**Project: AI-Powered News Aggregator**

**Description:**

A web app that aggregates news from multiple sources (e.g., Google News, Bing News, NewsAPI, or RSS feeds) and uses AI (e.g., OpenAI's API, LangChain, or your PrivateGPT setup) to summarize articles, categorize them, and provide personalized recommendations based on user interests.

**Tech Stack:**

* **Frontend:** React (Next.js optional) + Tailwind CSS
* **Backend:** Node.js with Express.js
* **Database:** PostgreSQL or MongoDB
* **Caching:** Redis
* **AI Integration:** OpenAI API (GPT) or PrivateGPT
* **Authentication:** OAuth (Google, GitHub) or JWT
* **Dockerization:** Yes (with Nginx for load balancing)

**Key Features:**

✅ **News Aggregation** – Fetch and display news from multiple sources  
✅ **AI Summarization** – Generate short summaries of news articles  
✅ **Personalized Recommendations** – Suggest news based on reading history  
✅ **Bookmark & Read Later** – Users can save articles for later reading  
✅ **Trending Topics** – Show trending news topics using AI analysis  
✅ **Dark Mode UI** – A sleek, modern interface with light/dark mode

--------------------------------------------------------------------------------------------------------------------------------------------------------

**Project Roadmap:**

1. **Setup & Planning**
   * Define the news sources (Google News, RSS feeds, NewsAPI, etc.).
   * Choose the AI model (OpenAI API, PrivateGPT).
   * Decide on the database (PostgreSQL or MongoDB).
   * Plan the UI (Wireframe or rough design).
2. **Backend Development (Node.js + Express.js)**
   * Set up Express.js with API routes.
   * Integrate a news API to fetch articles.
   * Implement AI summarization (GPT model).
   * Store user preferences and reading history in the database.
   * Set up Redis caching for performance.
3. **Frontend Development (React + Tailwind CSS)**
   * Create a clean, user-friendly UI.
   * Build components for news listing, summaries, and bookmarks.
   * Add authentication (Google OAuth or JWT).
   * Implement dark mode and responsive design.
4. **Dockerization & Deployment**
   * Containerize the backend, frontend, and database.
   * Set up Nginx for load balancing.
   * Deploy the application (AWS, DigitalOcean, or self-hosted).

--------------------------------------------------------------------------------------------------------------------------------------------------------

**Step 1: Backend Development (Node.js + Express.js) 🚀**

We'll start by setting up a basic Express.js server that will:  
✅ Fetch news from an API (e.g., NewsAPI or RSS feeds).  
✅ Provide an endpoint to return summarized articles.  
✅ Implement authentication (OAuth or JWT).  
✅ Store user preferences and reading history.  
✅ Use Redis for caching.

**1.1 - Setting Up the Project**

First, let's initialize a Node.js project and install dependencies:

mkdir ai-news-aggregator && cd ai-news-aggregator

mkdir backend && cd backend

npm init -y

**1.2 - Install Dependencies**

npm install express axios dotenv cors redis mongoose passport passport-google-oauth20 jsonwebtoken bcryptjs

* express → Backend framework
* axios → Fetch news from APIs
* dotenv → Manage environment variables
* cors → Handle cross-origin requests
* redis → Caching for performance
* mongoose → MongoDB ORM (switchable to PostgreSQL later)
* passport & passport-google-oauth20 → OAuth authentication
* jsonwebtoken → JWT authentication
* bcryptjs → Password hashing for users

**1.3 - Set Up Express Server**

Now, let's create a basic Express server.

📂 Inside the backend folder, create a file: **server.js**

require("dotenv").config();

const express = require("express");

const cors = require("cors");

const axios = require("axios");

const redis = require("redis");

const app = express();

const PORT = process.env.PORT || 5000;

// Middleware

app.use(cors());

app.use(express.json());

// Redis Client Setup

const redisClient = redis.createClient({

url: process.env.REDIS\_URL || "redis://localhost:6379",

});

redisClient.connect().catch(console.error);

// Sample News Route

app.get("/news", async (req, res) => {

try {

const cachedNews = await redisClient.get("news");

if (cachedNews) {

return res.json(JSON.parse(cachedNews));

