0auth admin login using mongodb and bcryptjs i need frontend and backend code with step by step guide

0Auth Admin Login with MongoDB and bcryptjs

This guide will walk you through creating a secure admin login system using MongoDB for data storage and bcryptjs for password hashing. We'll build both the backend (Node.js/Express) and frontend (React) components.

Table of Contents

1. Backend Setup
2. Frontend Setup
3. Connecting Frontend to Backend
4. Testing the System

Backend Setup

Step 1: Initialize the Node.js project

bash

Copy

Download

mkdir admin-auth-backend

cd admin-auth-backend

npm init -y

Step 2: Install required dependencies

bash

Copy

Download

npm install express mongoose bcryptjs jsonwebtoken cors dotenv

Step 3: Create the backend structure

Copy

Download

admin-auth-backend/

├── .env

├── app.js

├── models/

│ └── Admin.js

├── controllers/

│ └── authController.js

├── routes/

│ └── authRoutes.js

└── config/

└── db.js

Step 4: Set up environment variables (.env)

env

Copy

Download

MONGODB\_URI=mongodb://localhost:27017/admin\_auth

JWT\_SECRET=your\_jwt\_secret\_key

PORT=5000

Step 5: Configure database connection (config/db.js)

javascript

Copy

Download

const mongoose = require('mongoose');

require('dotenv').config();

const connectDB = async () => {

try {

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

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log('MongoDB connected successfully');

} catch (error) {

console.error('MongoDB connection error:', error);

process.exit(1);

}

};

module.exports = connectDB;

Step 6: Create Admin model (models/Admin.js)

javascript

Copy

Download

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const AdminSchema = new mongoose.Schema({

username: {

type: String,

required: true,

unique: true,

trim: true,

lowercase: true

},

password: {

type: String,

required: true,

minlength: 6

},

createdAt: {

type: Date,

default: Date.now

}

});

*// Hash password before saving*

AdminSchema.pre('save', async function(next) {

if (!this.isModified('password')) return next();

try {

const salt = await bcrypt.genSalt(10);

this.password = await bcrypt.hash(this.password, salt);

next();

} catch (error) {

next(error);

}

});

*// Method to compare passwords*

AdminSchema.methods.comparePassword = async function(candidatePassword) {

return await bcrypt.compare(candidatePassword, this.password);

};

module.exports = mongoose.model('Admin', AdminSchema);

Step 7: Create auth controller (controllers/authController.js)

javascript

Copy

Download

const Admin = require('../models/Admin');

const jwt = require('jsonwebtoken');

*// Register a new admin*

exports.register = async (req, res) => {

try {

const { username, password } = req.body;

*// Check if admin already exists*

const existingAdmin = await Admin.findOne({ username });

if (existingAdmin) {

return res.status(400).json({ message: 'Admin already exists' });

}

*// Create new admin*

const admin = new Admin({ username, password });

await admin.save();

*// Create token*

const token = jwt.sign({ id: admin.\_id }, process.env.JWT\_SECRET, {

expiresIn: '1h'

});

res.status(201).json({ token, admin: { id: admin.\_id, username: admin.username } });

} catch (error) {

res.status(500).json({ message: error.message });

}

};

*// Login admin*

exports.login = async (req, res) => {

try {

const { username, password } = req.body;

*// Find admin*

const admin = await Admin.findOne({ username });

if (!admin) {

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

}

*// Check password*

const isMatch = await admin.comparePassword(password);

if (!isMatch) {

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

}

*// Create token*

const token = jwt.sign({ id: admin.\_id }, process.env.JWT\_SECRET, {

expiresIn: '1h'

});

res.json({ token, admin: { id: admin.\_id, username: admin.username } });

} catch (error) {

res.status(500).json({ message: error.message });

}

};

*// Get current admin (protected route)*

exports.getCurrentAdmin = async (req, res) => {

try {

const admin = await Admin.findById(req.admin.id).select('-password');

res.json(admin);

} catch (error) {

res.status(500).json({ message: error.message });

}

};

Step 8: Create auth routes (routes/authRoutes.js)

javascript

Copy

Download

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

const authMiddleware = require('../middleware/authMiddleware');

*// Public routes*

router.post('/register', authController.register);

router.post('/login', authController.login);

*// Protected route*

router.get('/admin', authMiddleware, authController.getCurrentAdmin);

module.exports = router;

