Full Stack Development with MERN

API Development and Integration Report

Date	
Team ID	PNT2022TMIDxxxxxx
Project Name	Project - xxx
Maximum Marks	

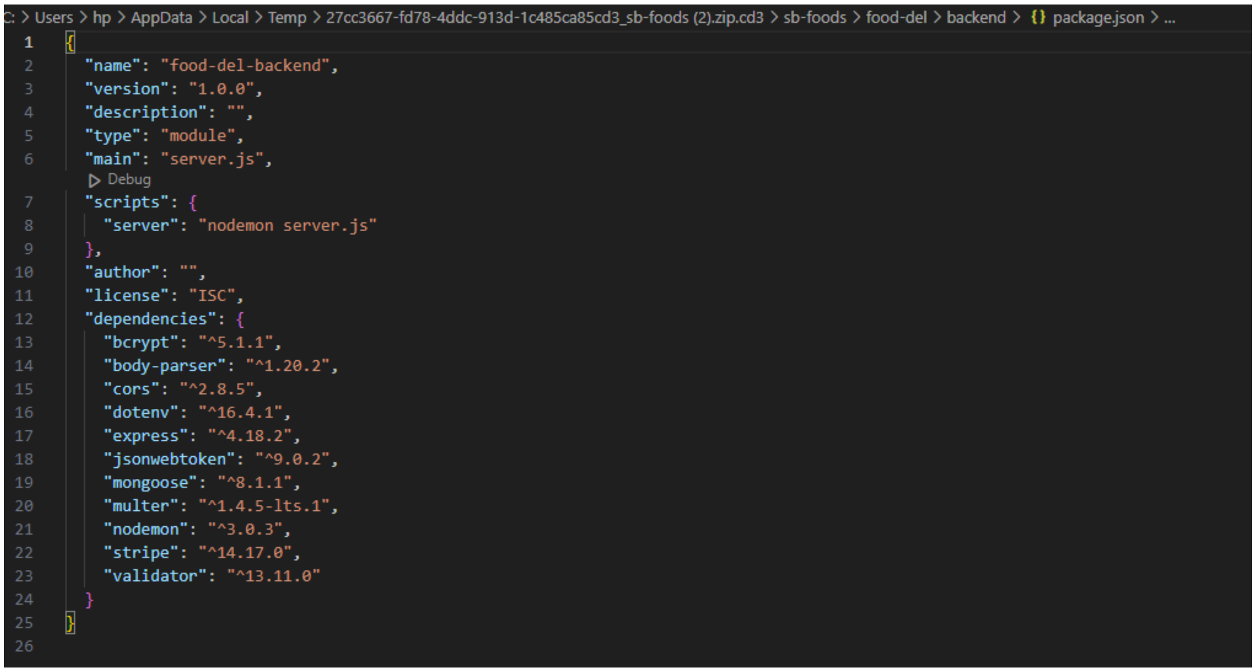
Project Title: SB Foods – Order on the go
Date: 20-07-2024
Prepared by: SB Foods – Food Ordering App

Objective
The primary objective of the SB Foods - Order on the Go project is to develop a user-friendly mobile application and/or web platform that allows customers to efficiently browse, select, and order food items from SB Foods.

Technologies Used

- **Backend Framework:** Express.js (Node.js) , Django (Python)
- **Database:** MongoDB, My SQL
- **Authentication:** JWT (JSON Web Tokens) , OAuth 2.0, Passport.js

Project Structure



```
C: > Users > hp > AppData > Local > Temp > 27cc3667-fd78-4ddc-913d-1c485ca85cd3_sb-foods (2).zip.cd3 > sb-foods > food-del > backend > {} package.json > ...
{
  "name": "food-del-backend",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "server.js",
  ▷ Debug
  "scripts": {
    "server": "nodemon server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.4.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.1.1",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.0.3",
    "stripe": "^14.17.0",
    "validator": "^13.11.0"
  }
}
```

Key Directories and Files

1. **src/controllers/**: A directory to hold controller-like functions or classes that handle business logic and interact with services.

   UserController.js

   OrderController.js

2. **src/models/**: A directory to hold model definitions or TypeScript interfaces that represent the data structures used in your application.

   UserModel.js (or UserModel.ts)

   OrderModel.js (or OrderModel.ts)

3. **src/routes/**: A directory to hold route configuration files.

   index.js: The main routing configuration file.

4. **src/middlewares/**: A directory to hold middleware-like functions or higher-order components.

   authMiddleware.js

   loggingMiddleware.js

.

