03	The system should have a clean user interface	N/A	TC-13	Passed
NFREQ-04	The system should be able to work offline by caching data and syncing when it's back online	N/A	TC-14	Passed
NFREQ-05	The system should support multiple	N/A	TC-15	Not Implemented

02

System Architecture

MVC (Model-View-Controller)

Model:

The backend API, which handles meal analysis and data management, acts as the model by maintaining and processing data through Gemini and database operations.

Controller:

Frontend functions explicitly handle user interactions and logic, sending requests from the view to the model via APIs.

View:

The React native frontend components represent the view by rendering UI and visualizing the app data.

Design Patterns

Singleton:

Express server explicitly acts as a Singleton, ensuring that only one instance manages all incoming API requests consistently:

```
const app = express();
app.listen(PORT, () => {
    console.log(`Server running on ${PORT}`);
});
```

Explicitly one instance running throughout the application's lifecycle

Factory:

A clear use of the Factory Pattern can be seen in the Axios HTTP requests handling API calls explicitly in a modular fashion:

```
const response = await axios.post(
  `${endpoint}?key=${apiKey}`,
  payload
);
```

SOLID Principles

Single Responsibility:

Every module explicitly has a single, clearly defined responsibility:

```
app.post('/api/store-meal', (req, res) => {
    saveMeal(req.body);
    res.status(200).send('Stored');
});
```

Liskov Substitution

React components explicitly follow LSP as each component can substitute another:

```
const PieChart = ({ data }) => (
    <PieChartComponent data={data}
);
```

Interface Segregation:

The project explicitly segregates interfaces based on usage:

```
app.post('/api/get-summary', getSummary);
app.post('/api/analyze-food', analyzeFood);
```

Open-Closed:

Components explicitly follow this principle as they allow extension without modification:

```
const MealCarousel = ({ data, renderItem }) => (
    <Carousel data={data} renderItem={renderItem}
);
```

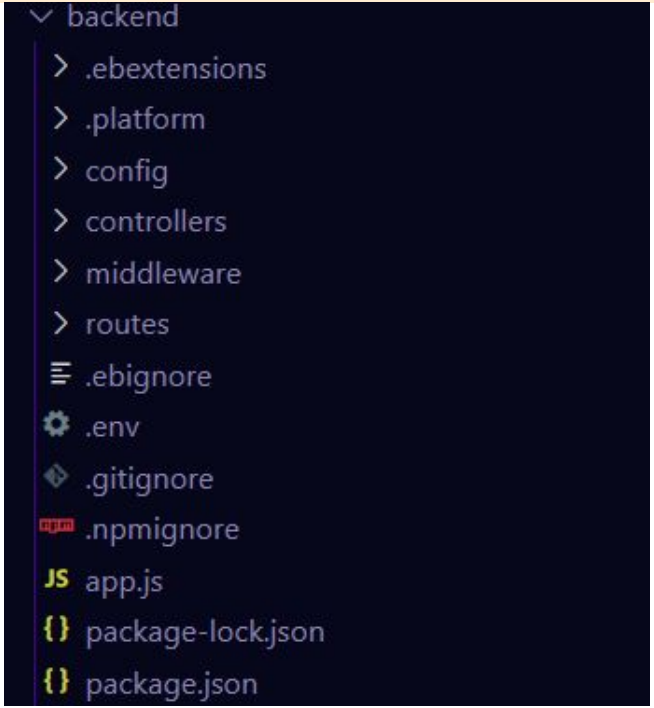
Dependency Inversion:

Explicitly implemented DIP by depending on abstractions rather than concrete implementations:

```
const axiosInstance = axios.create({
    baseURL: endpoint,
});
```