}

const response = await axios.get("https://newsapi.org/v2/top-headlines", {

params: {

country: "us",

apiKey: process.env.NEWS\_API\_KEY,

},

});

const newsData = response.data.articles;

// Cache news for 30 minutes

await redisClient.setEx("news", 1800, JSON.stringify(newsData));

res.json(newsData);

} catch (error) {

console.error(error);

res.status(500).json({ error: "Failed to fetch news" });

}

});

// Start Server

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

**1.4 - Environment Variables**

Create a .env file in the backend directory:

ini

CopyEdit

PORT=5000

NEWS\_API\_KEY=your\_news\_api\_key\_here

REDIS\_URL=redis://localhost:6379

**Next Steps**

✅ **Basic server setup complete!**  
🔹 Next, do you want to add **authentication (OAuth/JWT)** or build **AI-powered news summarization** next? 🚀

**Step 2: Add OAuth Authentication (Google) 🔐**

We'll set up authentication using **Google OAuth 2.0** with passport-google-oauth20.

**2.1 - Install Dependencies**

Run the following command in the backend folder:

sh

CopyEdit

npm install passport passport-google-oauth20 express-session

* passport → Authentication middleware
* passport-google-oauth20 → Google OAuth strategy
* express-session → Stores user sessions

**2.2 - Configure Passport.js**

Create a new file 📂 backend/config/passport.js:

javascript

CopyEdit

const passport = require("passport");

const GoogleStrategy = require("passport-google-oauth20").Strategy;

passport.use(

new GoogleStrategy(

{

clientID: process.env.GOOGLE\_CLIENT\_ID,

clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,

callbackURL: "/auth/google/callback",

},

(accessToken, refreshToken, profile, done) => {

console.log("User Profile:", profile);

return done(null, profile);

}

)

);

// Serialize and Deserialize User

passport.serializeUser((user, done) => {

done(null, user);

});

passport.deserializeUser((obj, done) => {

done(null, obj);

});

**2.3 - Update Express Server for Authentication**

Modify server.js to include authentication routes:

javascript

CopyEdit

require("dotenv").config();

const express = require("express");

const cors = require("cors");

const session = require("express-session");

const passport = require("passport");

require("./config/passport");

const app = express();

const PORT = process.env.PORT || 5000;

// Middleware

app.use(cors());

app.use(express.json());

app.use(

session({

secret: "news-secret",

resave: false,

saveUninitialized: true,

})

);

app.use(passport.initialize());

app.use(passport.session());

// Google OAuth Routes

app.get(

"/auth/google",

passport.authenticate("google", { scope: ["profile", "email"] })

);

app.get(

"/auth/google/callback",

passport.authenticate("google", { failureRedirect: "/login" }),

(req, res) => {

res.redirect("/dashboard"); // Redirect after successful login

}

);

app.get("/auth/logout", (req, res) => {

req.logout(() => {

res.redirect("/");

});

});

app.get("/auth/user", (req, res) => {

res.json(req.user || null);

});

// Start Server

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

**2.4 - Environment Variables**

Update the .env file with your Google OAuth credentials:

ini

CopyEdit

GOOGLE\_CLIENT\_ID=your\_google\_client\_id

GOOGLE\_CLIENT\_SECRET=your\_google\_client\_secret

👉 **Get these from Google Developer Console:**

1. Create a new project.
2. Go to **APIs & Services > Credentials**.
3. Create an OAuth 2.0 Client ID.
4. Set redirect URI to http://localhost:5000/auth/google/callback.
5. Copy the **Client ID** and **Client Secret** into .env.

**2.5 - Testing the Authentication**

1. Start your backend server:

sh

CopyEdit

node server.js

1. Open http://localhost:5000/auth/google in your browser.
2. Login with Google.
3. After authentication, you'll be redirected to /dashboard.
4. Check http://localhost:5000/auth/user to see logged-in user details.

**Next Steps**

✅ **OAuth authentication is working!**  
🔹 Next, do you want to:  
1️⃣ **Store user info in the database** (MongoDB/PostgreSQL)?  
2️⃣ **Proceed with AI-powered summarization of news articles**?

**Step 3: Store User Info in MongoDB 🚀**

We'll use **MongoDB** to store user details after authentication. Later, we can switch to PostgreSQL if needed.

