EatWellthy

SDAA GROUP 31

Qixian, Rishika, Mahi, Xiaotao, Yichi, Nicole

Problem Statement

1. Many individuals find it challenging to manage their dietary needs due to the lack of personalization, difficulty in tracking nutritional intake as well as not knowing how to integrate their diet plans into their daily routines

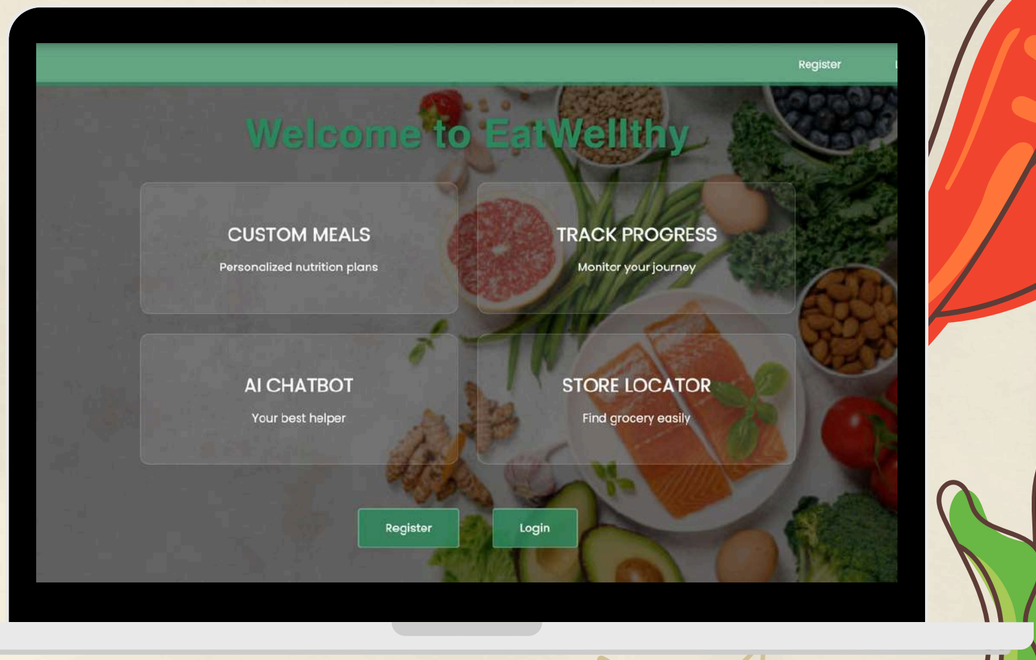
2. No existing apps provide an integration of all these functions in such a way that it is tailored to one's dietary preferences and needs

How can we encourage people to Eat Well and Healthy?

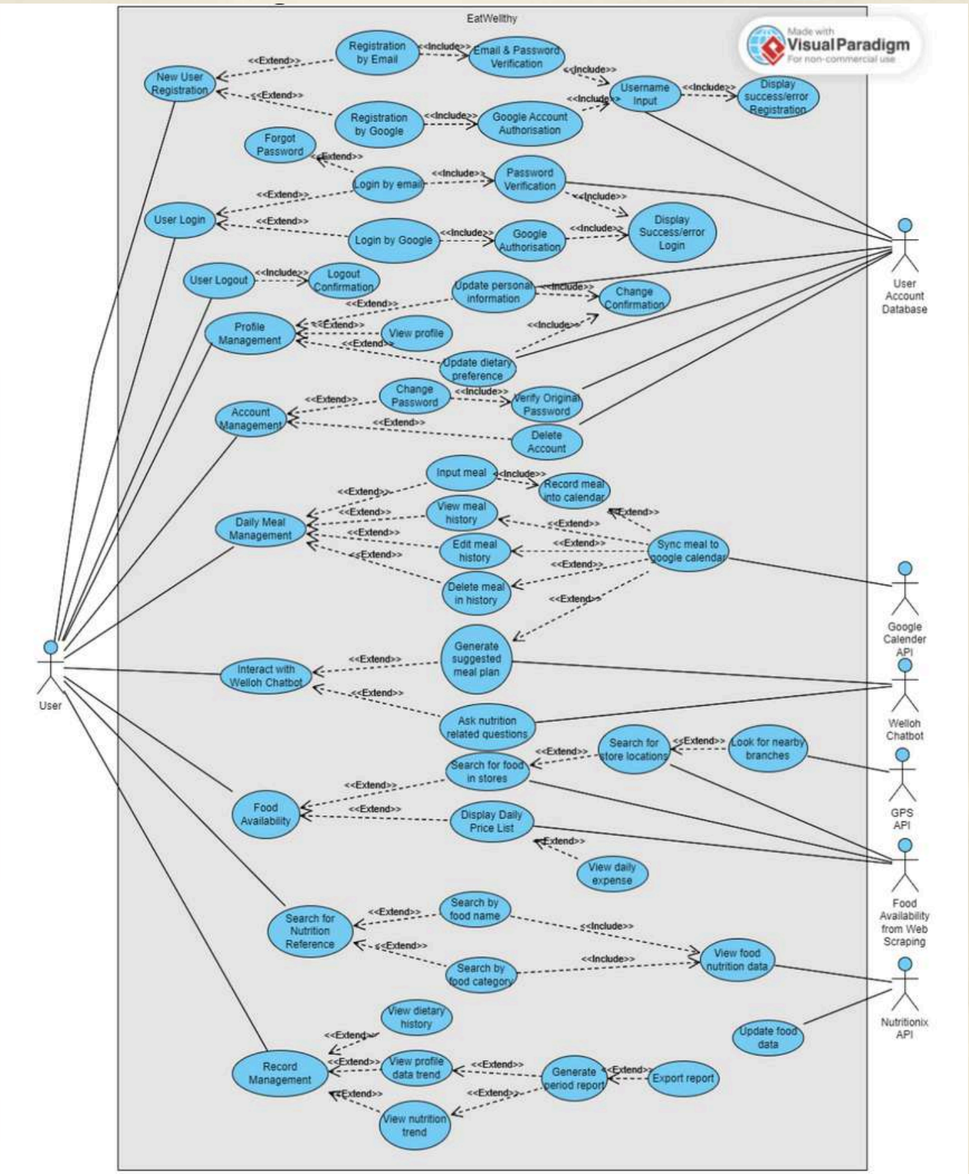
Solution - EatWellthy

EatWellthy is a user-friendly, multifaceted web application that aims to simplify the nutrition management process for users.