5. **src/config/**: A directory for configuration files related to your application.

   apiConfig.js: Configuration for API endpoints.

   appConfig.js: General application settings.

API Endpoints:

```
const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

export const apiEndpoints = {

  users: `${API_URL}/users`,

  orders: `${API_URL}/orders`,

  products: `${API_URL}/products`,

  login: `${API_URL}/auth/login`,

  register: `${API_URL}/auth/register`,

};
```

User Authentication:

- **POST /api/user/register**

  The way we define the API endpoint for user registration in our configuration
  Code :

  ```
  // src/config/apiConfig.js

  const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

  export const apiEndpoints = {
    registerUser: `${API_URL}/api/user/register`,
    // Other endpoints...
  };
  ```

We use this endpoint in a service file to handle user registration:

Code :
```
// src/services/auth.js

import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';

export const registerUser = async (userData) => {
  try {
    const response = await axios.post(apiEndpoints.registerUser, userData);
    return response.data;
  } catch (error) {
    console.error('Error registering user:', error);
    throw error;
  }
};
```

This setup will allow you to use the registerUser function to send a POST request to /api/user/register with the provided userData.

- **POST /api/user/login**

  We define the API endpoint for user login in our configuration

```
// src/config/apiConfig.js

const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

export const apiEndpoints = {
  registerUser: `${API_URL}/api/user/register`,
  loginUser: `${API_URL}/api/user/login`,
  // Other endpoints...
};
```

  We use this endpoint in a service file to handle user login

```
// src/services/auth.js

import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';

export const loginUser = async (credentials) => {
  try {
    const response = await axios.post(apiEndpoints.loginUser, credentials);
    return response.data;
  } catch (error) {
    console.error('Error logging in:', error);
    throw error;
  }
};
```

This setup will allow us to use the loginUser function to send a POST request to /api/user/login with the provided credentials.

User Management

- **GET /api/user/**

  We define the API endpoint for retrieving user data in your configuration

```
// src/config/apiConfig.js

const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

export const apiEndpoints = {
  registerUser: `${API_URL}/api/user/register`,
  loginUser: `${API_URL}/api/user/login`,
  getUsers: `${API_URL}/api/user/`,
  // Other endpoints...
};
```

We use this endpoint in a service file to get user data:

```
// src/services/user.js

import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';

export const fetchUsers = async () => {
  try {
    const response = await axios.get(apiEndpoints.getUsers);
    return response.data;
  } catch (error) {
    console.error('Error fetching users:', error);
    throw error;
  }
};
```

- **PUT /api/user/**

We define the API endpoint for updating user data in your configuration. Typically, you would include a placeholder for the user ID in the endpoint.

```
// src/config/apiConfig.js


const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

export const apiEndpoints = {

  registerUser: `${API_URL}/api/user/register`,

  loginUser: `${API_URL}/api/user/login`,

  getUsers: `${API_URL}/api/user/`,

  updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically

  // Other endpoints...

};
```

We use this endpoint in a service file to update user data:

```
// src/services/user.js


import { apiEndpoints } from '../config/apiConfig';

import axios from 'axios';

export const updateUser = async (userId, userData) => {

  try {

    const response = await axios.put(`${apiEndpoints.updateUser}${userId}`, userData);

    return response.data;

  } catch (error) {

    console.error('Error updating user:', error);

    throw error;
  }
};
```

To update a user with ID 123, we would call the updateUser function like this:

```
updateUser('123', { name: 'New Name', email: 'new.email@example.com' });
```

This will send a PUT request to `/api/user/123` with the `userData` you want to update.

Workout Plans

- **GET /api/workoutplans**

  We define the API endpoint for retrieving workout plans in your configuration:

  ```
  // src/config/apiConfig.js

  const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

  export const apiEndpoints = {
    registerUser: `${API_URL}/api/user/register`,
    loginUser: `${API_URL}/api/user/login`,
    getUsers: `${API_URL}/api/user/`,
    updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically
    getWorkoutPlans: `${API_URL}/api/workoutplans`,
    // Other endpoints...
  };
  ```

  We use this endpoint in a service file to get workout plans:

  ```
  // src/services/workout.js

  import { apiEndpoints } from '../config/apiConfig';
  import axios from 'axios';
  export const fetchWorkoutPlans = async () => {
    try {
      const response = await axios.get(apiEndpoints.getWorkoutPlans);
      return response.data;
    } catch (error) {
      console.error('Error fetching workout plans:', error);
      throw error;
    }
  };
  ```

This setup will allow you to send a GET request to /api/workoutplans to retrieve the workout plans.