**3.1 - Install MongoDB Dependencies**

Run this inside the backend folder:

sh

CopyEdit

npm install mongoose

* mongoose → MongoDB ORM for Node.js

**3.2 - Setup MongoDB in Docker (Recommended for You) 🐳**

Since you're using Docker, let's run MongoDB inside a container.

Run this command:

sh

CopyEdit

docker run --name mongodb -d -p 27017:27017 mongo

Or, if using **Docker Compose**, add this to your docker-compose.yml:

yaml

CopyEdit

version: "3.8"

services:

mongodb:

image: mongo

container\_name: mongodb

ports:

- "27017:27017"

volumes:

- mongodb\_data:/data/db

volumes:

mongodb\_data:

Then start it:

sh

CopyEdit

docker-compose up -d

**3.3 - Connect Node.js to MongoDB**

Modify .env to include:

bash

CopyEdit

MONGO\_URI=mongodb://localhost:27017/newsDB

Now, create a new file 📂 backend/config/db.js:

javascript

CopyEdit

const mongoose = require("mongoose");

const connectDB = async () => {

try {

await mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log("✅ MongoDB Connected...");

} catch (error) {

console.error("❌ MongoDB Connection Error:", error);

process.exit(1);

}

};

module.exports = connectDB;

**3.4 - Define the User Model**

Create a 📂 backend/models/User.js file:

javascript

CopyEdit

const mongoose = require("mongoose");

const UserSchema = new mongoose.Schema({

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

name: String,

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

profilePic: String,

createdAt: { type: Date, default: Date.now },

});

module.exports = mongoose.model("User", UserSchema);

**3.5 - Modify Google OAuth to Save Users**

Update backend/config/passport.js to save users in the database:

javascript

CopyEdit

const passport = require("passport");

const GoogleStrategy = require("passport-google-oauth20").Strategy;

const User = require("../models/User");

passport.use(

new GoogleStrategy(

{

clientID: process.env.GOOGLE\_CLIENT\_ID,

clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,

callbackURL: "/auth/google/callback",

},

async (accessToken, refreshToken, profile, done) => {

try {

let user = await User.findOne({ googleId: profile.id });

if (!user) {

user = new User({

googleId: profile.id,

name: profile.displayName,

email: profile.emails[0].value,

profilePic: profile.photos[0].value,

});

await user.save();

}

return done(null, user);

} catch (error) {

return done(error, null);

}

}

)

);

passport.serializeUser((user, done) => {

done(null, user.id);

});

passport.deserializeUser(async (id, done) => {

try {

const user = await User.findById(id);

done(null, user);

} catch (error) {

done(error, null);

}

});

**3.6 - Load DB Connection in Server**

Modify server.js to connect to MongoDB:

javascript

CopyEdit

require("dotenv").config();

const express = require("express");

const cors = require("cors");

const session = require("express-session");

const passport = require("passport");

const connectDB = require("./config/db");

require("./config/passport");

const app = express();

const PORT = process.env.PORT || 5000;

// Connect to Database

connectDB();

// Middleware

app.use(cors());

app.use(express.json());

app.use(

session({

secret: "news-secret",

resave: false,

saveUninitialized: true,

})

);

app.use(passport.initialize());

app.use(passport.session());

// Auth Routes

app.get("/auth/google", passport.authenticate("google", { scope: ["profile", "email"] }));

app.get("/auth/google/callback",

passport.authenticate("google", { failureRedirect: "/login" }),

(req, res) => res.redirect("/dashboard")

);

app.get("/auth/logout", (req, res) => {

req.logout(() => res.redirect("/"));

});

app.get("/auth/user", (req, res) => {

res.json(req.user || null);

});

// Start Server

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

**3.7 - Test Everything**

1️⃣ **Start MongoDB (if using Docker)**:

sh

CopyEdit

docker-compose up -d

2️⃣ **Run the Node.js backend**:

sh

CopyEdit

node server.js

3️⃣ **Try logging in**:  
Go to http://localhost:5000/auth/google, log in, then check:

* http://localhost:5000/auth/user → Should show user details.
* Open MongoDB shell (docker exec -it mongodb mongosh) and run:

sh

CopyEdit

use newsDB

db.users.find()

This should show stored users.