Our app offers a personalized experience to users, allowing them to effectively manage their diet and meet their nutritional goals in a more efficient manner.



Use Case Diagram



Feature List

1. Authentication
   - Register
   - Login
   - Login with Google
2. Dashboard
   - Progress tracking
   - Daily Nutritional Intake
   - Diet suggestions
3. Profile
   - Basic information such as name, age, dietary preferences, etc
   - Update password
   - Delete profile
4. Nutrition Tracker
   - Users can track nutritional content in their meals
5. Nutrition Analysis
   - Provides users with their daily nutritional consumption target
6. Grocery
   - Allows users to check prices of food from different supermarkets
7. Calendar/Google Calendar
   - Users can plan their meals for the day
   - Google users are able to import this into their Google Calendar
8. FAQs
   - Users can find answers to frequently asked questions
9. Store Locator
   - Users are able to locate nearby supermarkets
10. Welloh
    - Users can interact real-time with Welloh, our chatbot

External APIs

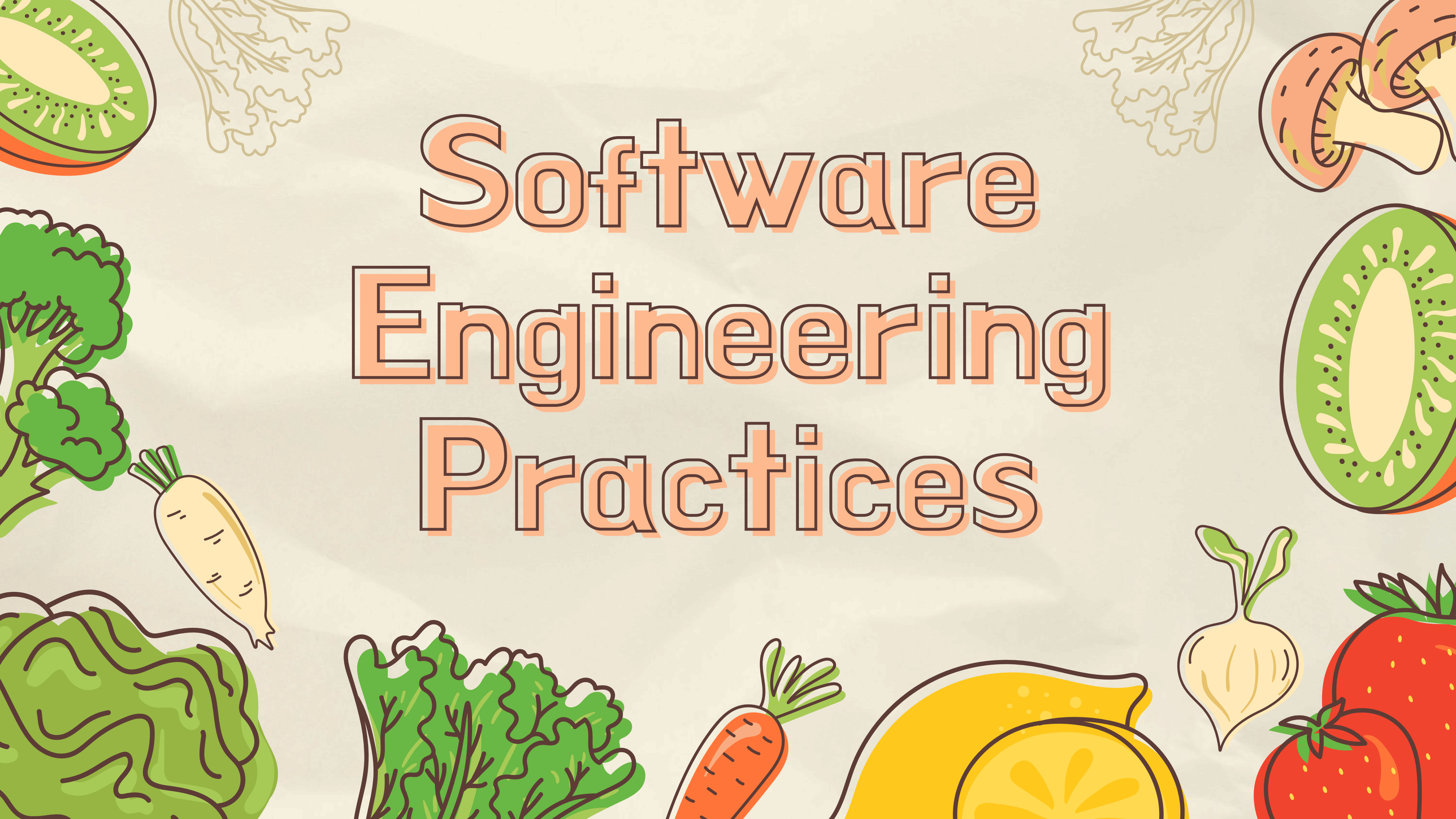
1	Google OAuth 2.0 API	Allows for user authentication with existing Google credentials
2	Google Maps API	To display supermarkets on a real-time live map
3	Nutritionix API	To provide users with nutritional data for different foods
4	OpenAI API	Allows for real-time interaction with Welloh
5	Brevo API	For sending email verification

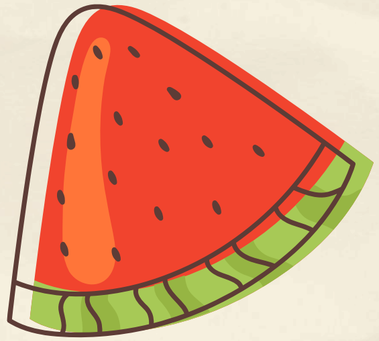
Live Demo!

Tech Stack

Front-End	Back-End
React.js	Node.js
Axios	Express.js
CSS for Styling	MongoDB
Redux (to manage state)	Mongoose

Software Engineering Practices





01 Good SWE Practices

1. Documentation
2. Good Code Practices
3. Reusability and Refactoring
4. SCRUM

Documentation - README

README

EatWellthy 🥗

A web application built with the MERN stack (MongoDB, Express.js, React.js, Node.js) to promote healthy eating habits! This README will guide you through setting up the project, managing changes, and making contributions.