- **POST /api/workoutplans**

  We define the API endpoint for creating a new workout plan in your configuration:

  ```
  // src/config/apiConfig.js

  const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

  export const apiEndpoints = {
    registerUser: `${API_URL}/api/user/register`,
    loginUser: `${API_URL}/api/user/login`,
    getUsers: `${API_URL}/api/user/`,
    updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically
    getWorkoutPlans: `${API_URL}/api/workoutplans`,
    createWorkoutPlan: `${API_URL}/api/workoutplans`,
    // Other endpoints...
  };
  ```

We use this endpoint in a service file to create a new workout plan:

```
    // src/services/workout.js

  import { apiEndpoints } from '../config/apiConfig';

import axios from 'axios';

export const createWorkoutPlan = async (workoutPlanData) => {

  try {

    const response = await axios.post(apiEndpoints.createWorkoutPlan, workoutPlanData);

    return response.data;

  } catch (error) {

    console.error('Error creating workout plan:', error);

    throw error;

  }

};
```

To create a new workout plan, we would call the createWorkoutPlan function like this:

```
const newWorkoutPlan = {

  name: 'Morning Routine',

  exercises: [

    { name: 'Push-up', sets: 3, reps: 15 },

    { name: 'Squat', sets: 3, reps: 20 },

  ],

};

createWorkoutPlan(newWorkoutPlan).then(plan => {

  console.log('Created Workout Plan:', plan);

});
```

This setup will send a POST request to `/api/workoutplans` with the `workoutPlanData` to create a new workout plan.

Equipment

- **GET /api/equipment**

  We define the API endpoint for retrieving equipment in your configuration:

  ```
  // src/config/apiConfig.js

  const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';
  export const apiEndpoints = {
    registerUser: `${API_URL}/api/user/register`,
    loginUser: `${API_URL}/api/user/login`,
    getUsers: `${API_URL}/api/user/`,
    updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically
    getWorkoutPlans: `${API_URL}/api/workoutplans`,
    createWorkoutPlan: `${API_URL}/api/workoutplans
  getEquipment: `${API_URL}/api/equipment`,
   // Other endpoints...
  };
  ```

  We use this endpoint in a service file to get equipment data:

```
// src/services/equipment.js

import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';

export const fetchEquipment = async () => {
  try {
    const response = await axios.get(apiEndpoints.getEquipment);
    return response.data;
  } catch (error) {
    console.error('Error fetching equipment:', error);
    throw error;
  }
};
```

To fetch equipment, you would call the `fetchEquipment` function like this:

```
fetchEquipment().then(equipment => {
  console.log('Equipment:', equipment);
});
```

This setup will send a GET request to `/api/equipment` to retrieve the equipment data.

- **POST /api/equipment**

We define the API endpoint for creating new equipment in your configuration.

```
// src/config/apiConfig.js

const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

export const apiEndpoints = {
  registerUser: `${API_URL}/api/user/register`,
  loginUser: `${API_URL}/api/user/login`,
  getUsers: `${API_URL}/api/user/`,
  updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically
  getWorkoutPlans: `${API_URL}/api/workoutplans`,
  createWorkoutPlan: `${API_URL}/api/workoutplans`,
  getEquipment: `${API_URL}/api/equipment`,
  createEquipment: `${API_URL}/api/equipment`,
  // Other endpoints...
};
```

We use this endpoint in a service file to create new equipment:

```
// src/services/equipment.js
```

```
import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';

export const createEquipment = async (equipmentData) => {
  try {
    const response = await axios.post(apiEndpoints.createEquipment, equipmentData);
    return response.data;
  } catch (error) {
    console.error('Error creating equipment:', error);
    throw error;
  }
};
```

To create new equipment, you would call the `createEquipment` function like this:

```
const newEquipment = {
  name: 'Dumbbell',
  type: 'Weight',
  quantity: 10,
};

createEquipment(newEquipment).then(equipment => {
  console.log('Created Equipment:', equipment);
});
```

This setup will send a POST request to /api/equipment with the equipmentData to create new equipment.