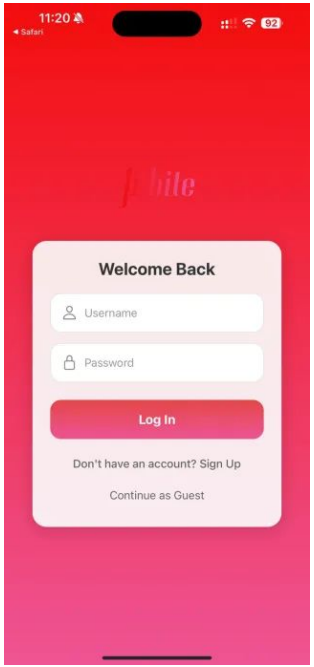
Backend

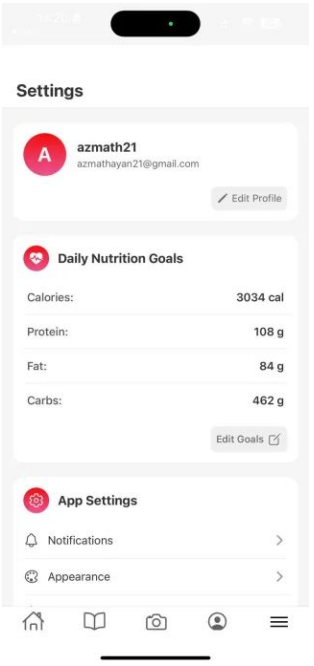
- Technologies Used: MySQL, AWS RDS, Node.js, Elastic beanstalk
- Deployment Using Elastic beanstalk
- AWS RDS: Cloud-based relational database service



Frontend

- Technologies Used: Figma, React Native, Photoshop





Sequence Diagram

03

Testing Strategies

Testing Strategies used

- White Box Testing
- Black Box Testing
- API Testing

White Box Testing

- Conducted Unit tests utilizing a JavaScript testing framework called Jest
- Testing key functions of our system

Example: BMI Calculation

```
test('should alert and set error when height or weight is missing or invalid', () => {
  calculateBMI(null, 70);// Height missing      You, 7 hours ago • Uncommitted changes
  expect(global.alert).toHaveBeenCalledWith("Your height or weight input box is empty");
  expect(global.setBmi).toHaveBeenCalledWith("Enter your data first");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(false);

  global.alert.mockClear();
  global.setBmi.mockClear();
  global.setShowBmiResults.mockClear();

  calculateBMI(170, null);//Weight missing
  expect(global.alert).toHaveBeenCalledWith("Your height or weight input box is empty");
  expect(global.setBmi).toHaveBeenCalledWith("Enter your data first");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(false);

  global.alert.mockClear();
  global.setBmi.mockClear();
  global.setShowBmiResults.mockClear();

  calculateBMI(-1, 70); //Negative input
  expect(global.alert).toHaveBeenCalledWith("Your height or weight input box is empty");
  expect(global.setBmi).toHaveBeenCalledWith("Enter your data first");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(false);
});
```

```
test('should alert and set error when non-numeric input is provided', () => {
  calculateBMI("abc", "def"); //Error invalid input
  expect(global.alert).toHaveBeenCalledWith("Please provide correct input");
  expect(global.setBmiResult).toHaveBeenCalledWith("");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(false);
});
```

```
// Verifies that the function calculates BMI correctly and sets "Normal weight" for valid inputs.
test('should calculate BMI and set normal weight for valid inputs', () => {
  calculateBMI(170, 70);// Calcualte nomally
  expect(global.setBmi).toHaveBeenCalledWith("24.22");
  expect(global.setBmiResult).toHaveBeenCalledWith("Normal weight");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(true);
});

// Verifies that the function sets "Underweight" when BMI is below 18.5.
test('should set Underweight when BMI is less than 18.5', () => {
  calculateBMI(170, 50); //Calculate underweight
  expect(global.setBmi).toHaveBeenCalledWith("17.30");
  expect(global.setBmiResult).toHaveBeenCalledWith("Underweight");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(true);
});

// Verifies that the function sets "Overweight" when BMI is between 25 and 30.
test('should set Overweight when BMI is between 25 and 30', () => {
  calculateBMI(170, 80); //Overweight
  expect(global.setBmi).toHaveBeenCalledWith("27.68");
  expect(global.setBmiResult).toHaveBeenCalledWith("Overweight");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(true);
});

// Verifies that the function sets "Obese" when BMI is 30 or above.
test('should set Obese when BMI is 30 or above', () => {
  calculateBMI(170, 100); //Obese
  expect(global.setBmi).toHaveBeenCalledWith("34.60");
  expect(global.setBmiResult).toHaveBeenCalledWith("Obese");
  expect(global.setShowBmiResults).toHaveBeenCalledWith(true);
});
```