🛠️ Getting Started

1. Clone the Repository

Clone the entire repository using Git or the GitHub Desktop app:

- **GitHub Desktop**: The app is beginner-friendly after a few tries and allows you to manage repositories easily.
- **Git Command Line**: If you prefer the terminal, use:

```
git clone <repository-url>
```

2. Fetch Latest Changes Regularly

Before making changes or reviewing code, always fetch the latest updates to stay in sync with the team:

- In GitHub Desktop: Click **Fetch Origin**.
- Or in the terminal:

```
git fetch origin
```

3. Open the Repository Locally

- Navigate to the cloned folder on your laptop, named `Github\EatWellthy`.
- Open PowerShell (Windows) or a terminal in VS Code to access the project directory.

4. Navigate to Project Folder

Open a terminal in the `EatWellthy` directory. Replace `WhereYouStoreTheProject` with your project's actual path:

```
cd "WhereYouStoreTheProject\EatWellthy"
```

🚀 Running the Application

To run the app, follow these steps:

1. **Navigate to the client directory**:

```
cd client
```

2. **Install Dependencies**:

```
npm install
```

3. **Start the Backend**:

```
cd ../backend
npm start
```

4. **Open in Browser**: The web app should open in your default browser.

🔄 Making & Managing Changes

After making updates, use the GitHub Desktop app to:

1. **Commit and Push**:
   - Make a commit with a description of your changes.
   - Push to the repository to update it for others.
2. **Undo Changes** (if needed):
   - Right-click on the modified file in GitHub Desktop and select "Discard Changes" to undo any uncommitted modifications.

🔗 Database Connection Setup

To connect components with the database, you'll likely edit these files:

- **Backend**:
  - `models`
  - `routes`
  - `server.js`
- **Client**:
  - `src/actions`
  - `src/components/reducers`

📦 Reinstalling Modules

If modules need reinstalling (usually in the `client` folder):

1. **Navigate to the client directory**:

```
cd client
```

2. **Remove Existing Modules**:
   - Windows:

```
Remove-Item -Recurse -Force .\node_modules
```

   - Mac & Linux:

```
rm -rf node_modules
```

3. **Install Dependencies with Legacy Peer Deps**:

```
npm install --legacy-peer-deps
```

   - For subsequent installs, you can simply use:

```
npm install
```

📝 Additional Notes

- Make sure to frequently fetch changes from the origin to prevent merge conflicts.
- Always test the application after making significant changes, especially those involving the backend or database.

Happy coding with **EatWellthy**! 💪

Detailed instruction on running the program for first time users and future developers

Documentation - Code Comments

```
//! @route   POST /users/forgot-password
//? @desc    Generate a new password and send via email
//? @access  Public

router.post(
  "/forgot-password",
  [check("email", "Please include a valid email").isEmail()],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email } = req.body;

    try {
      let user = await User.findOne({ email });

      if (!user) {
        return res.status(404).json({ errors: [{ msg: "User not found" }] });
      }

      //* Generate a temporary new password
      const temporaryPassword = generateTemporaryPassword();

      //* Hash the temporary password
      const salt = await bcrypt.genSalt(10);
      const hashedPassword = await bcrypt.hash(temporaryPassword, salt);

      //* Update the user's password
      user.password = hashedPassword;
      await user.save();

      //* Send the new password via email
      const emailSent = await sendForgotPasswordEmail(email, temporaryPassword

      if (!emailSent) {
        return res
```

```
const jwt = require("jsonwebtoken");

var jwtSecret = "mysecrettoken";

module.exports = function (req, res, next) {
  // todo: Get token from header
  const token = req.header("x-auth-token");

  // todo: Check if there is no token in the header
  if (!token) {
    return res.status(401).json({ msg: "No token, authorization denied" });
  }

  // todo: Verify token
  try {
    const decoded = jwt.verify(token, jwtSecret);
    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({ msg: "Token is not valid" });
  }
};
```

Curated comments to make the code more readable for humans and enhance collaboration

Consistent Code Style and Naming Conventions

Front-end

Back-end

Ensures readability and maintainability, Makes the code easier to understand and be extended

Reusability & Refactoring

Remove Duplication

Remove Redundancy

Consistent Naming Convention

Improve Reusability

Reusability & Refactoring

getProfile

axios

other http libraries

```
import axios from "axios";
import {
  SET_PROFILE,
  PROFILE_ERROR,
  LOADING_PROFILE,
  LOGOUT,
  CLEAR_PROFILE,
} from "./types";

// Get Profile
export const getProfile = () => async (dispatch) => {
  let config = null;

  try {
    dispatch({ type: LOADING_PROFILE });

    const token = localStorage.getItem("token");
    if (!token) {
      dispatch({
        type: PROFILE_ERROR,
        payload: "No token found. Please log in again.",
      });
      return null;
    }

    config = {
      headers: {
        "x-auth-token": token,
      },
    };

    console.log("Fetching profile...");
    const res = await axios.get("http://localhost:5050/api/profile/me", config);
    console.log("Profile response:", res.data);

    dispatch({
      type: SET_PROFILE,
      payload: res.data,
    });

    return res.data;
  } catch (err) {
    console.error("Error fetching profile:", err.response?.data || err.message);

    if (err.response?.status === 400 && config) {
      try {
        const defaultProfile = {
          age: 0,
          height: 0,
          weight: 0,
          targetWeight: 0,
          dailyBudget: 0,
```

```
onst DietSuggestions = ({ setDashboardLoading }) => {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const profileState = useSelector((state) => state.profile);
  const auth = useSelector((state) => state.auth);
  const [error, setError] = useState(null);
  const [showDialog, setShowDialog] = useState(false);
  const [showMealSelection, setShowMealSelection] = useState(false);
  const [isAdding, setIsAdding] = useState(false);
  const [showSuccessModal, setShowSuccessModal] = useState(false);
  const [mealDetails, setMealDetails] = useState(null);

  useEffect(() => {
    dispatch(getProfile());
  }, [dispatch]);

  useEffect(() => {
    if (profileState.profile && profileState.profile.userId) {
      dispatch(getAllEvents(profileState.profile.userId));
    }
  }, [profileState.profile, dispatch]);
```

```
useEffect(() => {
  dispatch(getProfile());
}, [dispatch]);
```

```
  const loadProfile = async () => {
    if (dataState.isProfileLoaded) return;

    try {
      const profileData = await dispatch(getProfile());

      const updatedProfile = {
        name: authState.user?.name || "",
        age: profileData.age || "",
        gender: profileData.gender || "",
        height: profileData.height || "",
        weight: profileData.weight || "",
        dailyBudget: profileData.dailyBudget || "",
        dietaryPreferences: profileData.dietaryPreferences || "",
        allergies: profileData.allergies?.join(", ") || ""
      };

      setProfile(updatedProfile);
      setDataState(prev => ({
        ...prev,
        isProfileLoaded: true,
      }));
    } catch (error) {
```

