You are an expert web developer. I am ready to begin Phase 3.

Refer to the attached Phase 3 roadmap and guide me through each section of the build with clear, step-by-step instructions that include Git save points and testing to ensure things are working properly before proceeding. All code needs to be clean, optimized and secure.

Begin with step 1.1

Do you understand?

Ask any clarifying questions if needed before proceeding.

Edit

Strategized comprehensive login page implementation for admin dashboard.

5s

Phase 3: Admin Dashboard Implementation - Authentication System

I'll guide you through implementing the login page for your Subject Line Pro admin dashboard. Let's start with Step 1.1.

Step 1.1: Login Page Design and Implementation

Project Structure Setup

First, let's create the necessary folders and files for the admin dashboard:

bash

Copy

# Navigate to your frontend directory

cd frontend

# Create admin-related folders

mkdir -p src/pages/admin

mkdir -p src/components/admin

mkdir -p src/context

mkdir -p src/hooks

Create Authentication Context

Let's create an authentication context to manage login state across the admin dashboard:

bash

Copy

# Create the authentication context file

touch src/context/AuthContext.jsx

Add the following code to src/context/AuthContext.jsx:

jsx

Copy

import { createContext, useState, useEffect } from 'react';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {

const [isAuthenticated, setIsAuthenticated] = useState(false);

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

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

const [error, setError] = useState(null);

// Check if user is already logged in on component mount

useEffect(() => {

const checkLoginStatus = async () => {

setLoading(true);

try {

// Check if token exists in localStorage

const token = localStorage.getItem('authToken');

if (!token) {

setIsAuthenticated(false);

setUser(null);

setLoading(false);

return;

}

// Verify token with backend

const response = await fetch('https://api.trevormchaffie.com/api/auth/verify', {

method: 'GET',

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.ok) {

const data = await response.json();

setIsAuthenticated(true);

setUser(data.user);

} else {

// Token is invalid or expired

localStorage.removeItem('authToken');

setIsAuthenticated(false);

setUser(null);

}

} catch (err) {

console.error('Error verifying authentication:', err);

setError('Failed to verify authentication status');

setIsAuthenticated(false);

setUser(null);

} finally {

setLoading(false);

}

};

checkLoginStatus();

}, []);

// Login function

const login = async (username, password, rememberMe = false) => {

setLoading(true);

setError(null);

try {

const response = await fetch('https://api.trevormchaffie.com/api/auth/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ username, password })

});

const data = await response.json();

if (response.ok) {

setIsAuthenticated(true);

setUser(data.user);

// Store token in localStorage if rememberMe is true, otherwise in sessionStorage

if (rememberMe) {

localStorage.setItem('authToken', data.token);

} else {

sessionStorage.setItem('authToken', data.token);

}

return true;

} else {

setError(data.message || 'Invalid credentials');

return false;

}

} catch (err) {

console.error('Login error:', err);

setError('Failed to connect to authentication server');

return false;

} finally {

setLoading(false);

}

};

// Logout function

const logout = () => {

localStorage.removeItem('authToken');

sessionStorage.removeItem('authToken');

setIsAuthenticated(false);

setUser(null);

};

// Get auth token (from either localStorage or sessionStorage)

const getAuthToken = () => {

return localStorage.getItem('authToken') || sessionStorage.getItem('authToken');

};

return (

<AuthContext.Provider value={{

isAuthenticated,

user,

loading,

error,

login,

logout,

getAuthToken

}}>

{children}

</AuthContext.Provider>

);

};

Create Authentication Hook

Create a custom hook for using the authentication context:

bash

Copy

touch src/hooks/useAuth.js

Add the following code to src/hooks/useAuth.js:

javascript

Copy

import { useContext } from 'react';

import { AuthContext } from '../context/AuthContext';

export const useAuth = () => {

const context = useContext(AuthContext);

if (!context) {

throw new Error('useAuth must be used within an AuthProvider');

}

return context;

};

Create Login Page Component

Now, let's create the login page:

bash

Copy

touch src/pages/admin/LoginPage.jsx

Add the following code to src/pages/admin/LoginPage.jsx:

jsx

Copy

import { useState } from 'react';

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

import { useAuth } from '../../hooks/useAuth';

const LoginPage = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const [rememberMe, setRememberMe] = useState(false);

const [isSubmitting, setIsSubmitting] = useState(false);

const [loginError, setLoginError] = useState('');

const { login, error: authError } = useAuth();

const navigate = useNavigate();

const handleSubmit = async (e) => {

e.preventDefault();

if (!username.trim() || !password.trim()) {

setLoginError('Username and password are required');

return;

}

setIsSubmitting(true);

setLoginError('');

try {

const success = await login(username, password, rememberMe);

if (success) {

navigate('/admin/dashboard');

} else {

setLoginError(authError || 'Invalid credentials');

}

} catch (error) {

console.error('Login submission error:', error);

setLoginError('An unexpected error occurred');

} finally {

setIsSubmitting(false);

}

};

return (

<div className="min-h-screen bg-gray-100 flex flex-col justify-center items-center px-4">

<div className="w-full max-w-md">

<div className="text-center mb-10">

<h1 className="text-3xl font-bold text-primary">Subject Line Pro</h1>

<h2 className="text-xl text-gray-600">Admin Dashboard</h2>

</div>

<div className="bg-white rounded-lg shadow-lg p-8">

<h3 className="text-xl font-semibold mb-6 text-center">Login</h3>

{loginError && (

<div className="bg-red-50 border-l-4 border-red-500 p-4 mb-6">

<p className="text-red-700">{loginError}</p>

</div>

)}

<form onSubmit={handleSubmit}>

<div className="mb-6">

<label htmlFor="username" className="block text-sm font-medium text-gray-700 mb-1">

Username

</label>

<input

id="username"

type="text"

value={username}

onChange={(e) => setUsername(e.target.value)}

className="w-full p-3 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-primary"

placeholder="Enter your username"

disabled={isSubmitting}

/>

</div>

<div className="mb-6">

<label htmlFor="password" className="block text-sm font-medium text-gray-700 mb-1">

Password

</label>

<input

id="password"

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

className="w-full p-3 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-primary"

placeholder="Enter your password"

disabled={isSubmitting}

/>

</div>

<div className="flex items-center mb-6">

<input

id="rememberMe"

type="checkbox"

checked={rememberMe}

onChange={(e) => setRememberMe(e.target.checked)}

className="h-4 w-4 text-primary focus:ring-primary border-gray-300 rounded"

disabled={isSubmitting}

/>

<label htmlFor="rememberMe" className="ml-2 block text-sm text-gray-700">

Remember me

</label>

</div>

<button

type="submit"

disabled={isSubmitting}

className={`w-full bg-primary text-white py-3 px-4 rounded-md transition-colors ${

isSubmitting ? 'opacity-70 cursor-not-allowed' : 'hover:bg-primary/80'

}`}

>

{isSubmitting ? (

<span className="flex items-center justify-center">

<svg className="animate-spin -ml-1 mr-2 h-4 w-4 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">

<circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>

<path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>

</svg>

Logging in...