Step 9: Create auth middleware (middleware/authMiddleware.js)

javascript

Copy

Download

const jwt = require('jsonwebtoken');

const Admin = require('../models/Admin');

const authMiddleware = async (req, res, next) => {

try {

*// Get token from header*

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ message: 'No token, authorization denied' });

}

*// Verify token*

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const admin = await Admin.findById(decoded.id).select('-password');

if (!admin) {

return res.status(401).json({ message: 'Token is not valid' });

}

req.admin = admin;

next();

} catch (error) {

res.status(401).json({ message: 'Token is not valid' });

}

};

module.exports = authMiddleware;

Step 10: Create the main app file (app.js)

javascript

Copy

Download

const express = require('express');

const cors = require('cors');

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

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

require('dotenv').config();

const app = express();

*// Connect to database*

connectDB();

*// Middleware*

app.use(cors());

app.use(express.json());

*// Routes*

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

*// Error handling*

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({ message: 'Something went wrong!' });

});

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

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

Step 11: Start the backend server

bash

Copy

Download

node app.js

Frontend Setup

Step 1: Initialize the React project

bash

Copy

Download

npx create-react-app admin-auth-frontend

cd admin-auth-frontend

Step 2: Install required dependencies

bash

Copy

Download

npm install axios react-router-dom @mui/material @emotion/react @emotion/styled @mui/icons-material

Step 3: Create the frontend structure

Copy

Download

admin-auth-frontend/

├── src/

│ ├── components/

│ │ ├── Auth/

│ │ │ ├── Login.jsx

│ │ │ ├── Register.jsx

│ │ │ └── PrivateRoute.jsx

│ │ └── Dashboard.jsx

│ ├── context/

│ │ └── AuthContext.js

│ ├── services/

│ │ └── authService.js

│ ├── App.js

│ ├── index.js

│ └── styles.css

Step 4: Create Auth Context (src/context/AuthContext.js)

javascript

Copy

Download

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

import axios from 'axios';

const AuthContext = createContext();

const AuthProvider = ({ children }) => {

const [admin, setAdmin] = useState(null);

const [token, setToken] = useState(localStorage.getItem('token'));

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

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

*// Check for token and load admin*

useEffect(() => {

if (token) {

axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;

loadAdmin();

} else {

setLoading(false);

}

}, [token]);

const loadAdmin = async () => {

try {

const res = await axios.get('http://localhost:5000/api/auth/admin');

setAdmin(res.data);

setIsAuthenticated(true);

} catch (err) {

console.error(err);

logout();

} finally {

setLoading(false);

}

};

const register = async (formData) => {

try {

const res = await axios.post('http://localhost:5000/api/auth/register', formData);

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

setToken(res.data.token);

setAdmin(res.data.admin);

setIsAuthenticated(true);

return { success: true };

} catch (err) {

return { success: false, message: err.response?.data?.message || 'Registration failed' };

}

};

const login = async (formData) => {

try {

const res = await axios.post('http://localhost:5000/api/auth/login', formData);

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

setToken(res.data.token);

setAdmin(res.data.admin);

setIsAuthenticated(true);

return { success: true };

} catch (err) {

return { success: false, message: err.response?.data?.message || 'Login failed' };

}

};

const logout = () => {

localStorage.removeItem('token');

delete axios.defaults.headers.common['Authorization'];

setAdmin(null);

setToken(null);

setIsAuthenticated(false);

};

return (

<AuthContext.Provider

value={{

admin,

token,

isAuthenticated,

loading,

register,

login,

logout

}}

>

{children}

</AuthContext.Provider>

);

};

export { AuthContext, AuthProvider };

Step 5: Create auth service (src/services/authService.js)

javascript

Copy

Download

import axios from 'axios';

const API\_URL = 'http://localhost:5000/api/auth';

*// Register admin*

const register = async (adminData) => {

const response = await axios.post(`${API\_URL}/register`, adminData);

if (response.data.token) {

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

}

return response.data;

};

*// Login admin*

const login = async (adminData) => {

const response = await axios.post(`${API\_URL}/login`, adminData);

if (response.data.token) {

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

}

return response.data;

};

*// Logout admin*

const logout = () => {

localStorage.removeItem('token');

};

*// Get current admin*

const getCurrentAdmin = async (token) => {

const config = {

headers: {

Authorization: `Bearer ${token}`,

},

};

const response = await axios.get(`${API\_URL}/admin`, config);

return response.data;