Reusability & Refactoring

```jsx
// In your MyCalendar.jsx, modify just the useEffect:
useEffect(() => {
  if (auth?.user?._id) {
    // Add this optional chaining
    getAllEvents(auth.user._id);
  }
}, [getAllEvents, open, auth?.user?._id]); // Add optional chaining here too

const openEventClick = async (event) => {
  // console.log("EVENT : ", event.id);
  await getEvent(event.id);
  setOpen(true);
};

// Add at the start of your MyCalendar component, right after const declarations:
if (!auth.user) {
  return (
    <div className="main-calendar">
      <div style={{ textAlign: "center", padding: "20px" }}>
        Loading calendar...
      </div>
    </div>
  );
}
```

Calendar call

getEvents()

```jsx
      setMealDetails(createEventDetails(mealType));
      setShowSuccessModal(true);
    }

    setShowDialog(false);
    dispatch(getAllEvents(auth.user._id));
  } catch (error) {
    console.error("Failed to add to calendar:", error);
    alert("Failed to add to calendar.");
```

getEvents()

```jsx
import React from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";

const Alert = ({ alerts }) =>
  alerts !== null &&
  alerts.length > 0 &&
  alerts.map((alert) => (
    <div key={alert.id} className={`alert alert-${alert.alertType}`}>
      {alert.msg}
    </div>
  ));

Alert.propTypes = {
  alerts: PropTypes.array.isRequired,
};

const mapStateToProps = (state) => ({
  alerts: state.alert,
});

export default connect(mapStateToProps)(Alert);
```

Layout Alert

```jsx
return (
  <div className="login-form">
    <h1 className="heading">Sign In</h1>
    <p className="lead">
      <i className="fas fa-user"></i> Sign Into Your Account
    </p>
    <Alert />
    <form className="form" onSubmit={onSubmit}>
      <div className="form-group">
        <input
          type="email"
          placeholder="Email Address"
```

Auth

```jsx
return (
  <div className="register-form">
    <h1 className="heading">Sign Up</h1>
    <p className="lead">
      <i className="fas fa-user"></i> Create Your Account
    </p>
    <Alert />
    <br />
    <form className="form" onSubmit={(e) => onSubmit(e)}>
      <div className="form-group">
        <input
          type="text"
```

Register

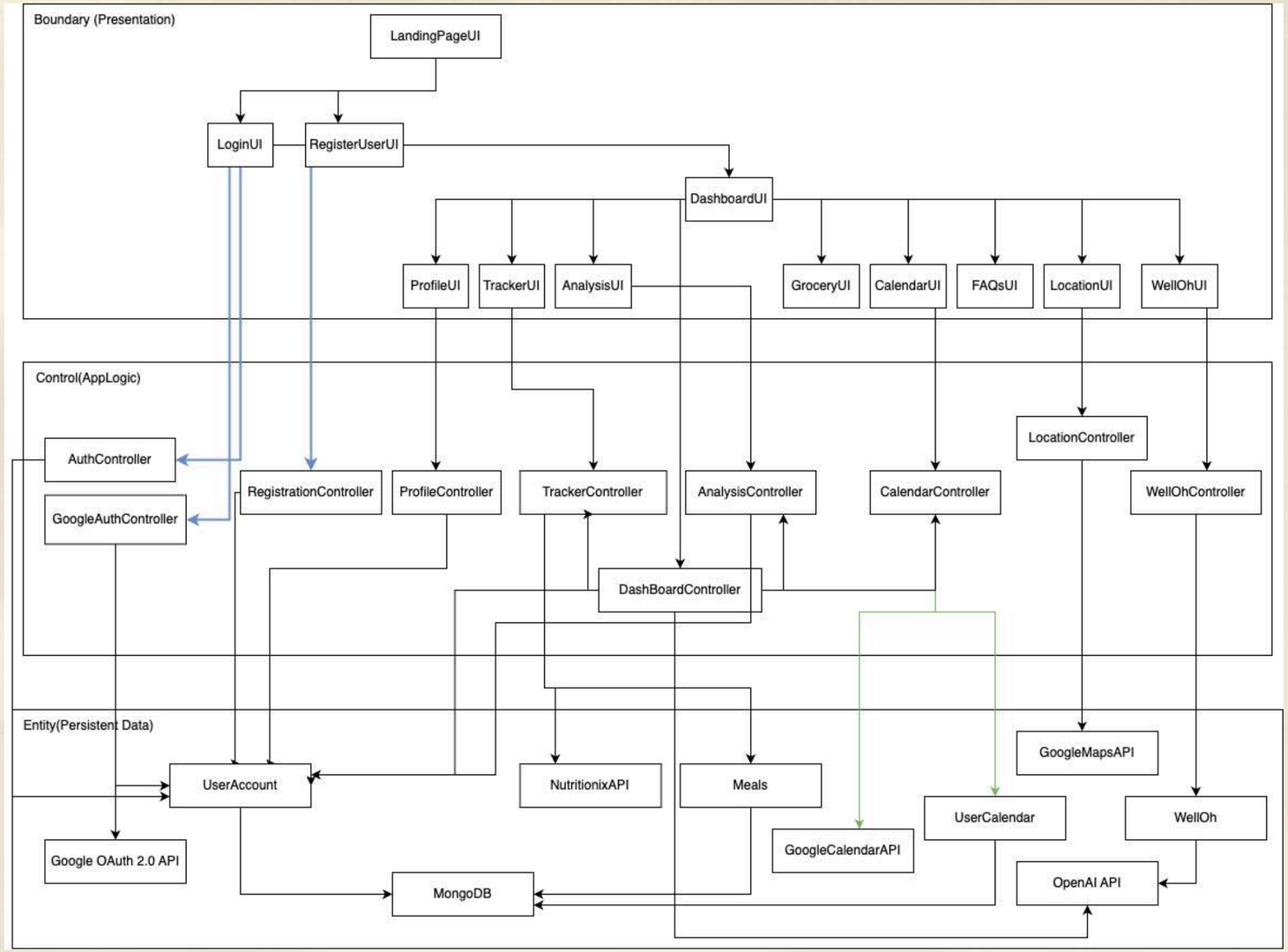
SCRUM


SCRUM MASTER
24 HOURS
REVEW
IMPLEMENTATIONn
RETROSPECT
2-4 WEEKS
PRODUCT OWNER
USER STORIES
SPRINT BACKLOG
PLANNING
DEFINITION OF DONE