**Next Steps**

✅ **User authentication + MongoDB storage is working!**  
🔹 Next, do you want to:  
1️⃣ **Start AI-powered news summarization?**  
2️⃣ **Design the frontend (React + Tailwind)?**

--------------------------------------------------------------------------------------------------------------------------------------------------------

**🧠 Step 4: AI-Powered News Summarization**

We'll use **OpenAI API (GPT-4)** or **PrivateGPT** to summarize news articles.

**🛠 4.1 - Install Required Dependencies**

Inside your backend folder, install:

sh

CopyEdit

npm install axios openai

* axios → Fetches news articles
* openai → Calls OpenAI API for summarization

If using **PrivateGPT**, you don’t need openai, just ensure it’s running in your Docker.

**📄 4.2 - Configure OpenAI API (If Using GPT-4)**

Add this to .env:

ini

CopyEdit

OPENAI\_API\_KEY=your-openai-api-key

If using **PrivateGPT**, add:

bash

CopyEdit

PRIVATEGPT\_URL=http://localhost:8000/v1/chat/completions

**📜 4.3 - Create Summarization Service**

Create a new file 📂 backend/services/summarizer.js:

javascript

CopyEdit

const axios = require("axios");

const OpenAI = require("openai");

const openai = new OpenAI({ apiKey: process.env.OPENAI\_API\_KEY });

async function summarizeArticle(articleText) {

try {

const response = await openai.chat.completions.create({

model: "gpt-4",

messages: [

{ role: "system", content: "You are an AI that summarizes news articles in a short, engaging way." },

{ role: "user", content: `Summarize this news article in 3 sentences:\n\n${articleText}` },

],

temperature: 0.7,

max\_tokens: 150,

});

return response.choices[0].message.content;

} catch (error) {

console.error("❌ Error summarizing:", error);

return "Error generating summary.";

}

}

module.exports = { summarizeArticle };

If using **PrivateGPT**, modify the openai call like this:

javascript

CopyEdit

const axios = require("axios");

async function summarizeArticle(articleText) {

try {

const response = await axios.post(process.env.PRIVATEGPT\_URL, {

model: "gpt-4",

messages: [

{ role: "system", content: "You are an AI that summarizes news articles." },

{ role: "user", content: `Summarize this article:\n\n${articleText}` },

],

temperature: 0.7,

max\_tokens: 150,

});

return response.data.choices[0].message.content;

} catch (error) {

console.error("❌ Error summarizing:", error);

return "Error generating summary.";

}

}

module.exports = { summarizeArticle };

**📡 4.4 - Create News API Route**

Modify 📂 backend/routes/news.js:

javascript

CopyEdit

const express = require("express");

const axios = require("axios");

const { summarizeArticle } = require("../services/summarizer");

const router = express.Router();

router.get("/fetch", async (req, res) => {

try {

const { data } = await axios.get(

`https://newsapi.org/v2/top-headlines?country=us&apiKey=${process.env.NEWS\_API\_KEY}`

);

const summarizedNews = await Promise.all(

data.articles.slice(0, 5).map(async (article) => ({

title: article.title,

url: article.url,

summary: await summarizeArticle(article.content || article.description),

}))

);

res.json(summarizedNews);

} catch (error) {

console.error("❌ Error fetching news:", error);

res.status(500).json({ error: "Failed to fetch news" });

}

});

module.exports = router;

**📢 Explanation:**

1. **Fetches news** from NewsAPI.org.
2. **Summarizes top 5 articles** using OpenAI API or PrivateGPT.
3. **Returns summarized news** with a title, URL, and AI-generated summary.

**🖥 4.5 - Register News Routes in server.js**

Modify backend/server.js:

javascript

CopyEdit

const newsRoutes = require("./routes/news");

app.use("/news", newsRoutes);

**🚀 4.6 - Test the Summarization API**

Start the backend:

sh

CopyEdit

node server.js

Then, open **Postman** or a browser and hit:

bash

CopyEdit

http://localhost:5000/news/fetch

✅ You should get **summarized news articles** with **title, URL, and AI-generated summary**.

**📌 Next Steps**

* Do you want to **build the frontend next** (React + Tailwind UI)?
* Or should we **store news summaries in MongoDB for caching**?

