Project Summary

Certainly! Here's a plain-English overview of the project's codebase:

Project Overview

This project is a full-stack web application built using React for the frontend and Node.js with Express for the backend. It integrates with a MongoDB database and uses Vite as the build tool. The application provides user authentication, a form for user details, and generates a personalized fitness and diet plan using the Google Generative AI API.

Frontend

1. **ESLint Configuration**:
   - The project uses ESLint to enforce coding standards. It is configured to extend recommended configurations for JavaScript, React hooks, and React refresh in a Vite setup. It also sets up global browser variables and parser options for ECMAScript 2020 (ES2020).

2. **Vite Configuration**:
   - Vite is configured to use the React plugin and Tailwind CSS for styling. This setup ensures that the application is optimized for development and production builds.

3. **HTML Structure**:
   - The `index.html` file sets up the basic structure of the web page, including a viewport meta tag, a title, and a script tag that loads the main React application.

4. **Main React Application**:

- The `main.jsx` file initializes the React application and renders the `App` component.

- The `App` component uses React Router to define routes for different pages, including an authentication form, a user details form, and a dashboard.

5. **Authentication Form**:

- The `AuthForm` component handles user registration and login. It uses state to manage form data and navigate between routes. It sends requests to the backend API to register or log in users.

6. **Styling**:

- The project uses Tailwind CSS for styling. Custom styles are defined in `index.css` to enhance the look and feel of the application.

Backend

1. **Express Server**:

- The server is built using Express, a popular Node.js web framework. It sets up middleware for handling JSON requests and CORS (Cross-Origin Resource Sharing).

- A middleware function is added to log incoming requests and their bodies.

- The server connects to a MongoDB database using the Mongoose ORM.

- Routes for user authentication and plan generation are mounted.

2. **Database Connection**:

- The `connectDB` function connects to the MongoDB database using the connection URI from environment variables. It logs the connection status.

3. **User Model**:

- The `User` model defines the schema for user data, including fields for name, email, and password. It ensures that the email field is unique.

4. **Authentication Routes**:
   - The `auth` router handles user registration and login. It validates input, checks for existing users, and creates or authenticates users in the database. It returns appropriate responses to the frontend.

5. **Plan Generation Route**:
   - The `plan` router handles the generation of a personalized fitness and diet plan using the Google Generative AI API. It constructs a prompt based on user details and sends it to the API. The generated plan is returned to the frontend as a JSON response.

Environment Configuration

- The project uses environment variables to store sensitive information like database URLs and API keys. These are loaded using the `dotenv` package.

Summary

This project is a well-structured web application that combines modern frontend and backend technologies. It provides user authentication, form handling, and integrates with an AI service to generate personalized plans. The use of Vite, ESLint, and Tailwind CSS ensures a smooth development experience and high-quality code.

AI Recommendations

Certainly! Let's go through the codebase and suggest specific improvements in style, performance,

and maintainability.

ESLint Configuration

```javascript
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import { defineConfig, globalIgnores } from 'eslint/config'

export default defineConfig([
  globalIgnores(['dist']),
  {
    files: ['**/*.{js,jsx}'],
    extends: [
      js.configs.recommended,
      reactHooks.configs['recommended-latest'],
      reactRefresh.configs.vite,
    ],
    languageOptions: {
      ecmaVersion: 'latest', // Use 'latest' instead of a specific version for better future-proofing
      globals: globals.browser,
      parserOptions: {
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
```

```
      },
    },
    rules: {
      'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z_]' }],
      'react-hooks/rules-of-hooks': 'error', // Ensure React hooks are used correctly
    },
  },
])
```

HTML File

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

```

### Vite Configuration

```javascript
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from 'tailwindcss'

// https://vite.dev/config/
export default defineConfig({
  plugins: [
    react(),
    tailwindcss()
  ],
})
```

Express Server

```javascript
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const authRoutes = require('./routes/auth');
const planRoutes = require('./routes/plan');
```
```

```
require('dotenv').config();

const app = express();
app.use(cors());
app.use(express.json());

// Request logging middleware
app.use((req, res, next) => {
  console.log(`Incoming ${req.method} request to ${req.url}`);
  console.log('Request body:', req.body);
  next();
});

mongoose.connect(process.env.MONGO_URL)
  .then(() => console.log('MongoDB connected'))
  .catch((err) => console.error('MongoDB connection error:', err));

// Mount auth and plan routes
app.use('/api/auth', authRoutes);
app.use('/api', planRoutes);

const PORT = process.env.PORT || 5000; // Use environment variable for port
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

MongoDB Connection