</span>

) : 'Login'}

</button>

</form>

</div>

<div className="text-center mt-6">

<a href="/" className="text-sm text-primary hover:underline">

Return to Website

</a>

</div>

</div>

</div>

);

};

export default LoginPage;

Update Backend API for Authentication

We need to create a couple of endpoints in your backend for authentication. Let's add them to your Express server.

Add the following route file for authentication:

bash

Copy

# Navigate to your backend directory (execute this in a new terminal)

cd ../backend

# Create auth routes file

mkdir -p src/routes

touch src/routes/authRoutes.js

Now add the following code to src/routes/authRoutes.js:

javascript

Copy

const express = require('express');

const jwt = require('jsonwebtoken');

const config = require('../config/config');

const router = express.Router();

// Login endpoint

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

try {

const { username, password } = req.body;

// Check if username and password match admin credentials

if (username === config.adminUsername && password === config.adminPassword) {

// Generate JWT token

const token = jwt.sign(

{ id: 'admin', username },

config.jwtSecret,

{ expiresIn: '7d' }

);

// Return token and user info

return res.json({

success: true,

token,

user: { username }

});

}

// Return error for invalid credentials

return res.status(401).json({

success: false,

message: 'Invalid username or password'

});

} catch (error) {

console.error('Login error:', error);

return res.status(500).json({

success: false,

message: 'Server error during authentication'

});

}

});

// Verify token endpoint

router.get('/verify', async (req, res) => {

try {

// Get token from authorization header

const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith('Bearer ')) {

return res.status(401).json({

success: false,

message: 'No token provided'

});

}

const token = authHeader.split(' ')[1];

// Verify token

jwt.verify(token, config.jwtSecret, (err, decoded) => {

if (err) {

return res.status(401).json({

success: false,

message: 'Invalid or expired token'

});

}

// Return user info

return res.json({

success: true,

user: { username: decoded.username }

});

});

} catch (error) {

console.error('Token verification error:', error);

return res.status(500).json({

success: false,

message: 'Server error during token verification'

});

}

});

module.exports = router;

Now update your config file to include JWT secret:

bash

Copy

# Update config file

touch src/config/config.js

Add or update src/config/config.js with:

javascript

Copy

require('dotenv').config();

module.exports = {

port: process.env.PORT || 3000,

nodeEnv: process.env.NODE\_ENV || 'development',

corsOrigin: process.env.CORS\_ORIGIN || 'http://localhost:5173',

rateLimitWindowMs: process.env.RATE\_LIMIT\_WINDOW\_MS || 900000, // 15 minutes

rateLimitMax: process.env.RATE\_LIMIT\_MAX || 100, // limit each IP to 100 requests per windowMs

adminUsername: process.env.ADMIN\_USERNAME || 'admin',

adminPassword: process.env.ADMIN\_PASSWORD || 'admin',

jwtSecret: process.env.JWT\_SECRET || 'your-secret-key' // Add this line

};

Update your main Express app file to include the auth routes:

javascript

Copy

// In your src/index.js or app.js file, add:

const authRoutes = require('./routes/authRoutes');

// Add this with your other route definitions

app.use('/api/auth', authRoutes);

Update your backend's .env file to include a JWT secret:

Copy

JWT\_SECRET=your\_secure\_random\_secret\_key

Update Router to Include Admin Routes

Now, let's update the frontend router to include admin routes:

bash

Copy

# Return to the frontend directory

cd ../frontend

# Create admin routes file

touch src/routes/AdminRoutes.jsx

Add the following code to src/routes/AdminRoutes.jsx:

jsx

Copy

import { Navigate, Outlet } from 'react-router-dom';

import { useAuth } from '../hooks/useAuth';

const AdminRoutes = () => {

const { isAuthenticated, loading } = useAuth();

// Show loading indicator while checking authentication

if (loading) {

return (

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

<div className="animate-spin rounded-full h-12 w-12 border-b-2 border-primary"></div>

</div>

);

}

// Redirect to login if not authenticated

if (!isAuthenticated) {

return <Navigate to="/admin/login" />;

}

// Render child routes if authenticated

return <Outlet />;

};

export default AdminRoutes;

Update your main router setup in src/main.jsx or wherever your router is defined:

jsx

Copy

import React from 'react';

import ReactDOM from 'react-dom/client';

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

import { AuthProvider } from './context/AuthContext';

import App from './App.jsx';

import LoginPage from './pages/admin/LoginPage';

import AdminRoutes from './routes/AdminRoutes';

import './index.css';

// Import a placeholder dashboard page (we'll implement this in later steps)

const AdminDashboard = () => <div>Admin Dashboard (Coming in Step 2)</div>;

ReactDOM.createRoot(document.getElementById('root')).render(

<React.StrictMode>

<BrowserRouter>

<AuthProvider>

<Routes>

{/\* Main app routes \*/}

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

{/\* Admin routes \*/}

<Route path="/admin/login" element={<LoginPage />} />

<Route element={<AdminRoutes />}>

<Route path="/admin/dashboard" element={<AdminDashboard />} />

{/\* More admin routes will be added in future steps \*/}

</Route>

</Routes>

</AuthProvider>

</BrowserRouter>

</React.StrictMode>

);

Git Save Point

bash

Copy

# Add new files

git add src/context/AuthContext.jsx

git add src/hooks/useAuth.js

git add src/pages/admin/LoginPage.jsx

git add src/routes/AdminRoutes.jsx

git add -p src/main.jsx # Review changes before committing

# Commit changes

git commit -m "Implement admin login page and authentication system"

Testing Login Functionality

Start your backend server:

bash

Copy

cd ../backend

npm run dev

In a new terminal, start your frontend:

bash

Copy

cd ../frontend

npm run dev

Open your browser and navigate to http://localhost:5173/admin/login

Try logging in with incorrect credentials - verify error messages appear

Try logging in with your admin credentials - you should be redirected to the dashboard placeholder

Test the "Remember Me" functionality:

Log in with "Remember Me" checked

Close the browser and reopen it

Navigate to /admin/dashboard - you should still be logged in

Log out and test again without "Remember Me" checked

Security Considerations

Ensure your JWT\_SECRET in .env is a strong, random string

Consider adding rate limiting specifically for login attempts

For production, set appropriate token expiration

Always use HTTPS in production (which you've already set up)

This completes step 1.1! Let me know when you're ready to proceed to step 1.2: JWT Authentication Setup.