--------------------------------------------------------------------------------------------------------------------------------------------------------

**🎨 Step 5: Building the Frontend (React + Tailwind UI)**

We'll create a **React-based UI** for our AI-powered news aggregator. The UI will: ✅ Fetch and display summarized news articles  
✅ Include **OAuth login (Google, GitHub)**  
✅ Use **Tailwind CSS** for styling

**🛠 5.1 - Set Up React Frontend**

Inside your project folder, run:

npx create-react-app frontend

cd frontend

npm install axios react-router-dom @reduxjs/toolkit react-redux @headlessui/react tailwindcss@latest postcss@latest autoprefixer@latest

* axios → Fetch data from the backend
* react-router-dom → Enable routing
* @reduxjs/toolkit & react-redux → Manage state
* tailwindcss → Styling
* @headlessui/react → Beautiful UI components

Then initialize Tailwind:

npx tailwindcss init -p

Modify tailwind.config.js:

module.exports = {

content: ["./src/\*\*/\*.{js,jsx,ts,tsx}"],

theme: {

extend: {},

},

plugins: [],

};

Add Tailwind to src/index.css:

css

CopyEdit

@tailwind base;

@tailwind components;

@tailwind utilities;

**🖥 5.2 - Build the News Feed Component**

Create 📂 src/components/NewsFeed.js:

javascript

CopyEdit

import { useEffect, useState } from "react";

import axios from "axios";

const NewsFeed = () => {

const [news, setNews] = useState([]);

const [loading, setLoading] = useState(true);

useEffect(() => {

axios.get("http://localhost:5000/news/fetch")

.then((response) => {

setNews(response.data);

setLoading(false);

})

.catch((error) => console.error("Error fetching news:", error));

}, []);

if (loading) return <p className="text-center text-xl">Loading...</p>;

return (

<div className="max-w-4xl mx-auto p-5">

<h1 className="text-3xl font-bold text-center mb-6">AI-Powered News</h1>

<div className="grid gap-6">

{news.map((article, index) => (

<div key={index} className="p-4 bg-white shadow-lg rounded-lg">

<h2 className="text-xl font-bold">{article.title}</h2>

<p className="text-gray-600">{article.summary}</p>

<a href={article.url} className="text-blue-500 mt-2 inline-block" target="\_blank" rel="noopener noreferrer">

Read More →

</a>

</div>

))}

</div>

</div>

);

};

export default NewsFeed;

**📢 Explanation:**

✔ Fetches news summaries from **http://localhost:5000/news/fetch**  
✔ Displays the **title, summary, and "Read More" link**  
✔ Uses **Tailwind CSS** for styling

**🌍 5.3 - Setup React Router**

Modify 📂 src/App.js:

javascript

CopyEdit

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import NewsFeed from "./components/NewsFeed";

import Login from "./components/Login";

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<NewsFeed />} />

<Route path="/login" element={<Login />} />

</Routes>

</Router>

);

}

export default App;

**🔑 5.4 - Add OAuth Login UI**

Create 📂 src/components/Login.js:

javascript

CopyEdit

import axios from "axios";

const Login = () => {

const handleGoogleLogin = () => {

window.location.href = "http://localhost:5000/auth/google";

};

const handleGitHubLogin = () => {

window.location.href = "http://localhost:5000/auth/github";

};

return (

<div className="flex flex-col items-center justify-center min-h-screen">

<h1 className="text-3xl font-bold mb-4">Login</h1>

<button

className="bg-red-500 text-white px-4 py-2 rounded-lg mb-3"

onClick={handleGoogleLogin}

>

Login with Google

</button>

<button

className="bg-gray-800 text-white px-4 py-2 rounded-lg"

onClick={handleGitHubLogin}

>

Login with GitHub

</button>

</div>

);

};

export default Login;

**📢 Explanation:**

✔ **Redirects users** to OAuth login when they click a button  
✔ Uses **Google & GitHub** login

**🚀 5.5 - Run the Frontend**

Start the React app:

sh

CopyEdit

npm start

It should open **http://localhost:3000/** with:  
✅ A **beautiful AI-powered news feed**  
✅ A **Login page for OAuth authentication**

**📌 Next Steps**

🔹 Do you want to **connect user authentication with MongoDB/PostgreSQL**?  
🔹 Should we **cache summarized news in Redis** for faster access?

--------------------------------------------------------------------------------------------------------------------------------------------------------

**🔐 Step 6: Connect User Authentication with MongoDB**

Now, let's **store user data in MongoDB** after authentication using **OAuth (Google, GitHub)**.