Monthly Plans

- **GET /api/monthlyplans**
  We define the API endpoint for retrieving monthly plans in your configuration:

```
// src/config/apiConfig.js

 const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

 export const apiEndpoints = {

registerUser: `${API_URL}/api/user/register`,

loginUser: `${API_URL}/api/user/login`,

getUsers: `${API_URL}/api/user/`,

updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically
getWorkoutPlans: `${API_URL}/api/workoutplans`,
createWorkoutPlan: `${API_URL}/api/workoutplans`,
getEquipment: `${API_URL}/api/equipment`,
createEquipment: `${API_URL}/api/equipment`,
getMonthlyPlans: `${API_URL}/api/monthlyplans`,
// Other endpoints...
};
```

We use this endpoint in a service file to get monthly plans:

```
// src/services/monthlyPlans.js
import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';
export const fetchMonthlyPlans = async () => {
  try {
    const response = await axios.get(apiEndpoints.getMonthlyPlans);
    return response.data;
  } catch (error) {
    console.error('Error fetching monthly plans:', error);
    throw error;
  }
};
```

To fetch monthly plans, you would call the fetchMonthlyPlans function like this:

```
fetchMonthlyPlans().then(plans => {
  console.log('Monthly Plans:', plans);
});
```

This setup will send a GET request to `/api/monthlyplans` to retrieve the monthly plans data.

POST /api/monthlyplans

We define the API endpoint for creating a new monthly plan in your configuration:

```
// src/config/apiConfig.js

const API_URL = import.meta.env.VITE_API_URL || 'https://api.example.com';

export const apiEndpoints = {

  registerUser: `${API_URL}/api/user/register`,

  loginUser: `${API_URL}/api/user/login`,

  getUsers: `${API_URL}/api/user/`,

  updateUser: `${API_URL}/api/user/`,  // User ID will be appended dynamically

  getWorkoutPlans: `${API_URL}/api/workoutplans`,

  createWorkoutPlan: `${API_URL}/api/workoutplans`,

  getEquipment: `${API_URL}/api/equipment`,

  createEquipment: `${API_URL}/api/equipment`,

  getMonthlyPlans: `${API_URL}/api/monthlyplans`,

  createMonthlyPlan: `${API_URL}/api/monthlyplans`,

  // Other endpoints...

};
```

We use this endpoint in a service file to create a new monthly plan:

```
// src/services/monthlyPlans.js

import { apiEndpoints } from '../config/apiConfig';
import axios from 'axios';

export const createMonthlyPlan = async (monthlyPlanData) => {
  try {
    const response = await axios.post(apiEndpoints.createMonthlyPlan, monthlyPlanData);
    return response.data;
  } catch (error) {
    console.error('Error creating monthly plan:', error);
    throw error;
  }
};
```

To create a new monthly plan, you would call the createMonthlyPlan function like this:

```
const newMonthlyPlan = {
  name: 'Summer Fitness Plan',
  duration: '4 weeks',
  details: [
    { week: 1, focus: 'Strength Training' },
    { week: 2, focus: 'Cardio' },
    { week: 3, focus: 'Flexibility' },
    { week: 4, focus: 'Mixed' },
  ],
};
 createMonthlyPlan(newMonthlyPlan).then(plan => {
  console.log('Created Monthly Plan:', plan);
});
```

Integration with Frontend

To integrate MongoDB with a frontend application for authentication and data fetching, we will typically follow these steps:

1. **Set up your backend:**
   - Use Node.js with Express.js to create an API server.
   - Use Mongoose to interact with MongoDB.
   - Implement user authentication using libraries like Passport.js or JWT (JSON Web Tokens).
2. **Set up MongoDB:**
   - Create a database and collection(s) in MongoDB.
   - Define Mongoose schemas and models.
3. **Implement authentication:**
   - Use Passport.js or JWT for authentication.
   - Create login and registration routes.
   - Hash passwords using bcrypt.
4. **Set up your frontend:**
   - Use React, Vue, Angular, or another frontend framework/library.
   - Create forms for user registration and login.
   - Use Axios or Fetch API to make HTTP requests to your backend.
5. **Data fetching:**
   - Fetch data from MongoDB via your backend API.
   - Display data on the frontend.

Error Handling and Validation

Backend Error Handling Strategy:

- Use libraries like `express-validator` to validate incoming request data.
- Check required fields, data types, and constraints.

Code:

```
const { check, validationResult } = require('express-validator');

router.post('/register', [
  check('name', 'Name is required').not().isEmpty(),

  check('email', 'Please include a valid email').isEmail(),

  check('password', 'Please enter a password with 6 or more characters').isLength({ min: 6 })

], (req, res) => {

  const errors = validationResult(req);

  if (!errors.isEmpty()) {

    return res.status(400).json({ errors: errors.array() });

  }

  // Proceed with registration

});
```

Centralized Error Handling Middleware:

- Create an error handling middleware that captures errors and sends standardized error responses.
- Log errors for further analysis and debugging.

```
const errorHandler = (err, req, res, next) => {

  console.error(err.stack);

  res.status(500).json({ errors: [{ msg: 'Server error' }] });

};

app.use(errorHandler);
```

Frontend Error Handling Strategy:

Displaying Error Messages:

- Capture error responses from the backend and display them to the user.
- Use state management to track and display errors.