Retry

TM

I am at this step: Add or update src/config/config.js

This is my config.js - can you show me where to update:

// Load environment variables from .env file

require("dotenv").config();

// Define and export configuration

const config = {

// Server configuration

port: process.env.PORT || 3000,

nodeEnv: process.env.NODE\_ENV || "development",

// CORS configuration

corsOrigin:

process.env.CORS\_ORIGIN || "https://subjectlinepro.trevormchaffie.com",

// Rate limiting

rateLimitWindowMs: parseInt(process.env.RATE\_LIMIT\_WINDOW\_MS) || 900000, // 15 minutes

rateLimitMax: parseInt(process.env.RATE\_LIMIT\_MAX) || 100, // 100 requests per window

// Data paths

leadsDataPath: process.env.LEADS\_DATA\_PATH || "./src/data/leads.json",

analyzedSubjectsPath:

process.env.ANALYZED\_SUBJECTS\_PATH || "./src/data/analyzed\_subjects.json",

// Admin credentials

adminUsername: process.env.ADMIN\_USERNAME || "admin",

adminPassword: process.env.ADMIN\_PASSWORD || "changeme",

};

// Simple validation to ensure critical values are set

function validateConfig() {

const requiredVars = ["port", "corsOrigin", "adminUsername", "adminPassword"];

for (const item of requiredVars) {

if (!config[item]) {

throw new Error(Required configuration ${item} is missing or empty);

}

}

// Warn if using default admin password

if (config.adminPassword === "changeme") {

console.warn(

"WARNING: Using default admin password. Change this in production!"

);

}

}

validateConfig();

module.exports = config;

Edit

Strategized JWT secret integration into configuration file.

3s

Here's how you should update your config.js file to include the JWT secret:

javascript

Copy

// Load environment variables from .env file

require("dotenv").config();

// Define and export configuration

const config = {

// Server configuration

port: process.env.PORT || 3000,

nodeEnv: process.env.NODE\_ENV || "development",

// CORS configuration

corsOrigin:

process.env.CORS\_ORIGIN || "https://subjectlinepro.trevormchaffie.com",

// Rate limiting

rateLimitWindowMs: parseInt(process.env.RATE\_LIMIT\_WINDOW\_MS) || 900000, // 15 minutes

rateLimitMax: parseInt(process.env.RATE\_LIMIT\_MAX) || 100, // 100 requests per window

// Data paths

leadsDataPath: process.env.LEADS\_DATA\_PATH || "./src/data/leads.json",

analyzedSubjectsPath:

process.env.ANALYZED\_SUBJECTS\_PATH || "./src/data/analyzed\_subjects.json",

// Admin credentials

adminUsername: process.env.ADMIN\_USERNAME || "admin",

adminPassword: process.env.ADMIN\_PASSWORD || "changeme",

// JWT Authentication (add this new section)

jwtSecret: process.env.JWT\_SECRET || "your-secret-key",

};

// Simple validation to ensure critical values are set

function validateConfig() {

const requiredVars = ["port", "corsOrigin", "adminUsername", "adminPassword", "jwtSecret"];

for (const item of requiredVars) {

if (!config[item]) {

throw new Error(`Required configuration ${item} is missing or empty`);

}

}

// Warn if using default admin password

if (config.adminPassword === "changeme") {

console.warn(

"WARNING: Using default admin password. Change this in production!"

);

}

// Warn if using default JWT secret

if (config.jwtSecret === "your-secret-key") {

console.warn(

"WARNING: Using default JWT secret. Change this in production!"

);

}

}

validateConfig();

module.exports = config;

Don't forget to also add the JWT\_SECRET to your .env file:

Copy

JWT\_SECRET=your\_secure\_random\_secret\_key

Make sure to replace your\_secure\_random\_secret\_key with a strong, randomly generated string for production use.

Retry

TM

What does this mean?

git add -p src/main.jsx

diff --git a/frontend/src/main.jsx b/frontend/src/main.jsx

index 52af6c6..43e3ff0 100644

--- a/frontend/src/main.jsx

+++ b/frontend/src/main.jsx

@@ -1,21 +1,31 @@

import React from "react";

import ReactDOM from "react-dom/client";

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

+import { AuthProvider } from "./context/AuthContext";

import App from "./App.jsx";

+import LoginPage from "./pages/admin/LoginPage";

+import AdminRoutes from "./routes/AdminRoutes";

import "./index.css";

-import ErrorBoundary from "./components/ErrorBoundary.jsx";

-import { registerServiceWorker } from "./registerSW.js";

-import syncService from "./services/syncService.js";

-// Register the service worker

-registerServiceWorker();

-

-// Initialize offline data synchronization

-syncService.init();

+// Import a placeholder dashboard page (we'll implement this in later steps)

+const AdminDashboard = () => <div>Admin Dashboard (Coming in Step 2)</div>;

ReactDOM.createRoot(document.getElementById("root")).render(

<React.StrictMode>

- <ErrorBoundary>

- <App />

- </ErrorBoundary>

+ <BrowserRouter>

+ <AuthProvider>

+ <Routes>

+ {/\* Main app routes /}

+ <Route path="/" element={<App />} />

+

+ {/ Admin routes /}

+ <Route path="/admin/login" element={<LoginPage />} />

+ <Route element={<AdminRoutes />}>

+ <Route path="/admin/dashboard" element={<AdminDashboard />} />

+ {/ More admin routes will be added in future steps \*/}

+ </Route>

+ </Routes>

+ </AuthProvider>

+ </BrowserRouter>

</React.StrictMode>

);

(1/1) Stage this hunk [y,n,q,a,d,s,e,p,?]?

Edit

Mapped routing strategy, preserving key app functionality and structure.

6s

This is Git's interactive patch mode, showing you the changes to be made to main.jsx. The output shows:

Lines with - are being removed

Lines with + are being added

The changes add React Router and authentication setup while removing the ErrorBoundary, service worker registration, and offline sync functionality.

Important note: This will remove your existing offline capabilities and error handling. You probably want to keep these features while adding the new admin functionality.

Instead of replacing everything, I recommend typing n to reject this hunk, and then manually combining both approaches:

jsx

Copy

import React from "react";

import ReactDOM from "react-dom/client";

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

import { AuthProvider } from "./context/AuthContext";

import App from "./App.jsx";

import LoginPage from "./pages/admin/LoginPage";

import AdminRoutes from "./routes/AdminRoutes";

import ErrorBoundary from "./components/ErrorBoundary.jsx";

import { registerServiceWorker } from "./registerSW.js";

import syncService from "./services/syncService.js";

import "./index.css";

// Import a placeholder dashboard page

const AdminDashboard = () => <div>Admin Dashboard (Coming in Step 2)</div>;

// Register the service worker

registerServiceWorker();

// Initialize offline data synchronization

syncService.init();

ReactDOM.createRoot(document.getElementById("root")).render(

<React.StrictMode>

<ErrorBoundary>

<BrowserRouter>

<AuthProvider>

<Routes>

{/\* Main app routes \*/}

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

{/\* Admin routes \*/}

<Route path="/admin/login" element={<LoginPage />} />

<Route element={<AdminRoutes />}>

<Route path="/admin/dashboard" element={<AdminDashboard />} />

{/\* More admin routes will be added in future steps \*/}

</Route>

</Routes>

</AuthProvider>

</BrowserRouter>

</ErrorBoundary>

</React.StrictMode>

);

This preserves your existing functionality while adding the new admin routes.