**📌 6.1 - Install Required Packages**

In your backend directory, install these dependencies:

sh

CopyEdit

npm install mongoose dotenv

* mongoose → Connects Node.js to MongoDB
* dotenv → Loads environment variables

**🛠 6.2 - Setup MongoDB Connection**

Create a **.env file** (if not created) in the backend directory:

ini

CopyEdit

MONGO\_URI=mongodb://localhost:27017/ai-news

If you're using **MongoDB Atlas**, replace the URI with:

ini

CopyEdit

MONGO\_URI=mongodb+srv://<username>:<password>@cluster0.mongodb.net/ai-news?retryWrites=true&w=majority

Now, update server.js to connect to MongoDB:

javascript

CopyEdit

const mongoose = require("mongoose");

require("dotenv").config();

mongoose

.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("✅ MongoDB Connected"))

.catch((err) => console.error("❌ MongoDB Connection Error:", err));

**📝 6.3 - Define User Schema**

Create a new file 📂 models/User.js:

javascript

CopyEdit

const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({

googleId: { type: String, unique: true, sparse: true },

githubId: { type: String, unique: true, sparse: true },

name: String,

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

profilePic: String,

});

module.exports = mongoose.model("User", userSchema);

**🔑 6.4 - Modify OAuth Authentication to Store Users**

Modify 📂 routes/auth.js:

**🔹 Google Authentication**

Modify Google strategy inside passport.js:

javascript

CopyEdit

const passport = require("passport");

const GoogleStrategy = require("passport-google-oauth20").Strategy;

const GitHubStrategy = require("passport-github2").Strategy;

const User = require("../models/User");

passport.use(

new GoogleStrategy(

{

clientID: process.env.GOOGLE\_CLIENT\_ID,

clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,

callbackURL: "http://localhost:5000/auth/google/callback",

},

async (accessToken, refreshToken, profile, done) => {

try {

let user = await User.findOne({ googleId: profile.id });

if (!user) {

user = new User({

googleId: profile.id,

name: profile.displayName,

email: profile.emails[0].value,

profilePic: profile.photos[0].value,

});

await user.save();

}

done(null, user);

} catch (error) {

done(error, null);

}

}

)

);

**🔹 GitHub Authentication**

Modify GitHub strategy inside passport.js:

javascript

CopyEdit

passport.use(

new GitHubStrategy(

{

clientID: process.env.GITHUB\_CLIENT\_ID,

clientSecret: process.env.GITHUB\_CLIENT\_SECRET,

callbackURL: "http://localhost:5000/auth/github/callback",

},

async (accessToken, refreshToken, profile, done) => {

try {

let user = await User.findOne({ githubId: profile.id });

if (!user) {

user = new User({

githubId: profile.id,

name: profile.displayName || profile.username,

email: profile.emails ? profile.emails[0].value : null,

profilePic: profile.photos ? profile.photos[0].value : null,

});

await user.save();

}

done(null, user);

} catch (error) {

done(error, null);

}

}

)

);

**📌 6.5 - Test Authentication**

1. **Start MongoDB**

sh

CopyEdit

mongod

1. **Start the Backend**

sh

CopyEdit

node server.js

1. Visit:
   * http://localhost:5000/auth/google
   * http://localhost:5000/auth/github
2. Check **MongoDB (ai-news.users collection)**:

sh

CopyEdit

mongo

use ai-news

db.users.find().pretty()

**✅ Next Steps**

* **🔹 Protect API routes** (only allow authenticated users)
* **🔹 Implement JWT-based session management**

--------------------------------------------------------------------------------------------------------------------------------------------------------

**📌 7.1 - Update the Backend (CORS & Routes)**

Modify 📂 backend/server.js to allow frontend requests:

**✅ Install CORS**

Run in the backend directory:

sh

CopyEdit

npm install cors

**✅ Update server.js**

Modify server.js to allow frontend requests:

javascript

CopyEdit

const express = require("express");

const cors = require("cors");

const passport = require("passport");

require("dotenv").config();

require("./config/passport"); // Ensure passport config is loaded

const authRoutes = require("./routes/auth");

const mongoose = require("mongoose");

const app = express();

// Middleware

app.use(express.json());

app.use(cors({ origin: "http://localhost:3000", credentials: true }));

app.use("/auth", authRoutes);

mongoose.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("✅ MongoDB Connected"))

.catch((err) => console.error("❌ MongoDB Error:", err));

const PORT = process.env.PORT || 5000;

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

**📌 7.2 - Update the Frontend**

Modify 📂 frontend/src/App.js to fetch authentication status.