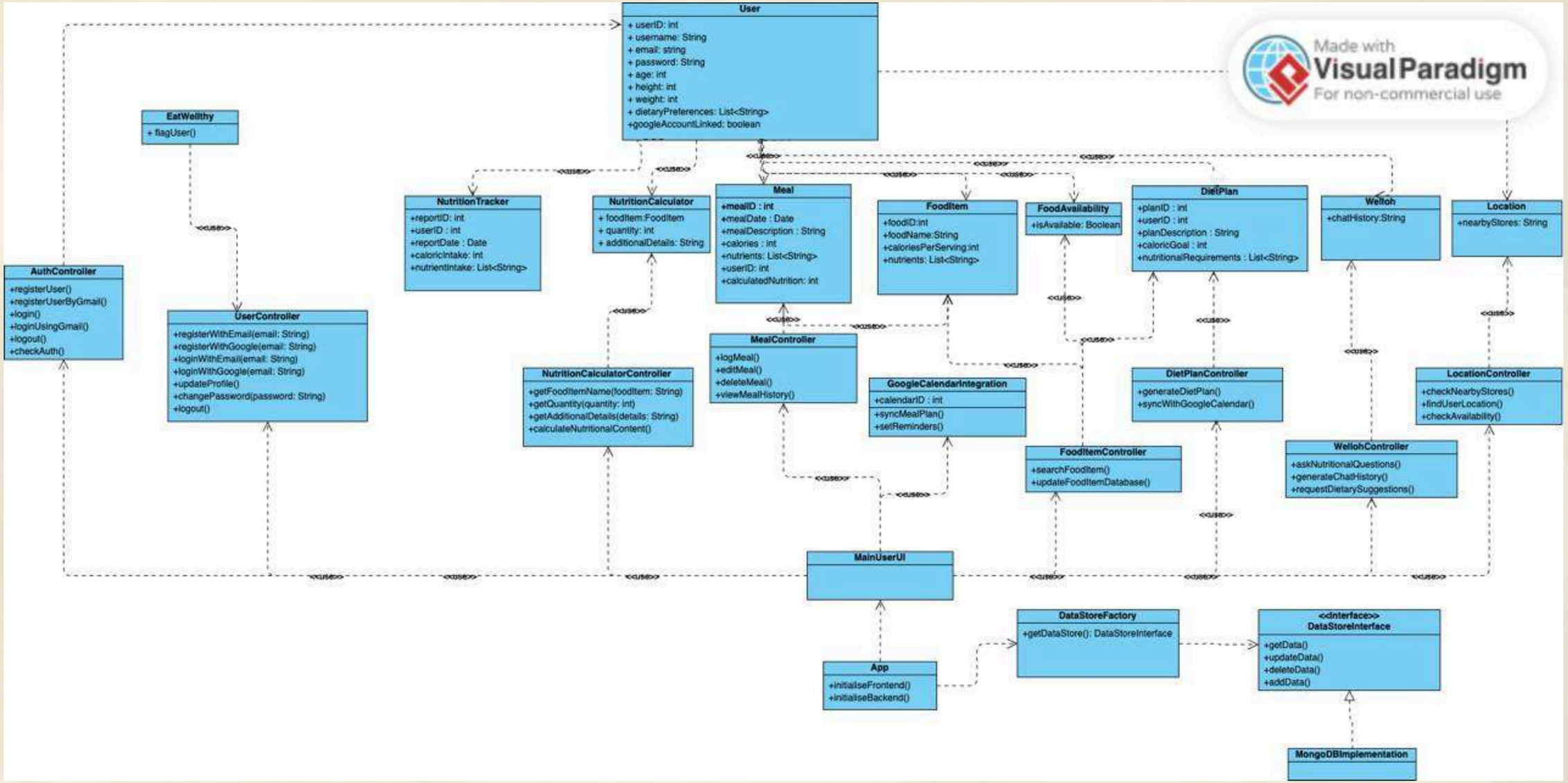
02 System Design

1. Architecture Diagram
2. Class Diagram
3. Overview Diagram

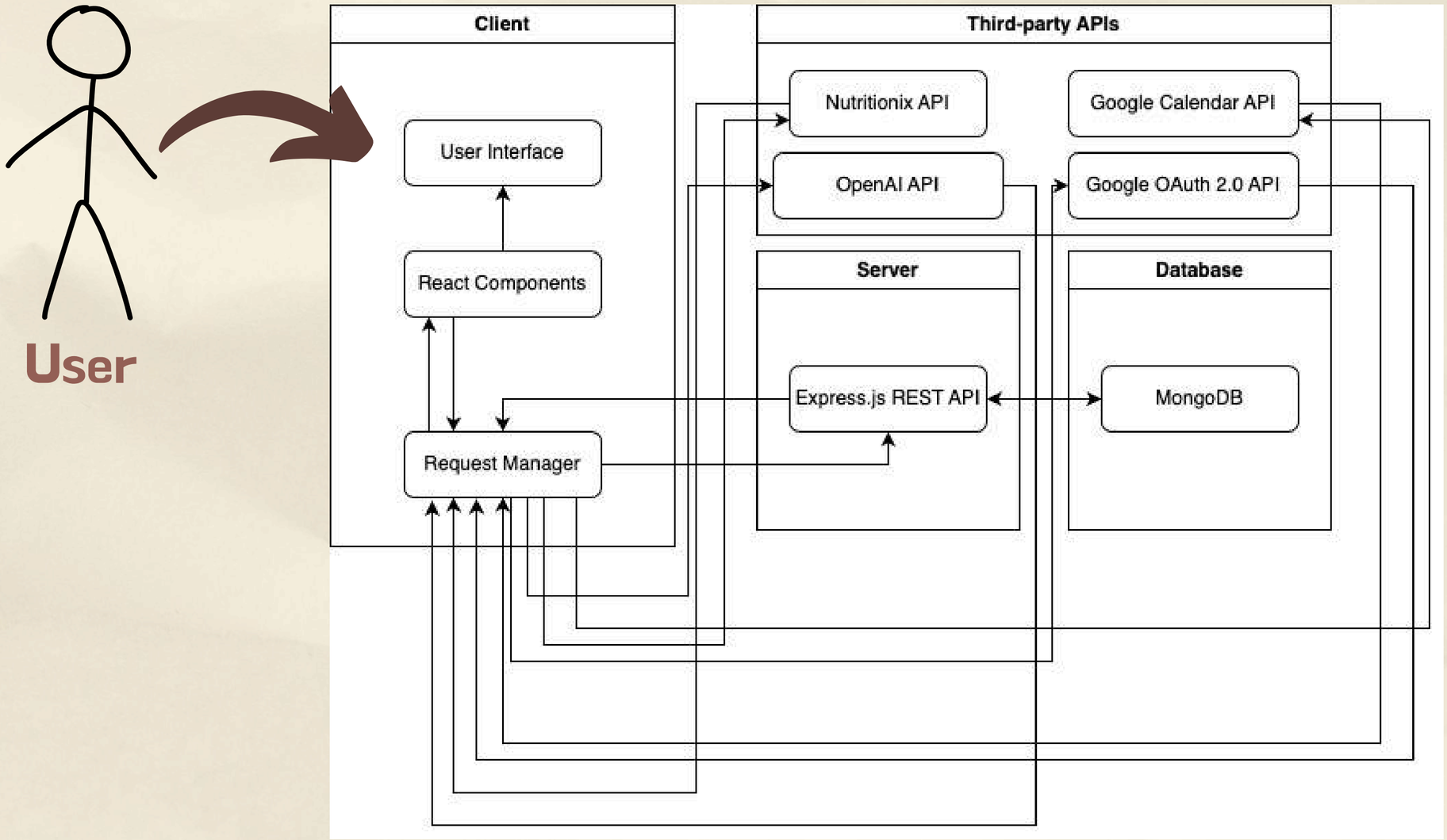
Architecture Diagram



Class Diagram



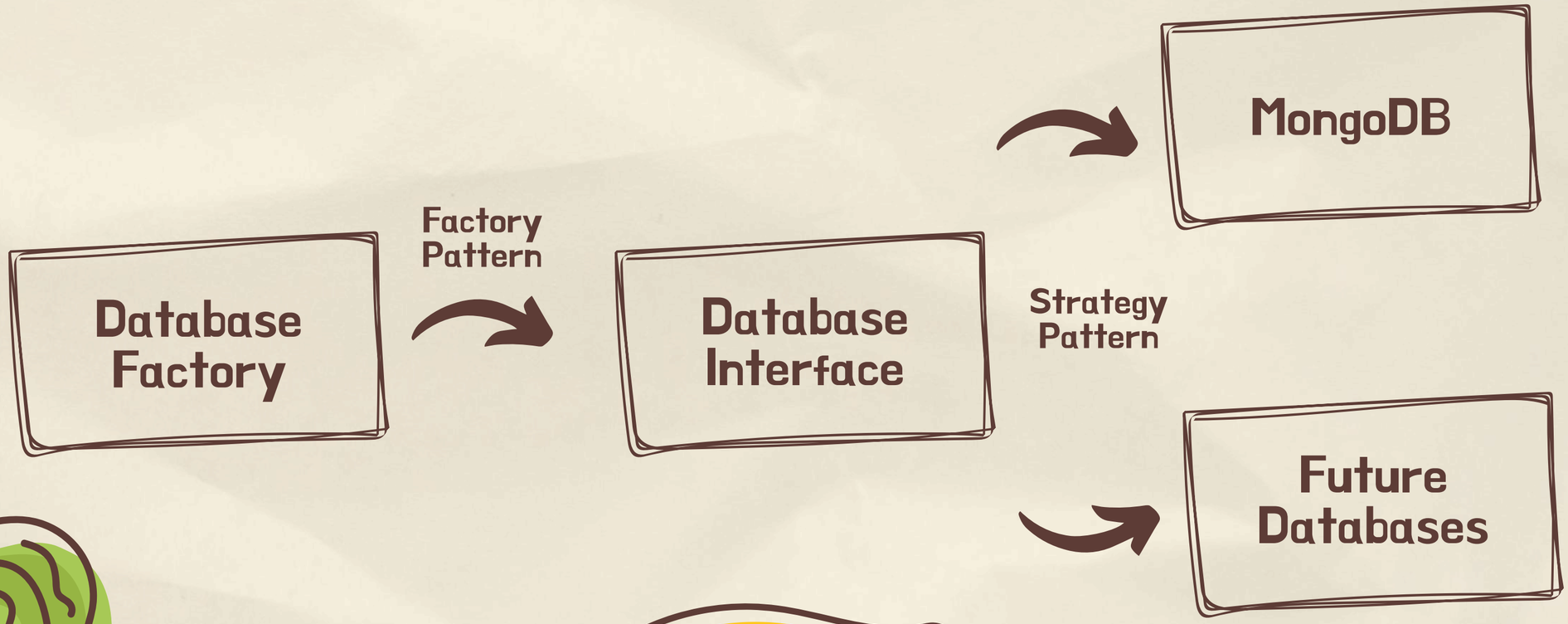
Overview Diagram



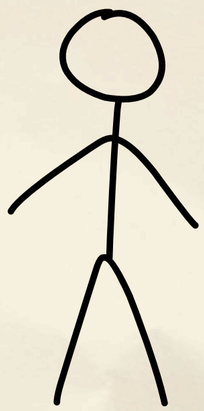
03 Design Patterns

1. Strategy & Factory Pattern
2. Facade Pattern
3. SOLID Principles

Strategy & Factory Pattern



Facade Pattern



User

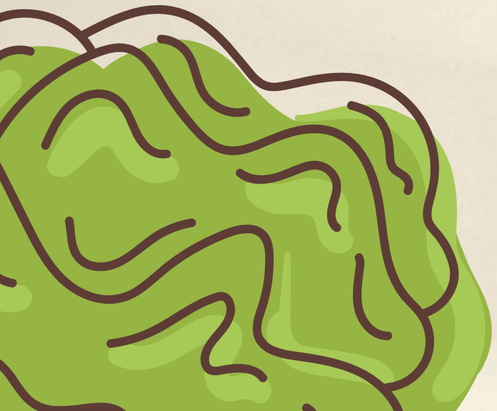
Front-end

Facade Pattern

API Endpoint

Back-end

Future Extensibilities



Traceability in Project Deliverables

Update Password

Use-Case Description

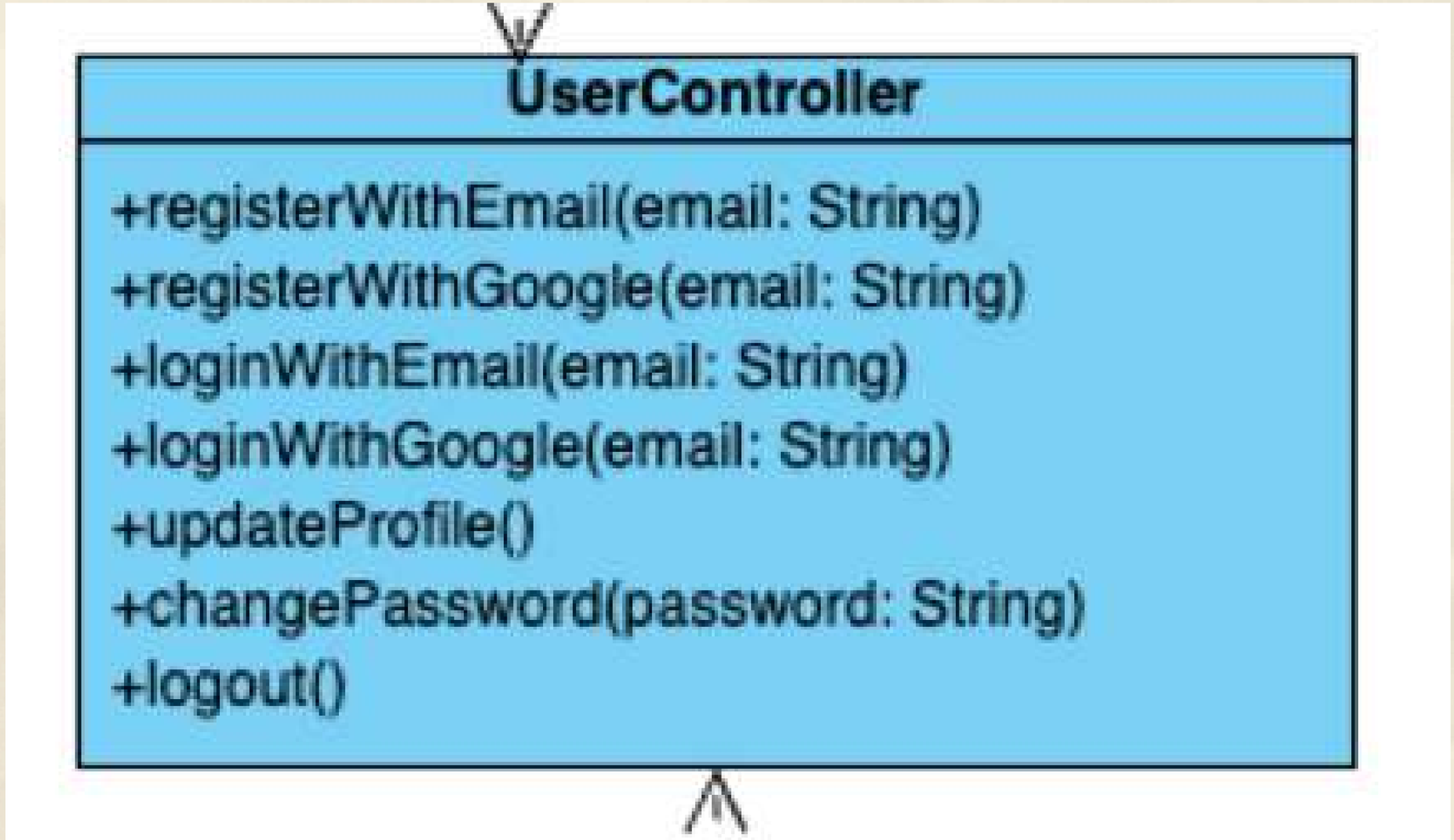
U0106 Change Account Password

Use Case ID:	U0106		
Use Case Name:	Change Account Password		
Created By:	Mahi Pandey	Last Updated By:	Mahi Pandey
Date Created:	02/09/2024	Date Last Updated:	02/09/2024

Actor:	Registered User
Description:	This use case allows registered users to change their account password while logged into the system.
Preconditions:	59. The user must be logged into their account. 60. The user must know their current password.
Postconditions:	61. The user's password is successfully changed. 62. The system logs the user out and prompts them to log in again with the new password.
Priority:	Medium
Frequency of Use:	Infrequent, as needed