Retry

TM

Trying to login as admin. Not sure what the password is, Where can I find.set the admin password, JWT token as well - not sure if this affects anything.

I didn't change the placeholders in the code, could this be the issue?

Edit

Decoded login credentials and recommended secure configuration steps.

5s

Yes, not changing the placeholder values is the issue! Based on your config.js, the default values are being used.

Login Credentials

You can log in with:

Username: admin

Password: changeme

These are the default values from your config.js when not overridden by environment variables.

Setting Proper Values

For security, you should update these values in your .env file on the backend server:

Copy

ADMIN\_USERNAME=your\_secure\_username

ADMIN\_PASSWORD=your\_secure\_password

JWT\_SECRET=your\_long\_random\_string

For the JWT\_SECRET, use a strong random string. You can generate one with:

bash

Copy

node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"

After updating your .env file, restart your backend server for the changes to take effect:

bash

Copy

pm2 restart subject-line-pro

# or if you're running in development mode

npm run dev

Remember, using default placeholder values is fine for development, but make sure to set strong, secure values before deploying to production.

Retry

TM

Still can't login. PLease check my settings.

Config.js:

// Load environment variables from .env file

require("dotenv").config();

// Define and export configuration

const config = {

// Server configuration

port: process.env.PORT || 3000,

nodeEnv: process.env.NODE\_ENV || "development",

// CORS configuration

corsOrigin:

process.env.CORS\_ORIGIN || "https://subjectlinepro.trevormchaffie.com",

// Rate limiting

rateLimitWindowMs: parseInt(process.env.RATE\_LIMIT\_WINDOW\_MS) || 900000, // 15 minutes

rateLimitMax: parseInt(process.env.RATE\_LIMIT\_MAX) || 100, // 100 requests per window

// Data paths

leadsDataPath: process.env.LEADS\_DATA\_PATH || "./src/data/leads.json",

analyzedSubjectsPath:

process.env.ANALYZED\_SUBJECTS\_PATH || "./src/data/analyzed\_subjects.json",

// Admin credentials

adminUsername: process.env.ADMIN\_USERNAME || "admin",

adminPassword: process.env.ADMIN\_PASSWORD || "changeme",

// JWT Authentication (add this new section)

jwtSecret: process.env.JWT\_SECRET || "your-secret-key",

};

// Simple validation to ensure critical values are set

function validateConfig() {

const requiredVars = [

"port",

"corsOrigin",

"adminUsername",

"adminPassword",

"jwtSecret",

];

for (const item of requiredVars) {

if (!config[item]) {

throw new Error(Required configuration ${item} is missing or empty);

}

}

// Warn if using default admin password

if (config.adminPassword === "changeme") {

console.warn(

"WARNING: Using default admin password. Change this in production!"

);

}

// Warn if using default JWT secret

if (config.jwtSecret === "your-secret-key") {

console.warn(

"WARNING: Using default JWT secret. Change this in production!"

);

}

}

validateConfig();

module.exports = config;

.env:

PORT=3000

NODE\_ENV=development

CORS\_ORIGIN=http://localhost:5173

RATE\_LIMIT\_WINDOW\_MS=900000

RATE\_LIMIT\_MAX=100

ADMIN\_USERNAME=mr1018

ADMIN\_PASSWORD=Maya03112005

JWT\_SECRET=db9b93fda0bbeaff11e0c64555e735ba51e986b667edf1867697a1c4ac045e62

index.js:

// Import dependencies

const express = require("express");

const cors = require("cors");

const helmet = require("helmet");

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

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

// Import middleware

const errorHandler = require("./middleware/errorHandler");

const rateLimiter = require("./middleware/rateLimiter");

const requestLogger = require("./middleware/requestLogger");

// Import routes

const apiRoutes = require("./routes");

// Initialize Express app

const app = express();

// Configure middleware

app.use(helmet()); // Adds security HTTP headers

app.use(

cors({

origin: config.corsOrigin,

methods: ["GET", "POST"],

allowedHeaders: ["Content-Type", "Authorization"],

})

);

app.use(express.json()); // Parse JSON request body

app.use(express.urlencoded({ extended: true })); // Parse URL-encoded request body

app.use(requestLogger); // Log requests

app.use(rateLimiter); // Apply rate limiting

// Apply routes

app.use("/api", apiRoutes);

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

// Health check endpoint

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

res.status(200).json({ status: "UP", timestamp: new Date().toISOString() });

});

// Root route

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

res.status(200).json({

message: "Subject Line Pro API is running",

version: "1.0.0",

endpoints: ["/health", "/api/analyze", "/api/leads", "/api/stats"],

});

});

// Error handling middleware (should be last)

app.use(errorHandler);

// Start the server

const PORT = config.port;

app.listen(PORT, () => {

console.log(Server running in ${config.nodeEnv} mode on port ${PORT});

});

// Handle unhandled promise rejections

process.on("unhandledRejection", (err) => {

console.error("Unhandled Rejection:", err);

});

// Handle uncaught exceptions

process.on("uncaughtException", (err) => {

console.error("Uncaught Exception:", err);

process.exit(1);

});

Edit

Diagnosed login issues by scrutinizing authentication configuration.

9s

jsx

I see several potential issues. Let's debug step by step:

1. Frontend API URL Mismatch

In your AuthContext.jsx, the frontend is likely trying to call:

javascript

Copy

fetch('https://api.trevormchaffie.com/api/auth/login', ...)

But your .env file shows:

Copy

CORS\_ORIGIN=http://localhost:5173

This indicates you're running in development mode with the frontend on localhost, but the API calls might be going to the production server.

2. Let's Check the Auth Route Implementation

The authRoutes.js file might be missing or have issues. Make sure it looks like:

javascript

Copy

const express = require('express');

const jwt = require('jsonwebtoken');

const config = require('../config/config');

const router = express.Router();

// Login endpoint

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

try {

const { username, password } = req.body;

console.log('Login attempt:', { username }); // Add for debugging

// Check if username and password match admin credentials

if (username === config.adminUsername && password === config.adminPassword) {

// Generate JWT token

const token = jwt.sign(

{ id: 'admin', username },

config.jwtSecret,

{ expiresIn: '7d' }

);

// Return token and user info

return res.json({

success: true,

token,

user: { username }

});

}

console.log('Invalid credentials. Expected:', {

username: config.adminUsername,

passwordLength: config.adminPassword.length

});

// Return error for invalid credentials

return res.status(401).json({

success: false,

message: 'Invalid username or password'

});

} catch (error) {

console.error('Login error:', error);

return res.status(500).json({

success: false,

message: 'Server error during authentication'

});

}

});

// Export router

module.exports = router;