};

const authService = {

register,

login,

logout,

getCurrentAdmin,

};

export default authService;

Step 6: Create PrivateRoute component (src/components/Auth/PrivateRoute.jsx)

javascript

Copy

Download

import React, { useContext } from 'react';

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

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

const PrivateRoute = () => {

const { isAuthenticated, loading } = useContext(AuthContext);

if (loading) {

return <div>Loading...</div>;

}

return isAuthenticated ? <Outlet /> : <Navigate to="/login" />;

};

export default PrivateRoute;

Step 7: Create Login component (src/components/Auth/Login.jsx)

javascript

Copy

Download

import React, { useState, useContext } from 'react';

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

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

import { TextField, Button, Container, Typography, Box } from '@mui/material';

const Login = () => {

const [formData, setFormData] = useState({

username: '',

password: '',

});

const { login } = useContext(AuthContext);

const navigate = useNavigate();

const handleChange = (e) => {

setFormData({ ...formData, [e.target.name]: e.target.value });

};

const handleSubmit = async (e) => {

e.preventDefault();

const result = await login(formData);

if (result.success) {

navigate('/dashboard');

} else {

alert(result.message);

}

};

return (

<Container maxWidth="xs">

<Box sx={{ mt: 8, display: 'flex', flexDirection: 'column', alignItems: 'center' }}>

<Typography component="h1" variant="h5">

Admin Login

</Typography>

<Box component="form" onSubmit={handleSubmit} sx={{ mt: 1 }}>

<TextField

margin="normal"

required

fullWidth

label="Username"

name="username"

autoComplete="username"

autoFocus

value={formData.username}

onChange={handleChange}

/>

<TextField

margin="normal"

required

fullWidth

name="password"

label="Password"

type="password"

autoComplete="current-password"

value={formData.password}

onChange={handleChange}

/>

<Button

type="submit"

fullWidth

variant="contained"

sx={{ mt: 3, mb: 2 }}

>

Sign In

</Button>

</Box>

</Box>

</Container>

);

};

export default Login;

Step 8: Create Register component (src/components/Auth/Register.jsx)

javascript

Copy

Download

import React, { useState, useContext } from 'react';

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

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

import { TextField, Button, Container, Typography, Box } from '@mui/material';

const Register = () => {

const [formData, setFormData] = useState({

username: '',

password: '',

});

const { register } = useContext(AuthContext);

const navigate = useNavigate();

const handleChange = (e) => {

setFormData({ ...formData, [e.target.name]: e.target.value });

};

const handleSubmit = async (e) => {

e.preventDefault();

const result = await register(formData);

if (result.success) {

navigate('/dashboard');

} else {

alert(result.message);

}

};

return (

<Container maxWidth="xs">

<Box sx={{ mt: 8, display: 'flex', flexDirection: 'column', alignItems: 'center' }}>

<Typography component="h1" variant="h5">

Admin Registration

</Typography>

<Box component="form" onSubmit={handleSubmit} sx={{ mt: 1 }}>

<TextField

margin="normal"

required

fullWidth

label="Username"

name="username"

autoComplete="username"

autoFocus

value={formData.username}

onChange={handleChange}

/>

<TextField

margin="normal"

required

fullWidth

name="password"

label="Password"

type="password"

autoComplete="new-password"

value={formData.password}

onChange={handleChange}

/>

<Button

type="submit"

fullWidth

variant="contained"

sx={{ mt: 3, mb: 2 }}

>

Register

</Button>

</Box>

</Box>

</Container>

);

};

export default Register;

Step 9: Create Dashboard component (src/components/Dashboard.jsx)

javascript

Copy

Download

import React, { useContext } from 'react';

import { Button, Container, Typography, Box } from '@mui/material';

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

const Dashboard = () => {

const { admin, logout } = useContext(AuthContext);

return (

<Container maxWidth="md">

<Box sx={{ mt: 4 }}>

<Typography variant="h4" component="h1" gutterBottom>

Admin Dashboard

</Typography>

{admin && (

<Typography variant="body1">

Welcome, {admin.username}!

