

JS auth.js

✕

routes > JS auth.js > ...

```
1  const express = require('express');
2  const bcrypt = require('bcryptjs');
3  const jwt = require('jsonwebtoken');
4  const { body, validationResult } = require('express-validator');
5
6  const router = express.Router();
7
8  // In-memory user storage (in production, use a database)
9  const users = [];
10
11 // Register endpoint
12 router.post('/register', [
13   body('email').isEmail().normalizeEmail(),
14   body('password').isLength({ min: 6 }),
15   body('role').isIn(['user', 'admin', 'moderator'])
16 ], async (req, res) => {
17   try {
18     const errors = validationResult(req);
19     if (!errors.isEmpty()) {
20       return res.status(400).json({ errors: errors.array() });
21     }
22
23     const { email, password, role, name } = req.body;
24
25     // Check if user already exists
26     const existingUser = users.find(user => user.email === email);
27     if (existingUser) {
28       return res.status(400).json({ message: 'User already exists' });
29     }
30
31     // Hash password
32     const saltRounds = 10;
33     const hashedPassword = await bcrypt.hash(password, saltRounds);
34
35     // Create user
36     const user = {
37       id: users.length + 1,
38       email,
39       password: hashedPassword,
40       role,
41       name: name || email.split('@')[0],
42       createdAt: new Date()
43     };
44
45     users.push(user);
```

JS auth.js

X

routes > JS auth.js > ...

```
16   ], async (req, res) => {
17
47     // Generate JWT token
48     const token = jwt.sign(
49       {
50         id: user.id,
51         email: user.email,
52         role: user.role,
53         name: user.name
54       },
55       process.env.JWT_SECRET,
56       { expiresIn: '24h' }
57     );
58
59     res.status(201).json({
60       message: 'User registered successfully',
61       token,
62       user: {
63         id: user.id,
64         email: user.email,
65         role: user.role,
66         name: user.name
67       }
68     });
69   } catch (error) {
70     res.status(500).json({ message: 'Server error', error: error.message });
71   }
72 });
73
74 // Login endpoint
75 router.post('/login', [
76   body('email').isEmail().normalizeEmail(),
77   body('password').exists()
78 ], async (req, res) => {
79   try {
80     const errors = validationResult(req);
81     if (!errors.isEmpty()) {
82       return res.status(400).json({ errors: errors.array() });
83     }
84
85     const { email, password } = req.body;
86
87     // Find user
88     const user = users.find(user => user.email === email);
89     if (!user) {
90       return res.status(400).json({ message: 'Invalid credentials' });
```

JS auth.js X

routes > JS auth.js > ...

```
78 ], async (req, res) => {
79
80 // Check password
81 const isValidPassword = await bcrypt.compare(password, user.password);
82 if (!isValidPassword) {
83   return res.status(400).json({ message: 'Invalid credentials' });
84 }
85
86 // Generate JWT token
87 const token = jwt.sign(
88   {
89     id: user.id,
90     email: user.email,
91     role: user.role,
92     name: user.name
93   },
94   process.env.JWT_SECRET,
95   { expiresIn: '24h' }
96 );
97
98 res.json({
99   message: 'Login successful',
100   token,
101   user: {
102     id: user.id,
103     email: user.email,
104     role: user.role,
105     name: user.name
106   }
107 });
108 } catch (error) {
109   res.status(500).json({ message: 'Server error', error: error.message });
110 }
111 });
112
113 // Get current user profile
114 router.get('/profile', require('../middleware/auth').authenticateToken, (req, res) => {
115   res.json({
116     user: {
117       id: req.user.id,
118       email: req.user.email,
119       role: req.user.role,
120       name: req.user.name
121     }
122   });
123 });
124 }
125
126 }
```

middleware > JS authjs > ...

```
1  const jwt = require('jsonwebtoken');
2
3  const authenticateToken = (req, res, next) => {
4    const authHeader = req.headers['authorization'];
5    const token = authHeader && authHeader.split(' ')[1];
6
7    if (!token) {
8      return res.status(401).json({ message: 'Access token required' });
9    }
10
11    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
12      if (err) {
13        return res.status(403).json({ message: 'Invalid or expired token' });
14      }
15      req.user = user;
16      next();
17    });
18  };
19
20  const requireRole = (roles) => {
21    return (req, res, next) => {
22      if (!req.user) {
23        return res.status(401).json({ message: 'Authentication required' });
24      }
25
26      if (!roles.includes(req.user.role)) {
27        return res.status(403).json({ message: 'Insufficient permissions' });
28      }
29
30      next();
31    };
32  };
33
34  module.exports = { authenticateToken, requireRole };
35
```

Search results - x

D15 A FSD EXPE x

GitHub - Yogita x

SHOURYADALV x

FSD/FSD EXP 4: x

FSD_Lab/FSD_e x

FSD Exp list - G x

JWT Auth App x

+ x

- x

x

localhost:3000/register

⌕ ☆ 📄 School ⋮

Register

Full Name

Paul Anderson

Email

paulanderson@gmail.com

Password

Confirm Password

Role

User

Register

Already have an account? [Login here](#)

Login

Email

paulanderson@gmail.com

Password

Login

Don't have an account? [Register here](#)

Dashboard

Welcome to your dashboard, Paul Anderson!

User Information

Name: Paul Anderson

Email: paulanderson@gmail.com

Role: USER

Protected Data

This is protected data that only authenticated users can see.