3. Add Console Logs for Debugging

Update your LoginPage.jsx to include more detailed error logging:

jsx

Copy

const handleSubmit = async (e) => {

e.preventDefault();

if (!username.trim() || !password.trim()) {

setLoginError('Username and password are required');

return;

}

setIsSubmitting(true);

setLoginError('');

try {

console.log('Attempting login with:', { username });

// Log the API URL being used

console.log('API URL:', 'https://api.trevormchaffie.com/api/auth/login');

const success = await login(username, password, rememberMe);

console.log('Login response:', success);

if (success) {

navigate('/admin/dashboard');

} else {

setLoginError(authError || 'Invalid credentials');

}

} catch (error) {

console.error('Login submission error:', error);

setLoginError('An unexpected error occurred');

} finally {

setIsSubmitting(false);

}

};

4. Check the Network Tab in DevTools

When attempting to login:

Open browser DevTools (F12)

Go to the Network tab

Try to login

Look for the request to /api/auth/login

Check:

Request URL

Request payload

Response status code

Response body

5. Quick Fix: Update API URL for Development

If you're running the frontend in development mode (localhost:5173), update your AuthContext to use a development URL:

jsx

Copy

// In src/context/AuthContext.jsx

const API\_URL = import.meta.env.MODE === 'development'

? 'http://localhost:3000/api/auth' // Development

: 'https://api.trevormchaffie.com/api/auth'; // Production

// Then update all fetch calls to use this base URL

const response = await fetch(`${API\_URL}/login`, {

method: 'POST',

// ...

});

Try these changes and let me know if you're still having issues. The most likely issue is that your frontend is trying to call the production API while you're testing in development mode.

Retry

TM

AuthContext.jsx:

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

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {

const [isAuthenticated, setIsAuthenticated] = useState(false);

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

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

const [error, setError] = useState(null);

// Check if user is already logged in on component mount

useEffect(() => {

const checkLoginStatus = async () => {

setLoading(true);

try {

// Check if token exists in localStorage

const token = localStorage.getItem("authToken");

if (!token) {

setIsAuthenticated(false);

setUser(null);

setLoading(false);

return;

}

// Verify token with backend

const response = await fetch(

"https://api.trevormchaffie.com/api/auth/verify",

{

method: "GET",

headers: {

Authorization: Bearer ${token},

},

}

);

if (response.ok) {

const data = await response.json();

setIsAuthenticated(true);

setUser(data.user);

} else {

// Token is invalid or expired

localStorage.removeItem("authToken");

setIsAuthenticated(false);

setUser(null);

}

} catch (err) {

console.error("Error verifying authentication:", err);

setError("Failed to verify authentication status");

setIsAuthenticated(false);

setUser(null);

} finally {

setLoading(false);

}

};

checkLoginStatus();

}, []);

// Login function

const login = async (username, password, rememberMe = false) => {

setLoading(true);

setError(null);

try {

const response = await fetch(

"https://api.trevormchaffie.com/api/auth/login",

{

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({ username, password }),

}

);

const data = await response.json();

if (response.ok) {

setIsAuthenticated(true);

setUser(data.user);

// Store token in localStorage if rememberMe is true, otherwise in sessionStorage

if (rememberMe) {

localStorage.setItem("authToken", data.token);

} else {

sessionStorage.setItem("authToken", data.token);

}

return true;

} else {

setError(data.message || "Invalid credentials");

return false;

}

} catch (err) {

console.error("Login error:", err);

setError("Failed to connect to authentication server");

return false;

} finally {

setLoading(false);

}

};

// Logout function

const logout = () => {

localStorage.removeItem("authToken");

sessionStorage.removeItem("authToken");

setIsAuthenticated(false);

setUser(null);

};

// Get auth token (from either localStorage or sessionStorage)

const getAuthToken = () => {

return (

localStorage.getItem("authToken") || sessionStorage.getItem("authToken")

);

};

return (

<AuthContext.Provider

value={{

isAuthenticated,

user,

loading,

error,

login,

logout,

getAuthToken,

}}

>

{children}

</AuthContext.Provider>

);

};

Is this the problem here?

Edit

Diagnosed hardcoded API URLs causing development environment issues.

5s

javascript

javascript

Yes, the problem is in your AuthContext.jsx file. You've hardcoded the API URL to https://api.trevormchaffie.com, but your backend is currently running locally during development.

Here's how to fix it:

First, create an API URL configuration that automatically adjusts based on the environment:

javascript

Copy

// At the top of AuthContext.jsx, add this:

const API\_BASE\_URL = window.location.hostname === 'localhost'

? 'http://localhost:3000/api/auth' // Development

: 'https://api.trevormchaffie.com/api/auth'; // Production

console.log('Using API URL:', API\_BASE\_URL);

Then update both fetch calls to use this base URL:

javascript

Copy

// Change this:

const response = await fetch(

"https://api.trevormchaffie.com/api/auth/verify",

{

// ...

}

);

// To this:

const response = await fetch(

`${API\_BASE\_URL}/verify`,

{

// ...

}

);

And:

javascript

Copy

// Change this:

const response = await fetch(

"https://api.trevormchaffie.com/api/auth/login",

{

// ...

}

);

// To this:

const response = await fetch(

`${API\_BASE\_URL}/login`,

{

// ...

}

);

This way, when you're running on localhost, it will call your local backend server, and when deployed to production, it will call your production API.

With these changes, your login should work when testing locally.

Retry

TM

Tests working, But how to logout after using remember me? No logout on the page

Edit

Engineered logout solution for admin dashboard with auth hook.

4s

jsx

You're right! Let's add a logout button to the admin dashboard placeholder. Since we haven't built the full dashboard yet, let's create a simple header with a logout button.

Create a new file for the admin dashboard:

bash

Copy

# Create a temporary dashboard component

touch src/pages/admin/DashboardPage.jsx

Add this code to src/pages/admin/DashboardPage.jsx:

jsx

Copy

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

import { useAuth } from '../../hooks/useAuth';

const DashboardPage = () => {

const { logout, user } = useAuth();

const navigate = useNavigate();

const handleLogout = () => {

logout();

navigate('/admin/login');

};

return (

<div className="min-h-screen bg-gray-100">

{/\* Admin Header \*/}

<header className="bg-white shadow-md">

<div className="container mx-auto px-4 py-4 flex justify-between items-center">

<div className="flex items-center">

<h1 className="text-xl font-bold text-primary">Subject Line Pro</h1>

<span className="ml-2 bg-gray-200 px-2 py-1 rounded text-xs text-gray-700">Admin</span>

</div>

<div className="flex items-center">

<span className="mr-4 text-sm text-gray-600">

Logged in as <span className="font-medium">{user?.username || 'Admin'}</span>

</span>

<button

onClick={handleLogout}

className="bg-gray-200 text-gray-700 hover:bg-gray-300 px-4 py-2 rounded transition-colors"

>

Logout

</button>

</div>

</div>

</header>

{/\* Dashboard Content Placeholder \*/}

<main className="container mx-auto px-4 py-8">

<div className="bg-white rounded-lg shadow-md p-6">

<h2 className="text-xl font-semibold mb-4">Admin Dashboard</h2>

<p className="text-gray-600">

Full dashboard coming in Step 2. You are now successfully logged in to the admin area.