```javascript
import mongoose from 'mongoose';

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('MongoDB connected');
  } catch (err) {
    console.error('MongoDB connection failed:', err.message);
  }
};

export default connectDB;
```

User Model

```javascript
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
```

```
  password: { type: String, required: true },
});

const User = mongoose.model('User', UserSchema);

module.exports = User;
```

Auth Routes

```javascript
const express = require('express');
const User = require('../models/User');
const bcrypt = require('bcryptjs'); // Add bcrypt for password hashing
const jwt = require('jsonwebtoken'); // Add JWT for token generation
const router = express.Router();

// Register route
router.post('/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;

    if (!name || !email || !password) {
      return res.status(400).json({ msg: "Please fill in all fields" });
    }
```

```
    const existing = await User.findOne({ email });
    if (existing) return res.status(400).json({ msg: 'User already exists' });

    const hashedPassword = await bcrypt.hash(password, 10); // Hash the password
    const newUser = new User({ name, email, password: hashedPassword });
    await newUser.save();

    return res.status(201).json({
      msg: 'User registered successfully',
      user: {
        _id: newUser._id,
        name: newUser.name,
        email: newUser.email,
      }
    });

  } catch (err) {
    console.error('Error in /register:', err);
    return res.status(500).json({ msg: 'Server error', error: err.message });
  }
});

// Login route
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
```

```
  if (!email || !password) {
    return res.status(400).json({ msg: 'Please fill in all fields' });
  }

  const user = await User.findOne({ email });
  if (!user) {
    return res.status(401).json({ msg: 'User not found' });
  }

  const isMatch = await bcrypt.compare(password, user.password); // Compare hashed password
  if (!isMatch) {
    return res.status(401).json({ msg: 'Incorrect password' });
  }

    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, { expiresIn: '1h' }); // Generate JWT token
  return res.status(200).json({
    msg: 'Login successful',
    user: {
      _id: user._id,
      name: user.name,
      email: user.email,
    },
    token
  });
```

```
  } catch (err) {
    console.error(' Error in /login:', err.message);
    return res.status(500).json({ msg: 'Server error', error: err.message });
  }
});

module.exports = router;
```

Plan Routes

```javascript
const express = require('express');
const { GoogleGenerativeAI } = require('@google/generative-ai');
const router = express.Router();

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);

router.post('/generate-plan', async (req, res) => {
  try {
    const user = req.body;

    const prompt = `
      Act as an expert Indian fitness coach and nutritionist.
      Create a personalized fitness and diet plan for an Indian user with the following details.
```

Format the entire response using markdown.

User Details:

- **Name:** ${user.name}
- **Age:** ${user.age}
- **Gender:** ${user.gender}
- **Height:** ${user.height} cm
- **Weight:** ${user.weight} kg
- **Activity Level:** ${user.activity}
- **Primary Goal:** ${user.goal}

Please generate a comprehensive plan with the following sections, using these exact headings:

Your Personalized Plan for ${user.name}

1. Workout Routine

Provide a 3-day sample workout plan. Include exercises, sets, and reps.

2. Meal Plan

Provide a sample one-day meal plan (Breakfast, Lunch, Dinner, Snacks). Focus on common Indian foods. Give estimated portion sizes.

3. Lifestyle Tips

Provide 3-5 actionable lifestyle tips to help achieve the user's goal.

`;

```
    const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });

    const result = await model.generateContent(prompt);
    const text = result.text;

    res.json({ plan: text });
  } catch (err) {
    console.error('Gemini API Error:', err);
    res.status(500).json({ msg: 'Failed to generate plan', error: err.message });
  }
});

module.exports = router;
```

CSS

```css
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
```

```
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}
```

```
.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}

:root {
  font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
  line-height: 1.5;
  font-weight: 400;

  color-scheme: light dark;
  color: rgba(255, 255, 255, 0.87);
  background-color: #242424;

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

a {
  font-weight: 500;
```

```
  color: #646cff;
  text-decoration: inherit;
}
a:hover {
  color: #535bf2;
}

body {
  margin: 0;
  display: flex;
  place-items: center;
  min-width: 320px;
  min-height: 100vh;
}

h1 {
  font-size: 3.2em;
  line-height: 1.1;
}

button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
```

```
  font-family: inherit;
  background-color: #1a1a1a;
  cursor: pointer;
  transition: border-color 0.25s;
}
button:hover {
  border-color: #646cff;
}
button:focus,
button:focus-visible {
  outline: 4px auto -webkit-focus-ring-color;
}