</Typography>

)}

<Box sx={{ mt: 2 }}>

<Button variant="contained" color="error" onClick={logout}>

Logout

</Button>

</Box>

</Box>

</Container>

);

};

export default Dashboard;

Step 10: Update App.js (src/App.js)

javascript

Copy

Download

import React from 'react';

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

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

import PrivateRoute from './components/Auth/PrivateRoute';

import Login from './components/Auth/Login';

import Register from './components/Auth/Register';

import Dashboard from './components/Dashboard';

function App() {

return (

<AuthProvider>

<Router>

<Routes>

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

<Route path="/register" element={<Register />} />

<Route element={<PrivateRoute />}>

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

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

</Route>

</Routes>

</Router>

</AuthProvider>

);

}

export default App;

Step 11: Update index.js (src/index.js)

javascript

Copy

Download

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import './styles.css';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

Step 12: Add basic styles (src/styles.css)

css

Copy

Download

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

monospace;

}

Step 13: Start the frontend

bash

Copy

Download

npm start

Connecting Frontend to Backend

1. Ensure both backend (port 5000) and frontend (port 3000) are running
2. The frontend is already configured to make API calls to http://localhost:5000/api/auth
3. CORS is enabled in the backend to allow requests from the frontend

Testing the System

1. **Register an admin**:
   * Navigate to http://localhost:3000/register
   * Fill in a username and password (min 6 characters)
   * Submit the form
   * You should be redirected to the dashboard
2. **Login**:
   * Navigate to http://localhost:3000/login
   * Enter the credentials you registered with
   * Submit the form
   * You should be redirected to the dashboard
3. **Access protected routes**:
   * Try accessing http://localhost:3000/dashboard directly
   * If not logged in, you should be redirected to the login page
4. **Logout**:
   * Click the logout button on the dashboard
   * You should be logged out and unable to access protected routes

Additional Security Considerations

1. **Environment Variables**:
   * Never commit your .env file to version control
   * Use strong JWT secrets
2. **Password Requirements**:
   * Consider adding more complex password requirements
   * Implement password strength meters on the frontend
3. **Rate Limiting**:
   * Add rate limiting to prevent brute force attacks
4. **HTTPS**:
   * In production, always use HTTPS for all communications
5. **Token Storage**:
   * Consider using HttpOnly cookies for more secure token storage

This complete system provides a secure admin authentication flow with password hashing using bcryptjs and JWT for session management.

node app.js file:///D:/Users/Indizine/Sahana/bni/backend/app.js:1 const express = require('express'); ^ ReferenceError: require is not defined in ES module scope, you can use import instead This file is being treated as an ES module because it has a '.js' file extension and 'D:\Users\Indizine\Sahana\bni\backend\package.json' contains "type": "module". To treat it as a CommonJS script, rename it to use the '.cjs' file extension. at file:///D:/Users/Indizine/Sahana/bni/backend/app.js:1:17 at ModuleJob.run (node:internal/modules/esm/module\_job:274:25) at async onImport.tracePromise.\_\_proto\_\_ (node:internal/modules/esm/loader:644:26) at async asyncRunEntryPointWithESMLoader (node:internal/modules/run\_main:116:5) Node.js v22.15.0

This error occurs because your project is configured to use ES modules (indicated by "type": "module" in your package.json), but you're using CommonJS require() syntax. Here's how to fix it:

Solution 1: Convert to ES Modules (Recommended)

1. Update app.js to use ES modules syntax:

javascript

Copy

Download

import express from 'express';

import cors from 'cors';

import connectDB from './config/db.js';

import authRoutes from './routes/authRoutes.js';

import dotenv from 'dotenv';

dotenv.config();

const app = express();

*// Connect to database*

connectDB();

*// Middleware*

app.use(cors());

app.use(express.json());

*// Routes*

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

*// Error handling*

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({ message: 'Something went wrong!' });

});

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

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

2. Update all other files to use ES modules:

For each file in your project:

* Replace require() with import
* Replace module.exports with export default or named exports

Example for db.js:

javascript

Copy

Download

import mongoose from 'mongoose';

import dotenv from 'dotenv';

dotenv.config();

const connectDB = async () => {

try {

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

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log('MongoDB connected successfully');

} catch (error) {

console.error('MongoDB connection error:', error);

process.exit(1);

}

};

export default connectDB;

Solution 2: Switch back to CommonJS (Alternative)

If you prefer to use CommonJS instead of ES modules:

1. Update package.json:

json

Copy

Download

{

"type": "commonjs",

*// ... rest of your package.json*

}

2. Keep all your files using require() and module.exports

Additional Notes:

1. For ES modules:
   * File extensions must be explicit in imports (e.g., './config/db.js')
   * Some older packages might need to be imported differently (check their documentation)
2. For Node.js v22.15.0, ES modules are fully supported, so Solution 1 is recommended for new projects.
3. If you choose ES modules, make sure to update all your imports:
   * const express = require('express') → import express from 'express'
   * module.exports = connectDB → export default connectDB
   * For named exports: exports.functionName → export function functionName
4. If you get "Cannot use import statement outside a module" after switching to ES modules, make sure:
   * Your package.json has "type": "module"
   * All file extensions are .js (not .mjs)
   * You're using Node.js version that supports ES modules (v12+)

Choose the approach that best fits your project needs. For new projects, ES modules are generally recommended as they're the modern standard.

const jwt = require('jsonwebtoken'); const Admin = require('../models/Admin'); const authMiddleware = async (req, res, next) => { try { // Get token from header const token = req.header('Authorization')?.replace('Bearer ', ''); if (!token) { return res.status(401).json({ message: 'No token, authorization denied' }); } // Verify token const decoded = jwt.verify(token, process.env.JWT\_SECRET); const admin = await Admin.findById(decoded.id).select('-password'); if (!admin) { return res.status(401).json({ message: 'Token is not valid' }); } req.admin = admin; next(); } catch (error) { res.status(401).json({ message: 'Token is not valid' }); } }; module.exports = authMiddleware; change this also

Here's the updated authMiddleware.js using ES module syntax (import/export) instead of CommonJS (require/module.exports):

javascript

Copy

Download

import jwt from 'jsonwebtoken';

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

const authMiddleware = async (req, res, next) => {

try {

*// Get token from header*

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ message: 'No token, authorization denied' });

}

*// Verify token*

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const admin = await Admin.findById(decoded.id).select('-password');

if (!admin) {

return res.status(401).json({ message: 'Token is not valid' });

}

req.admin = admin;

next();

} catch (error) {

res.status(401).json({ message: 'Token is not valid' });

}

};

export default authMiddleware;

Key changes made:

1. Changed require() to import
2. Changed module.exports to export default
3. Added .js extension to the Admin model import
4. The actual middleware logic remains exactly the same

Remember to also:

1. Make sure your package.json has "type": "module"
2. Update all other files in your project to use ES module syntax consistently
3. Ensure all import paths include file extensions (.js)

const mongoose = require('mongoose'); const bcrypt = require('bcryptjs'); const AdminSchema = new mongoose.Schema({ username: { type: String, required: true, unique: true, trim: true, lowercase: true }, password: { type: String, required: true, minlength: 6 }, createdAt: { type: Date, default: Date.now } }); // Hash password before saving AdminSchema.pre('save', async function(next) { if (!this.isModified('password')) return next(); try { const salt = await bcrypt.genSalt(10); this.password = await bcrypt.hash(this.password, salt); next(); } catch (error) { next(error); } }); // Method to compare passwords AdminSchema.methods.comparePassword = async function(candidatePassword) { return await bcrypt.compare(candidatePassword, this.password); }; module.exports = mongoose.model('Admin', AdminSchema); change this also

Here's the updated Admin.js model file using ES module syntax:

javascript

Copy

Download

import mongoose from 'mongoose';

import bcrypt from 'bcryptjs';

const AdminSchema = new mongoose.Schema({

username: {

type: String,

required: true,

unique: true,

trim: true,

lowercase: true

},

password: {

type: String,

required: true,

minlength: 6

},

createdAt: {

type: Date,

default: Date.now

}

});

*// Hash password before saving*

AdminSchema.pre('save', async function(next) {

if (!this.isModified('password')) return next();

try {

const salt = await bcrypt.genSalt(10);

this.password = await bcrypt.hash(this.password, salt);

next();

} catch (error) {

next(error);

}

});

*// Method to compare passwords*

AdminSchema.methods.comparePassword = async function(candidatePassword) {

return await bcrypt.compare(candidatePassword, this.password);

};

const Admin = mongoose.model('Admin', AdminSchema);

export default Admin;

Key changes made:

1. Replaced const mongoose = require('mongoose') with import mongoose from 'mongoose'
2. Replaced const bcrypt = require('bcryptjs') with import bcrypt from 'bcryptjs'
3. Replaced module.exports = mongoose.model('Admin', AdminSchema) with:

javascript

Copy

Download

const Admin = mongoose.model('Admin', AdminSchema);

export default Admin;