</p>

</div>

</main>

</div>

);

};

export default DashboardPage;

Now update your main.jsx to use this component instead of the placeholder:

jsx

Copy

// Find this line

const AdminDashboard = () => <div>Admin Dashboard (Coming in Step 2)</div>;

// Replace it with

import DashboardPage from './pages/admin/DashboardPage';

// Then update the route

<Route path="/admin/dashboard" element={<DashboardPage />} />

This gives you a functional logout button and a more polished admin dashboard placeholder. The logout button will clear your authentication tokens and redirect you back to the login page.

Don't forget to commit these changes:

bash

Copy

git add src/pages/admin/DashboardPage.jsx

git add -p src/main.jsx

git commit -m "Add logout functionality to admin dashboard"

Retry

TM

git add -p src/main.jsx

diff --git a/frontend/src/main.jsx b/frontend/src/main.jsx

index 4bfc277..6fdfe5b 100644

--- a/frontend/src/main.jsx

+++ b/frontend/src/main.jsx

@@ -11,7 +11,7 @@ import syncService from "./services/syncService.js";

import "./index.css";

// Import a placeholder dashboard page

-const AdminDashboard = () => <div>Admin Dashboard (Coming in Step 2)</div>;

+import DashboardPage from "./pages/admin/DashboardPage";

// Register the service worker

registerServiceWorker();

(1/2) Stage this hunk [y,n,q,a,d,j,J,g,/,e,p,?]?

Edit

Thinking about staging specific code changes in Git repository

2s

Type y to accept this hunk, which replaces the placeholder dashboard component with the import of your new DashboardPage component.

This is the first part of the change, and it looks correct. After accepting this hunk, git will show you the second part of the change which should update the Route element.

Retry

TM

@@ -31,7 +31,7 @@ ReactDOM.createRoot(document.getElementById("root")).render(

{/\* Admin routes /}

<Route path="/admin/login" element={<LoginPage />} />

<Route element={<AdminRoutes />}>

- <Route path="/admin/dashboard" element={<AdminDashboard />} />

+ <Route path="/admin/dashboard" element={<DashboardPage />} />

{/ More admin routes will be added in future steps \*/}

</Route>

</Routes>

(2/2) Stage this hunk [y,n,q,a,d,K,g,/,e,p,?]?

Edit

Replacing AdminDashboard with new DashboardPage component

1s

Type y to accept this hunk as well.

This change updates the Route component to use your new DashboardPage component instead of the placeholder AdminDashboard. This is exactly what we want - it will display your new dashboard with the logout button when you navigate to /admin/dashboard.

After staging both hunks, you can complete the commit with:

bash

Copy

git commit -m "Add logout functionality to admin dashboard"

Retry

TM

Ready for step 1.2. PLease provide clean, optimized and secure code.

Edit

Strategized JWT authentication improvements for secure token management.

5s

Step 1.2: JWT Authentication Setup

Now that we have basic authentication working, let's enhance the JWT implementation for better security and user experience. We'll:

Improve the token structure and expiration

Implement token refresh functionality

Add better security measures

Create an authentication middleware for protected routes

1. Backend JWT Enhancements

First, let's create an auth middleware file for backend route protection:

bash

Copy

# Create auth middleware file

touch backend/src/middleware/authMiddleware.js

Add this code to authMiddleware.js:

javascript

Copy

const jwt = require('jsonwebtoken');

const config = require('../config/config');

/\*\*

\* Middleware to authenticate JWT tokens

\* Adds user information to request object if valid

\*/

const authenticateJWT = (req, res, next) => {

// Get auth header

const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith('Bearer ')) {

return res.status(401).json({

success: false,

message: 'Access denied. No token provided.'

});

}

try {

// Extract token from header

const token = authHeader.split(' ')[1];

// Verify token

const decoded = jwt.verify(token, config.jwtSecret);

// Add user info to request

req.user = decoded;

// Continue to the next middleware/route handler

next();

} catch (error) {

if (error.name === 'TokenExpiredError') {

return res.status(401).json({

success: false,

message: 'Token expired',

expired: true

});

}

return res.status(403).json({

success: false,

message: 'Invalid token'

});

}

};

module.exports = { authenticateJWT };

2. Enhance Auth Routes

Now let's update our authentication routes to add refresh token functionality:

bash

Copy

# Update auth routes file

touch backend/src/routes/authRoutes.js

Replace the contents of authRoutes.js with:

javascript

Copy

const express = require('express');

const jwt = require('jsonwebtoken');

const crypto = require('crypto');

const config = require('../config/config');

const { authenticateJWT } = require('../middleware/authMiddleware');

const router = express.Router();

// Store for refresh tokens (in-memory for simplicity; use a database in production)

const refreshTokens = new Set();

/\*\*

\* Generate access token

\*/

const generateAccessToken = (user) => {

return jwt.sign(

{ id: user.id, username: user.username },

config.jwtSecret,

{ expiresIn: '1h' } // Short-lived token

);

};

/\*\*

\* Generate refresh token

\*/

const generateRefreshToken = (user) => {

const refreshToken = crypto.randomBytes(40).toString('hex');

refreshTokens.add(refreshToken);

return refreshToken;

};

/\*\*

\* Login endpoint

\*/

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

try {

const { username, password } = req.body;

// Check if username and password match admin credentials

if (username === config.adminUsername && password === config.adminPassword) {

const user = { id: 'admin', username };

// Generate tokens

const accessToken = generateAccessToken(user);

const refreshToken = generateRefreshToken(user);

// Calculate token expiration for client reference

const now = Math.floor(Date.now() / 1000);

const expiresAt = now + (60 \* 60); // 1 hour from now

// Return tokens and user info

return res.json({

success: true,

token: accessToken,

refreshToken,

expiresAt,

user

});

}

// Return error for invalid credentials

return res.status(401).json({

success: false,

message: 'Invalid username or password'

});

} catch (error) {

console.error('Login error:', error);

return res.status(500).json({

success: false,

message: 'Server error during authentication'

});

}

});

/\*\*

\* Token refresh endpoint

\*/

router.post('/refresh', async (req, res) => {

try {

const { refreshToken } = req.body;

// Check if refresh token exists

if (!refreshToken || !refreshTokens.has(refreshToken)) {

return res.status(401).json({

success: false,

message: 'Invalid refresh token'

});

}

try {

// Verify existing access token to get user data

const oldToken = req.body.token;

const decoded = jwt.decode(oldToken);

if (!decoded || !decoded.username) {

throw new Error('Invalid token format');

}

const user = { id: decoded.id, username: decoded.username };

// Generate new access token

const accessToken = generateAccessToken(user);