@media (prefers-color-scheme: light) {
  :root {
    color: #213547;
    background-color: #ffffff;
  }
  a:hover {
    color: #747bff;
  }
  button {
    background-color: #f9f9f9;
  }
}
```

App.jsx

```javascript
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';

import AuthForm from './components/AuthForm';
import UserDetailsForm from './pages/UserDetailsForm';
import Dashboard from './pages/Dashboard';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<AuthForm />} />
        <Route path="/form" element={<UserDetailsForm />} />
        <Route path="/dashboard" element={<Dashboard />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

main.jsx

```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

AuthForm Component

```javascript
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';

function AuthForm() {
  const [isSignup, setIsSignup] = useState(false);
  const [formData, setFormData] = useState({
    name: '',
    email: '',
```

```
  password: '',
});

const navigate = useNavigate();

const handleChange = (e) => {
  setFormData((prev) => ({
    ...prev,
    [e.target.name]: e.target.value,
  }));
};

const handleSubmit = async (e) => {
  e.preventDefault();

  const endpoint = isSignup ? '/api/auth/register' : '/api/auth/login';

  try {
    const res = await fetch(`${import.meta.env.VITE_API_BASE_URL}${endpoint}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    });
```

```
    const data = await res.json();

    if (!res.ok) {
      alert(data.msg || 'Something went wrong');
      return;
    }

    console.log('Success:', data);
    navigate('/form');
  } catch (err) {
    console.error('Error:', err);
    alert('Server error');
  }
};

return (
  <div className="min-h-screen w-screen flex items-center justify-center bg-gray-900">
    <div className="w-full max-w-md p-8 bg-white rounded-lg shadow-md">
      <form onSubmit={handleSubmit}>
        <h2 className="text-2xl font-bold mb-6 text-center text-gray-800">
          {isSignup ? 'Sign Up' : 'Login'}
        </h2>

        {isSignup && (
          <input
            type="text"
```

```
            name="name"
            placeholder="Name"
            value={formData.name}
            onChange={handleChange}
            required
                        className="w-full mb-4 p-3 border rounded focus:outline-none focus:ring-2
focus:ring-blue-400"
          />
        )}

        <input
          type="email"
          name="email"
          placeholder="Email"
          value={formData.email}
          onChange={handleChange}
          required
                      className="w-full mb-4 p-3 border rounded focus:outline-none focus:ring-2
focus:ring-blue-400"
        />

        <input
          type="password"
          name="password"
          placeholder="Password"
          value={formData.password}
```

```
          onChange={handleChange}
          required
                      className="w-full mb-4 p-3 border rounded focus:outline-none focus:ring-2
focus:ring-blue-400"
        />

        <button
          type="submit"
          className="w-full bg-blue-600 text-white py-3 rounded hover:bg-blue-700 transition"
        >
          {isSignup ? 'Create Account' : 'Login'}
        </button>

        <p
          onClick={() => setIsSignup(!isSignup)}
          className="mt-4 text-center text-sm text-gray-600 cursor-pointer hover:text-blue-600"
        >
          {isSignup
            ? 'Already have an account? Login'
            : "Don't have an account? Sign Up"}
        </p>
      </form>
    </div>
  </div>
 );
}
```

export default AuthForm;

```

### Summary of Improvements

1. **ESLint Configuration**: Updated to use `latest` for `ecmaVersion` and added `react-hooks/rules-of-hooks` to ensure React hooks are used correctly.
2. **Express Server**: Added environment variable for port and combined route imports.
3. **Auth Routes**: Added password hashing using `bcryptjs` and JWT token generation for secure authentication.
4. **CSS**: Kept the same, but ensured it is consistent and maintainable.
5. **React Components**: Ensured consistent and maintainable code, including proper form handling and navigation.

These changes should improve the codebase's style, performance, and maintainability.

## Complexity Report

(anonymous) - CC: 1 - LOC: 2

(anonymous) - CC: 1 - LOC: 1

(anonymous) - CC: 1 - LOC: 1

(anonymous) - CC: 1 - LOC: 2

connectDB - CC: 2 - LOC: 8

(anonymous) - CC: 6 - LOC: 29

(anonymous) - CC: 6 - LOC: 32

(anonymous) - CC: 1 - LOC: 12
```