**✅ Install Axios**

Run in the frontend directory:

sh

CopyEdit

npm install axios react-router-dom

**✅ Update App.js**

Modify frontend/src/App.js:

import React from "react";

import { BrowserRouter as Router, Route, Routes, Navigate } from "react-router-dom";

import Dashboard from "./Dashboard"; // Create a Dashboard component

import Home from "./Home"; // Move your main content to a separate Home component

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/dashboard" element={<Dashboard />} />

{/\* Redirect unknown routes to Home \*/}

<Route path="\*" element={<Navigate to="/" />} />

</Routes>

</Router>

);

}

export default App;

**Create Home.js**

Since you moved your main content to Home.js, create a new file Home.js:

jsx

CopyEdit

import React from "react";

function Home() {

return (

<div className="container text-center mt-5">

<h1>Welcome to AI News Aggregator</h1>

<p>Get personalized AI-powered news summaries from multiple sources.</p>

<div className="d-flex flex-column align-items-center gap-3 mt-4">

<a href="http://localhost:5000/auth/google" className="btn btn-primary rounded-circle px-4 py-2">

Login with Google

</a>

<a href="http://localhost:5000/auth/github" className="btn btn-dark rounded-circle px-4 py-2">

Login with GitHub

</a>

</div>

</div>

);

}

export default Home;

**4️⃣ Create Dashboard.js**

Create a new file Dashboard.js so users see content after logging in.

jsx

CopyEdit

import React, { useEffect, useState } from "react";

import { useNavigate } from "react-router-dom";

import axios from "axios";

function Dashboard() {

const [user, setUser] = useState(null);

const navigate = useNavigate();

useEffect(() => {

axios.get("http://localhost:5000/auth/user", { withCredentials: true })

.then(response => {

if (response.data) {

setUser(response.data);

} else {

navigate("/"); // Redirect to home if not logged in

}

})

.catch(() => {

navigate("/"); // Redirect to home on error

});

}, [navigate]);

return (

<div className="container text-center mt-5">

{user ? (

<>

<h1>Welcome, {user.name}!</h1>

<p>Here are your personalized AI-powered news summaries.</p>

<a href="http://localhost:5000/auth/logout" className="btn btn-danger">

Logout

</a>

</>

) : (

<p>Loading...</p>

)}

</div>

);

}

export default Dashboard;

**📌 7.3 - Dockerize the Backend & Frontend**

Let's create **Dockerfiles** for both services.

**🔹 7.3.1 - Backend Dockerfile**

Create backend/Dockerfile:

Dockerfile

CopyEdit

FROM node:18

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 5000

CMD ["node", "server.js"]

**🔹 7.3.2 - Frontend Dockerfile**

Create frontend/Dockerfile:

Dockerfile

CopyEdit

FROM node:18

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["npm", "start"]

**🔹 7.3.3 - Create docker-compose.yml**

Create a docker-compose.yml file in the project root:

yaml

CopyEdit

version: "3.8"

services:

backend:

build: ./backend

ports:

- "5000:5000"

environment:

- MONGO\_URI=mongodb://mongo:27017/ai-news

depends\_on:

- mongo

frontend:

build: ./frontend

ports:

- "3000:3000"

depends\_on:

- backend

mongo:

image: mongo

ports:

- "27017:27017"

volumes:

- mongo\_data:/data/db

volumes:

mongo\_data:

**📌 7.4 - Run Everything**

**✅ 1. Start Docker Services**

docker-compose up --build

* 🚀 **Frontend** → http://localhost:3000
* 🚀 **Backend** → http://localhost:5000
* 🚀 **MongoDB** → Connected inside the backend container

**✅ 2. Login & Check Database**

1. Open http://localhost:3000
2. Login via **Google/GitHub**
3. Open MongoDB in a container:

docker exec -it <container\_id> mongo

use ai-news

db.users.find().pretty()