Flow of Events:	63. The user navigates to the "Account Settings" page. 64. The system presents the option to change the password. 65. The user enters their current password and the new password, then confirms the new password. 66. The system validates the current password and checks the new password against security requirements. 67. If valid, the system updates the password and logs the user out. 68. The user is prompted to log in with new password.
Alternative Flows:	69. Incorrect Current Password • If the current password is incorrect, the system informs the user and allows re-entry. 70. Password Strength Failure • If the new password does not meet security requirements, the system prompts the user to enter a stronger password. 71. New Password Equivalent to Current Password • If the new password is the same as the current password, the system prompts the user to enter a different password.
Exceptions:	If the system is unable to update the password due to technical issues, an error message is displayed, and the user is asked to try again later.
Includes:	72. Verify Original Password
Special Requirements:	The system must ensure secure handling and storage of passwords.
Assumptions:	The user has access to their current password.
Notes and Issues:	Provide feedback on the strength of the password.

Update Password

Class Diagram



Update Password

Code Snippet

Backend

```
// @route   PUT /users/updatepassword
// @desc    Update password
// @access  Private
router.put(
  "/updatepassword",
  auth,
  [
    check(
      "password",
      "Please enter a password with 6 or more characters"
    ).isLength({ min: 6 }),
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      console.log("Validation failed:", errors.array());
      return res.status(400).json({ errors: errors.array() });
    }

    const { password } = req.body;

    try {
      const userId = req.user.id;
      console.log("User ID from token:", userId);

      let user = await User.findById(userId);

      if (!user) {
        console.log("User not found");
        return res.status(404).json({ msg: "User not found" });
      }

      console.log("User found:", user.email);

      const salt = await bcrypt.genSalt(10);
      const hashedPassword = await bcrypt.hash(password, salt);
      console.log("Hashed password:", hashedPassword);

      user.password = hashedPassword;

      await user.save();
      console.log("Password updated successfully");

      res.json({ msg: "Password updated successfully" });
    } catch (err) {
      console.log(err);
      console.error("Error while updating password:", err.message);
      res.status(500).send("Server error");
    }
  }
);
```