// Calculate token expiration for client reference

const now = Math.floor(Date.now() / 1000);

const expiresAt = now + (60 \* 60); // 1 hour from now

// Return new access token

return res.json({

success: true,

token: accessToken,

expiresAt,

user

});

} catch (error) {

console.error('Token decode error:', error);

return res.status(401).json({

success: false,

message: 'Invalid token format'

});

}

} catch (error) {

console.error('Refresh error:', error);

return res.status(500).json({

success: false,

message: 'Server error during token refresh'

});

}

});

/\*\*

\* Verify token endpoint

\*/

router.get('/verify', authenticateJWT, async (req, res) => {

// If middleware passes, token is valid

return res.json({

success: true,

user: { username: req.user.username }

});

});

/\*\*

\* Logout endpoint

\*/

router.post('/logout', async (req, res) => {

try {

const { refreshToken } = req.body;

// Remove refresh token from valid tokens

if (refreshToken) {

refreshTokens.delete(refreshToken);

}

return res.json({

success: true,

message: 'Logged out successfully'

});

} catch (error) {

console.error('Logout error:', error);

return res.status(500).json({

success: false,

message: 'Server error during logout'

});

}

});

module.exports = router;

3. Update Frontend Authentication Context

Now let's enhance the frontend Authentication Context to handle token refresh:

javascript

Copy

import { createContext, useState, useEffect, useCallback } from "react";

// Get API base URL based on environment

const API\_BASE\_URL = window.location.hostname === 'localhost'

? 'http://localhost:3000/api/auth' // Development

: 'https://api.trevormchaffie.com/api/auth'; // Production

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {

const [isAuthenticated, setIsAuthenticated] = useState(false);

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

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

const [error, setError] = useState(null);

const [refreshingToken, setRefreshingToken] = useState(false);

// Function to get the access token

const getAccessToken = useCallback(() => {

return localStorage.getItem("authToken") || sessionStorage.getItem("authToken");

}, []);

// Function to get the refresh token

const getRefreshToken = useCallback(() => {

return localStorage.getItem("refreshToken") || sessionStorage.getItem("refreshToken");

}, []);

// Function to save tokens

const saveTokens = useCallback((accessToken, refreshToken, expiresAt, rememberMe) => {

const storage = rememberMe ? localStorage : sessionStorage;

if (accessToken) storage.setItem("authToken", accessToken);

if (refreshToken) storage.setItem("refreshToken", refreshToken);

if (expiresAt) storage.setItem("tokenExpiresAt", expiresAt.toString());

}, []);

// Function to clear tokens

const clearTokens = useCallback(() => {

localStorage.removeItem("authToken");

localStorage.removeItem("refreshToken");

localStorage.removeItem("tokenExpiresAt");

sessionStorage.removeItem("authToken");

sessionStorage.removeItem("refreshToken");

sessionStorage.removeItem("tokenExpiresAt");

}, []);

// Refresh the access token

const refreshToken = useCallback(async () => {

setRefreshingToken(true);

try {

const refreshToken = getRefreshToken();

const currentToken = getAccessToken();

if (!refreshToken || !currentToken) {

throw new Error("No refresh token available");

}

const response = await fetch(`${API\_BASE\_URL}/refresh`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

refreshToken,

token: currentToken

}),

});

const data = await response.json();

if (response.ok) {

// Determine storage type based on where refresh token is stored

const useLocalStorage = !!localStorage.getItem("refreshToken");

// Save the new access token

saveTokens(data.token, null, data.expiresAt, useLocalStorage);

return true;

} else {

throw new Error(data.message || "Token refresh failed");

}

} catch (error) {

console.error("Token refresh error:", error);

clearTokens();

setIsAuthenticated(false);

setUser(null);

return false;

} finally {

setRefreshingToken(false);

}

}, [getRefreshToken, getAccessToken, saveTokens, clearTokens]);

// Setup token refresh timer

useEffect(() => {

if (!isAuthenticated) return;

// Get token expiration time

const expiresAt = localStorage.getItem("tokenExpiresAt") || sessionStorage.getItem("tokenExpiresAt");

if (!expiresAt) return;

// Convert to milliseconds and subtract current time

const expiresAtMs = parseInt(expiresAt) \* 1000;

const nowMs = Date.now();

// If already expired, refresh immediately

if (expiresAtMs <= nowMs) {

refreshToken();

return;

}

// Set timer to refresh 5 minutes before expiration

const timeUntilRefresh = expiresAtMs - nowMs - (5 \* 60 \* 1000);

// Setup timer

const refreshTimer = setTimeout(() => {

refreshToken();

}, timeUntilRefresh);

// Cleanup timer on unmount

return () => clearTimeout(refreshTimer);

}, [isAuthenticated, refreshToken]);

// Check if user is already logged in on component mount

useEffect(() => {

const checkLoginStatus = async () => {

setLoading(true);

try {

// Check if token exists

const token = getAccessToken();

if (!token) {

setIsAuthenticated(false);

setUser(null);

setLoading(false);

return;

}

// Verify token with backend

const response = await fetch(`${API\_BASE\_URL}/verify`, {

method: "GET",

headers: {

Authorization: `Bearer ${token}`,

},

});

if (response.ok) {

const data = await response.json();

setIsAuthenticated(true);

setUser(data.user);

} else {

// Token is invalid or expired, try to refresh it

const refreshSuccessful = await refreshToken();

if (!refreshSuccessful) {

clearTokens();

setIsAuthenticated(false);

setUser(null);

}

}

} catch (err) {

console.error("Error verifying authentication:", err);

setError("Failed to verify authentication status");

setIsAuthenticated(false);

setUser(null);

} finally {

setLoading(false);

}

};

checkLoginStatus();

}, [clearTokens, getAccessToken, refreshToken]);

// Login function

const login = async (username, password, rememberMe = false) => {

setLoading(true);

setError(null);

try {

const response = await fetch(`${API\_BASE\_URL}/login`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({ username, password }),

});

const data = await response.json();

if (response.ok) {

setIsAuthenticated(true);

setUser(data.user);

// Store tokens

saveTokens(data.token, data.refreshToken, data.expiresAt, rememberMe);

return true;

} else {

setError(data.message || "Invalid credentials");

return false;

}

} catch (err) {

console.error("Login error:", err);

setError("Failed to connect to authentication server");

return false;

} finally {

setLoading(false);

}

};

// Logout function

const logout = async () => {

const refreshToken = getRefreshToken();

if (refreshToken) {

try {

await fetch(`${API\_BASE\_URL}/logout`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({ refreshToken }),

});

} catch (error) {

console.error("Logout API error:", error);

}

}

clearTokens();

setIsAuthenticated(false);

setUser(null);

};

return (

<AuthContext.Provider

value={{

isAuthenticated,

user,

loading,

error,

login,

logout,

getAccessToken,

refreshToken,

refreshingToken

}}

>

{children}

</AuthContext.Provider>

);