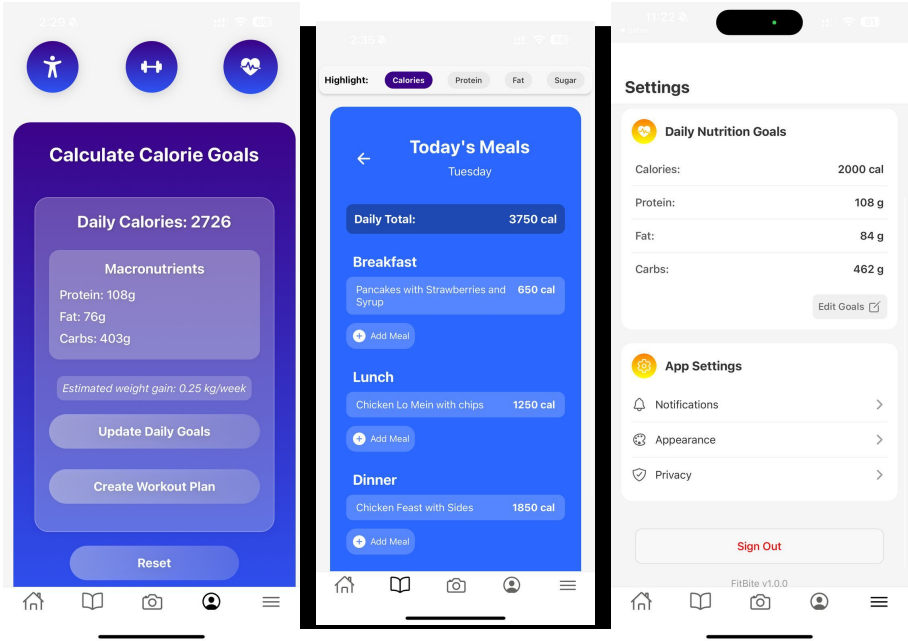
Output

```
Ran all test suites matching /bmiResultsTest/i.

Watch Usage: Press w to show more.
 PASS  __tests__/bmiResultsTest.test.js
  calculateBMI
    √ should alert and set error when height or weight is missing or invalid (7 ms)
    √ should calculate BMI and set normal weight for valid inputs
    √ should set Underweight when BMI is less than 18.5
    √ should set Overweight when BMI is between 25 and 30 (1 ms)
    √ should set Obese when BMI is 30 or above
    √ should alert and set error when non-numeric input is provided (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.671 s, estimated 1 s
Ran all test suites matching /bmiResultsTest/i.
```

Black Box Testing

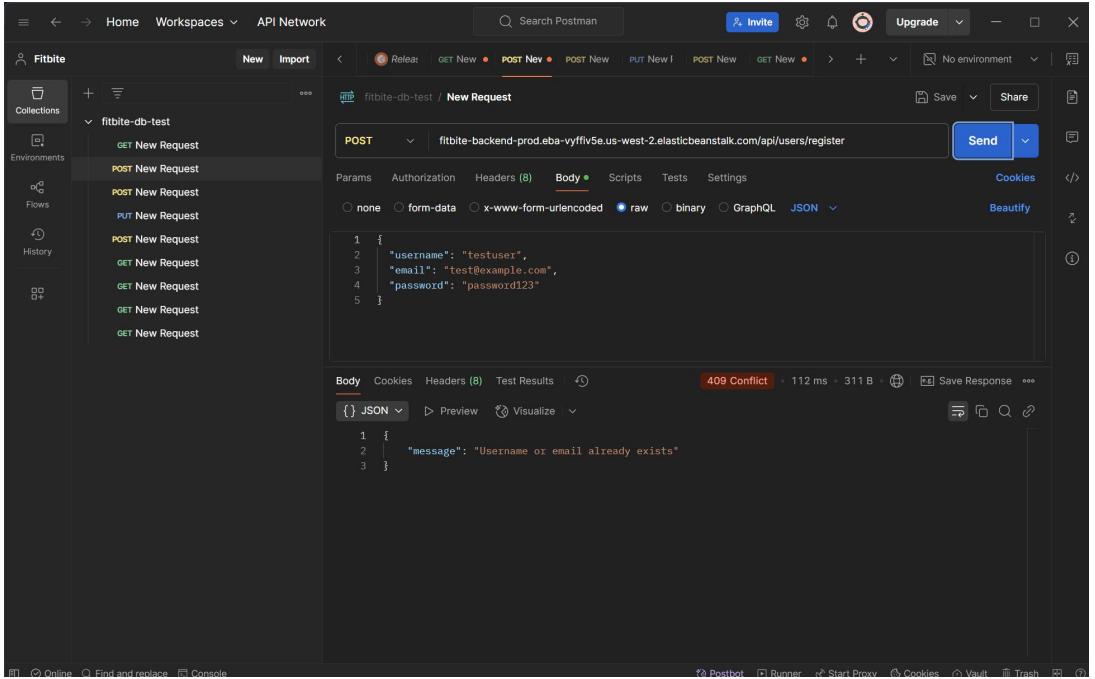
- Exploratory Testing
- Conducting System Tests
- Seeing if everything ran smoothly

- We played around with it to see



API Testing

- Postman
- Create and execute HTTP requests and check responses



05

Live Demo