The schema definition and all methods remain exactly the same - only the module syntax has been updated to ES modules format.

const express = require('express'); const router = express.Router(); const authController = require('../controllers/authController'); const authMiddleware = require('../middleware/authMiddleware'); // Public routes router.post('/register', authController.register); router.post('/login', authController.login); // Protected route router.get('/admin', authMiddleware, authController.getCurrentAdmin); module.exports = router; change this also

Here's the updated authRoutes.js file using ES module syntax:

javascript

Copy

Download

import express from 'express';

import { register, login, getCurrentAdmin } from '../controllers/authController.js';

import authMiddleware from '../middleware/authMiddleware.js';

const router = express.Router();

*// Public routes*

router.post('/register', register);

router.post('/login', login);

*// Protected route*

router.get('/admin', authMiddleware, getCurrentAdmin);

export default router;

Key changes made:

1. Changed require() statements to import statements
2. Used destructuring to import specific functions from the authController
3. Added .js extensions to the import paths
4. Changed module.exports to export default
5. Simplified the route definitions by directly using the imported functions

Note: Make sure your authController.js is exporting the functions using named exports like this:

javascript

Copy

Download

export const register = async (req, res) => { ... };

export const login = async (req, res) => { ... };

export const getCurrentAdmin = async (req, res) => { ... };

Alternatively, if you want to keep the original controller exports, you could import them like this:

javascript

Copy

Download

import authController from '../controllers/authController.js';

*// Then use authController.register, authController.login, etc.*

Choose whichever import style matches your controller's export style.

node app.js node:internal/modules/esm/resolve:275 throw new ERR\_MODULE\_NOT\_FOUND( ^ Error [ERR\_MODULE\_NOT\_FOUND]: Cannot find module 'D:\Users\Indizine\Sahana\bni\backend\config\db.js' imported from D:\Users\Indizine\Sahana\bni\backend\app.js at finalizeResolution (node:internal/modules/esm/resolve:275:11) at moduleResolve (node:internal/modules/esm/resolve:860:10) at defaultResolve (node:internal/modules/esm/resolve:984:11) at ModuleLoader.defaultResolve (node:internal/modules/esm/loader:780:12) at #cachedDefaultResolve (node:internal/modules/esm/loader:704:25) at ModuleLoader.resolve (node:internal/modules/esm/loader:687:38) at ModuleLoader.getModuleJobForImport (node:internal/modules/esm/loader:305:38) at ModuleJob.\_link (node:internal/modules/esm/module\_job:137:49) { code: 'ERR\_MODULE\_NOT\_FOUND', url: 'file:///D:/Users/Indizine/Sahana/bni/backend/config/db.js' } Node.js v22.15.0

This error occurs because Node.js can't find your db.js file when using ES modules. Let's fix this step by step:

Solution

1. **First, ensure your file structure is correct**:

Copy

Download

backend/

├── config/

│ └── db.js

├── app.js

1. **Update your**db.js**file** (must use ES modules syntax):

javascript

Copy

Download

import mongoose from 'mongoose';

import dotenv from 'dotenv';

dotenv.config();

const connectDB = async () => {

try {

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

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log('MongoDB connected successfully');

} catch (error) {

console.error('MongoDB connection error:', error);

process.exit(1);

}

};

export default connectDB;

1. **Update your**app.js**imports**:

javascript

Copy

Download

import express from 'express';

import cors from 'cors';

import connectDB from './config/db.js'; *// Note the .js extension*

*// ... rest of your app.js code*

1. **Critical checks**:
   * Verify the file exists at backend/config/db.js
   * Ensure the filename is exactly db.js (not db.mjs or similar)
   * Check your package.json has "type": "module"
   * Make sure all imports include the full path with .js extension
2. **If using Windows**, try these additional steps:
   * Use forward slashes in imports: './config/db.js'
   * Check for hidden file extensions (Windows might show it as db.js.js)
   * Try renaming the file to db.mjs and update import to './config/db.mjs'

Common Fixes

1. **Case sensitivity**:
   * Ensure the case matches exactly (DB.js vs db.js)
2. **File extension visibility**:
   * Run dir in your config folder to verify the exact filename
3. **Alternative import**:  
   Try using absolute path:

javascript

Copy

Download

import connectDB from 'file:///D:/Users/Indizine/Sahana/bni/backend/config/db.js';