};

4. Create a Protected API Request Helper

Let's create a helper function to make authenticated API requests:

bash

Copy

touch frontend/src/utils/apiUtils.js

Add this code to apiUtils.js:

javascript

Copy

import { useAuth } from '../hooks/useAuth';

/\*\*

\* Custom hook for making authenticated API requests

\* Handles token refresh when needed

\*/

export const useAuthenticatedRequest = () => {

const { getAccessToken, refreshToken, isAuthenticated } = useAuth();

/\*\*

\* Make an authenticated API request

\* @param {string} url - API endpoint

\* @param {Object} options - Fetch options

\* @param {boolean} requiresAuth - Whether the request requires authentication

\* @returns {Promise<Object>} - Response data

\*/

const fetchWithAuth = async (url, options = {}, requiresAuth = true) => {

// Don't try to authenticate if user isn't logged in and endpoint requires auth

if (requiresAuth && !isAuthenticated) {

throw new Error('Authentication required');

}

try {

// Add authentication header if needed

const headers = options.headers || {};

if (requiresAuth) {

const token = getAccessToken();

if (!token) {

throw new Error('No authentication token available');

}

headers.Authorization = `Bearer ${token}`;

}

// Make the request

const response = await fetch(url, {

...options,

headers: {

'Content-Type': 'application/json',

...headers,

},

});

// If unauthorized, try to refresh the token and retry once

if (response.status === 401 && requiresAuth) {

const refreshSuccessful = await refreshToken();

if (refreshSuccessful) {

// Get the new token and retry the request

const newToken = getAccessToken();

const retryResponse = await fetch(url, {

...options,

headers: {

'Content-Type': 'application/json',

...headers,

Authorization: `Bearer ${newToken}`,

},

});

return handleResponse(retryResponse);

} else {

throw new Error('Session expired. Please log in again.');

}

}

return handleResponse(response);

} catch (error) {

console.error('API request error:', error);

throw error;

}

};

/\*\*

\* Handle API response

\* @param {Response} response - Fetch response object

\* @returns {Promise<Object>} - Parsed response data

\*/

const handleResponse = async (response) => {

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'API request failed');

}

return data;

};

return { fetchWithAuth };

};

5. Update Dependencies

Make sure to install the required dependencies if not already installed:

bash

Copy

# Navigate to backend directory

cd backend

# Install the required packages

npm install jsonwebtoken

# Return to frontend directory

cd ../frontend

Git Save Point

bash

Copy

# Add backend files

git add backend/src/middleware/authMiddleware.js

git add backend/src/routes/authRoutes.js

# Add frontend files

git add frontend/src/context/AuthContext.jsx

git add frontend/src/utils/apiUtils.js

# Commit changes

git commit -m "Implement JWT Authentication with token refresh"

Testing the JWT Authentication

Test the Login and Token Refresh:

Login with your admin credentials

Check the browser's local storage or session storage to see the tokens

The token will automatically refresh 5 minutes before expiration

If you want to test refresh manually, you can call refreshToken() from the console

Test Protected Routes:

Try accessing /admin/dashboard directly without logging in

You should be redirected to the login page

After logging in, you should be able to access the dashboard

Test Logout:

Click the logout button

Verify you're redirected to the login page

Try accessing /admin/dashboard again - you should be redirected back to login

This completes Step 1.2! Your JWT authentication system now includes:

Secure token generation and validation

Automatic token refresh

Protection against expired sessions

Proper logout handling

Would you like to move on to Step 1.3: Protected Routes Configuration?

Retry

TM

Help with test 1 - How to check browser local storage, sessions storage. Best way to test this step?

Edit

Mapped out detailed steps for testing JWT token storage.

2s

How to Test the JWT Authentication Implementation

Checking Browser Storage (LocalStorage & SessionStorage)

Method 1: Using Browser DevTools

Open your admin login page (/admin/login)

Right-click anywhere on the page and select "Inspect" or press F12

Navigate to the "Application" tab (Chrome/Edge) or "Storage" tab (Firefox)

In the left sidebar:

Expand "Local Storage" and click on your domain

Expand "Session Storage" and click on your domain

You should see these key items after logging in:

authToken - Your JWT access token

refreshToken - Token used to get new access tokens

tokenExpiresAt - Unix timestamp when the token expires

Show Image

Method 2: Using Console Commands

You can also check storage using JavaScript commands in the console:

Open DevTools (F12)

Go to the "Console" tab

Enter these commands:

javascript

Copy

// Check LocalStorage

console.log('LocalStorage tokens:', {

authToken: localStorage.getItem('authToken'),

refreshToken: localStorage.getItem('refreshToken'),

expiresAt: localStorage.getItem('tokenExpiresAt')

});

// Check SessionStorage

console.log('SessionStorage tokens:', {

authToken: sessionStorage.getItem('authToken'),

refreshToken: sessionStorage.getItem('refreshToken'),

expiresAt: sessionStorage.getItem('tokenExpiresAt')

});

Testing Token Refresh

Method 1: Manual Refresh

Test token refresh manually by running this command in the console:

javascript

Copy

const authContext = document.querySelector('body').\_\_REACT\_CONTEXT\_DEVTOOL.ReactContext.values.find(

ctx => ctx.currentValue && ctx.currentValue.isAuthenticated !== undefined

).currentValue;

// Call the refresh function

authContext.refreshToken().then(success => {

console.log('Token refresh ' + (success ? 'successful' : 'failed'));

});

Method 2: Wait for Auto Refresh

Since tokens are set to expire in 1 hour and auto-refresh 5 minutes before expiration:

Login with "Remember Me" checked

Look at tokenExpiresAt value in storage

Convert it to a readable date:

javascript

Copy

new Date(localStorage.getItem('tokenExpiresAt') \* 1000).toLocaleString()

Wait until 5 minutes before that time

Watch the network tab - you should see a request to /api/auth/refresh

The authToken value should change after refresh

Testing "Remember Me" Functionality

Test 1: With "Remember Me" Checked

Log in with the "Remember Me" checkbox checked

Verify tokens are in localStorage

Close the browser completely

Reopen and navigate to /admin/dashboard

You should still be logged in

Test 2: Without "Remember Me" Checked

Log out

Log in again without checking "Remember Me"

Verify tokens are in sessionStorage (not localStorage)

Close the browser completely

Reopen and navigate to /admin/dashboard

You should be redirected to the login page

Comprehensive Testing Plan

Login Test: Check credentials work

Storage Test: Verify tokens exist in the right storage

Protected Route Test: Try accessing dashboard without logging in (should redirect)

Auto Refresh Test: Wait until near-expiration to see auto-refresh

Manual Refresh Test: Use console to trigger refresh

Session Persistence Test: Test browser close/reopen with both "Remember Me" options

Logout Test: Verify tokens are removed and routes are protected after logout

This comprehensive testing approach verifies all aspects of your JWT authentication implementation.