Code:

```
const [errors, setErrors] = useState([]);

const onSubmit = async e => {

  e.preventDefault();

  try {

    const response = await axios.post('/api/register', formData);

    console.log('User registered');

  } catch (err) {

    setErrors(err.response.data.errors);

  }

};

// Render errors

{errors.length > 0 && (

  <ul>

    {errors.map((error, index) => (

      <li key={index}>{error.msg}</li>

    ))}

  </ul>

)}
```

Security Considerations
Outline the security measures implemented:

- **Authentication**

  Secure token-based authentication involves using JSON Web Tokens (JWT) to manage user sessions in a secure and stateless manner.

  Steps:

. User Registration

When a user registers, hash their password using a strong hashing algorithm like bcrypt before storing it in the database.

Code:

```
import express from 'express';

import bcrypt from 'bcrypt';

import User from './models/User.js'; // Assuming you have a User model

const app = express();

app.use(express.json());

app.post('/register', async (req, res) => {

  const { name, email, password } = req.body;

  try {

    const salt = await bcrypt.genSalt(10);

    const hashedPassword = await bcrypt.hash(password, salt);

  const newUser = new User({

      name,

      email,

      password: hashedPassword

    });

await newUser.save();

    res.status(201).json({ msg: 'User registered successfully' });

  } catch (err) {

    res.status(500).json({ msg: 'Server error' });

  }
});
```

User Login:

When a user logs in, verify their password and generate a JWT if the credentials are valid.

```
import jwt from 'jsonwebtoken';

import dotenv from 'dotenv';

dotenv.config();

app.post('/login', async (req, res) => {

  const { email, password } = req.body;

try {

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

    if (!user) {

      return res.status(400).json({ msg: 'Invalid credentials' });

    }

const isMatch = await bcrypt.compare(password, user.password);

    if (!isMatch) {

      return res.status(400).json({ msg: 'Invalid credentials' });

    }

 const payload = { userId: user.id };

    const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '1h' });

  res.json({ token });

  } catch (err) {

    res.status(500).json({ msg: 'Server error' });

  }
});
```

- **Data Encryption:**

Encrypting sensitive data at rest and in transit is crucial for protecting user information and ensuring data security.

Setup and Configuration:

First, make sure you have the necessary dependencies installed. Your package.json already includes jsonwebtoken, bcrypt, express, mongoose, and dotenv.

Environment Variables:

.env file:

```
JWT_SECRET=your_jwt_secret

MONGO_URI=your_mongodb_uri
```

User Registration:

When a user registers, hash their password using `bcrypt` and store it in the database.

models/User.js:

```
import mongoose from 'mongoose';

const userSchema = new mongoose.Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  password: { type: String, required: true },

});

export default mongoose.model('User', userSchema);
```

server.js:

```
import express from 'express';

import mongoose from 'mongoose';

import bcrypt from 'bcrypt';

import jwt from 'jsonwebtoken';

import dotenv from 'dotenv';

import bodyParser from 'body-parser';

import cors from 'cors';

import User from './models/User.js';

dotenv.config();

const app = express();

app.use(bodyParser.json());

app.use(cors());

mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology:
true })

  .then(() => console.log('MongoDB connected'))

  .catch(err => console.log(err));

app.post('/register', async (req, res) => {

  const { name, email, password } = req.body;

  try {

    const salt = await bcrypt.genSalt(10);

    const hashedPassword = await bcrypt.hash(password, salt);

   const newUser = new User({

      name,

      email,
      password: hashedPassword
    });
   await newUser.save();
    res.status(201).json({ msg: 'User registered successfully' });
  } catch (err) {
    res.status(500).json({ msg: 'Server error' });
  }
});
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

User Login:

When a user logs in, verify their password and generate a JWT if the credentials are valid.

server.js (continued):

```
app.post('/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ msg: 'Invalid credentials' });
    }
 const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ msg: 'Invalid credentials' });

    }

 const payload = { userId: user.id };

    const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '1h' });

    res.json({ token });

  } catch (err) {

    res.status(500).json({ msg: 'Server error' });

  }

});
```