1. **Verify file content**:
   * Open db.js in your editor and confirm it has the ES module export

If you're still having issues, try:

bash

Copy

Download

node --experimental-modules app.js

This should resolve your module not found error when using ES modules in Node.js. The key is ensuring consistent ES module syntax across all files and proper file extensions in imports.

node app.js file:///D:/Users/Indizine/Sahana/bni/backend/routes/authRoutes.js:2 import { register, login, getCurrentAdmin } from '../controllers/authControllers.js'; ^^^^^^^^^^^^^^^ SyntaxError: The requested module '../controllers/authControllers.js' does not provide an export named 'getCurrentAdmin' at ModuleJob.\_instantiate (node:internal/modules/esm/module\_job:182:21) at async ModuleJob.run (node:internal/modules/esm/module\_job:266:5) at async onImport.tracePromise.\_\_proto\_\_ (node:internal/modules/esm/loader:644:26) at async asyncRunEntryPointWithESMLoader (node:internal/modules/run\_main:116:5) Node.js v22.15.0

This error occurs because there's a mismatch between what you're trying to import and what's actually being exported in your authController.js file. Let's fix this step by step:

Solution

1. First, check your authController.js file:

Make sure it's using named exports (not default export):

javascript

Copy

Download

*// controllers/authController.js*

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

import jwt from 'jsonwebtoken';

*// Named export for register*

export const register = async (req, res) => {

try {

const { username, password } = req.body;

const existingAdmin = await Admin.findOne({ username });

if (existingAdmin) {

return res.status(400).json({ message: 'Admin already exists' });

}

const admin = new Admin({ username, password });

await admin.save();

const token = jwt.sign({ id: admin.\_id }, process.env.JWT\_SECRET, {

expiresIn: '1h'

});

res.status(201).json({ token, admin: { id: admin.\_id, username: admin.username } });

} catch (error) {

res.status(500).json({ message: error.message });

}

};

*// Named export for login*

export const login = async (req, res) => {

try {

const { username, password } = req.body;

const admin = await Admin.findOne({ username });

if (!admin) {

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

}

const isMatch = await admin.comparePassword(password);

if (!isMatch) {

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

}

const token = jwt.sign({ id: admin.\_id }, process.env.JWT\_SECRET, {

expiresIn: '1h'

});

res.json({ token, admin: { id: admin.\_id, username: admin.username } });

} catch (error) {

res.status(500).json({ message: error.message });

}

};

*// Named export for getCurrentAdmin*

export const getCurrentAdmin = async (req, res) => {

try {

const admin = await Admin.findById(req.admin.id).select('-password');

res.json(admin);

} catch (error) {

res.status(500).json({ message: error.message });

}

};

2. Then update your authRoutes.js:

javascript

Copy

Download

*// routes/authRoutes.js*

import express from 'express';

import {

register,

login,

getCurrentAdmin

} from '../controllers/authController.js'; *// Note: Fixed typo from "authControllers" to "authController"*

import authMiddleware from '../middleware/authMiddleware.js';

const router = express.Router();

*// Public routes*

router.post('/register', register);

router.post('/login', login);

*// Protected route*

router.get('/admin', authMiddleware, getCurrentAdmin);

export default router;

Key fixes:

1. Changed authControllers.js to authController.js (removed the 's')
2. Ensured all three functions are exported as named exports in the controller
3. Used the correct import syntax in the routes file

3. Additional checks:

* Verify the filename is exactly authController.js (no typos)
* Make sure the file is in the correct controllers directory
* Check that all functions are exported using export const syntax
* Ensure all imported names match exactly with the exported names

Alternative Approach (if you prefer default exports):

If you want to use default exports instead, modify the files like this:

1. In authController.js:

javascript

Copy

Download

const authController = {

register: async (req, res) => { */\* ... \*/* },

login: async (req, res) => { */\* ... \*/* },

getCurrentAdmin: async (req, res) => { */\* ... \*/* }

};

export default authController;

1. In authRoutes.js:

javascript

Copy

Download

import authController from '../controllers/authController.js';

*// Then use:*

router.post('/register', authController.register);

router.post('/login', authController.login);

router.get('/admin', authMiddleware, authController.getCurrentAdmin);

Choose whichever pattern you prefer (named exports or default export), but be consistent throughout your project. The first approach (named exports) is generally recommended for larger projects as it's more explicit.

New chat



DeepThink (R1)

Search

AI-generated, for reference only