Frontend

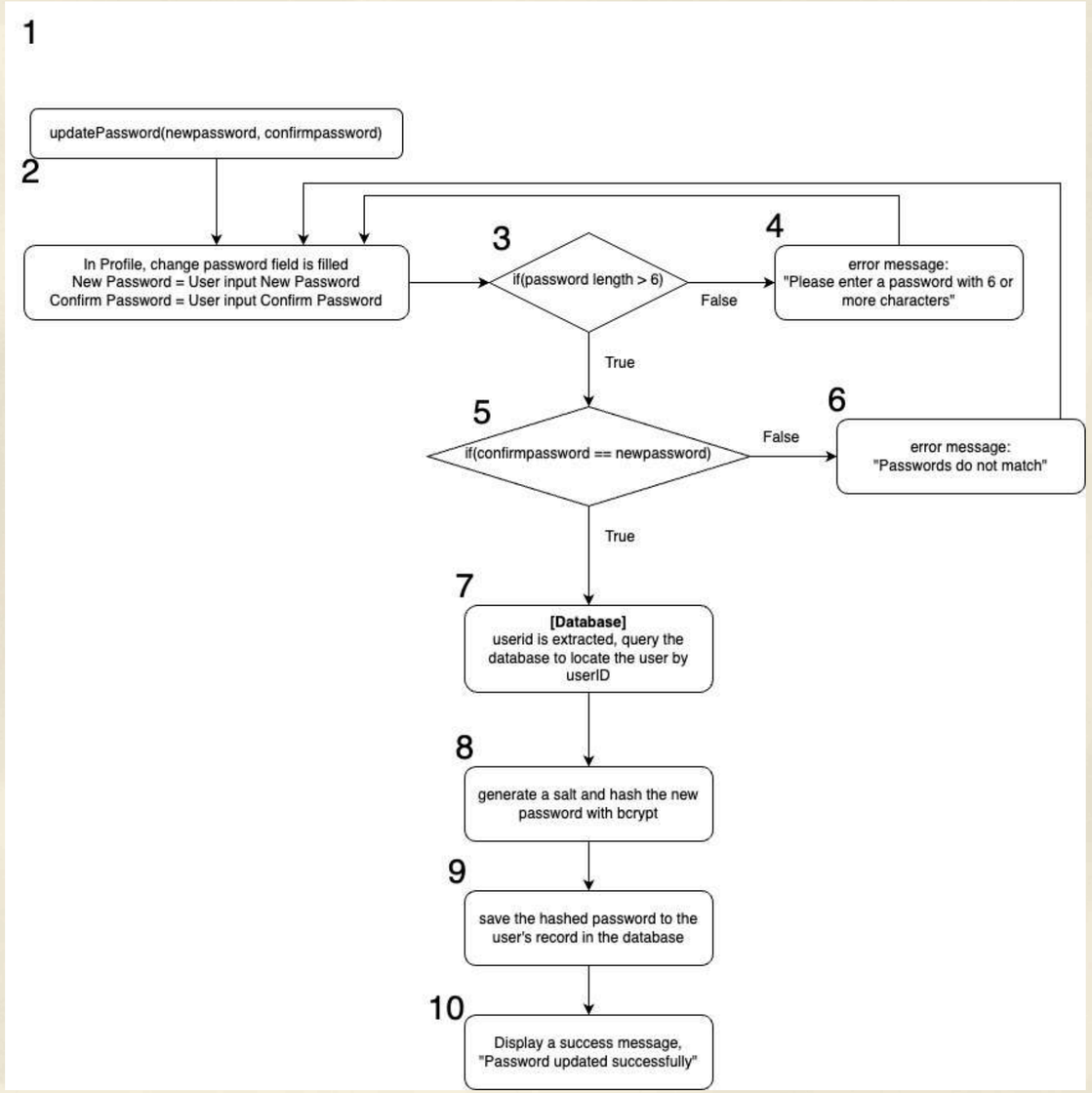
```
    const success = await dispatch(updateProfile(profileData));
    if (success) {
      showMessage("Profile updated successfully");
    }
  };

  const onSubmitPassword = async (e) => {
    e.preventDefault();
    if (newPassword.length < 6) {
      showMessage("Password must be at least 6 characters", "error");
      return;
    }
    if (newPassword !== confirmPassword) {
      showMessage("Passwords do not match", "error");
      return;
    }

    const success = await dispatch(changePassword(newPassword));
    if (success) {
      setFormData((prev) => ({
        ...prev,
        newPassword: "",
        confirmPassword: "",
      }));
      showMessage("Password updated successfully");
    }
  };
```

Update Password

Control Flow Diagram



Update Password

Testcases

C. Test Cases and Results

UpdatePasword(newpassword,confirmpassword)

No.	Test Input	Expected Output	Actual Output	Pass ?
1	**(Valid)** New Password: “Password12” **(Valid)** Confirm Password: “Password12”	Success message displayed, "Password updated successfully”	Success message displayed, "Password updated successfully”	Yes
2	**(Invalid)** New Password: “” **(Valid)** Confirm Password: “Password12”	Error message, “Password must be at least 6 characters”	Error message, “Password must be at least 6 characters”	Yes
3	**(Invalid)** New Password: “Password12” **(Valid)** Confirm Password: “”	Error message, “Passwords do not match”	Error message, “Passwords do not match”	Yes
4	**(Invalid)** New Password: “Pass” **(Valid)** Confirm Password: “Password12”	Error message, “Passwords do not match”	Error message, “Passwords do not match”	Yes
5	**(Invalid)** New Password: “Password12” **(Valid)** Confirm Password: “Pass”	Error message, “Passwords do not match”	Error message, “Passwords do not match”	Yes

B. Basis Path Testing

Cyclomatic Complexity = | decision points | + 1 = 2 + 1 = 3

Basis Paths

1. Baseline Path: 1,2,3,5,7,8,9,10
2. Basis Path 2: 1,2,3,4
3. Basis Path 3: 1,2,3,5,6

Future Plans

Future plans and implementations for EatWellthy

Web-Scrapping

Introduce a WebScraper to find the lowest price ingredients across all grocery stores

Adding a Physical Fitness Component

Collaborate with fitness apps or trainers to provide holistic plans that combine diet and exercise

Community Challenges & Achievements

Add communities challenges such as “Meatless Mondays”

Predictive Health Insights

Presentations are tools that can be used as demonstrations, lectures, speeches, and more.

Thank you!

Let's Eat Well and Healthy
with EatWellthy!