ROLE-BASED ACCESS CONTROL (RBAC) IN A MERN APPLICATION

A PROJECT REPORT

Submitted by

ARUN KUMAR

(23BAI70142)

ASMIT KUMAR

(23BAI70229)

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(AIML)

CHANDIGARH UNIVERSITY

NOVEMBER 2025

BONAFIDE CERTIFICATE

Certified that this project report " **ROLE-BASED ACCESS CONTROL** " is the bonafide work of **ARUN KUMAR BHAGAT** AND **ASMIT KUMAR** who carried out the project work under my/our supervision.

<<Signature of the Supervisors>>

SIGNATURE

Submitted for the project viva-voce examination held on
INTERNAL EXAMINER

<<Signature of the AGM-Technical>>

SIGNATURE

EXTERNAL EXAMINER

PROJECT TITLE

ROLE-BASED ACCESS CONTROL (RBAC) IN A MERN APPLICATION

PROJECT DESCRIPTION

This project aims to design and implement a fine-grained Role-Based Access Control (RBAC) system within a MERN (MongoDB, Express.js, React.js, Node.js) application. The RBAC model defines permissions based on roles such as Admin, Editor, and Viewer, controlling access to both UI components and backend APIs.

The Node.js backend enforces security through JWT-based authentication, encoding role and ownership claims. Middleware ensures authorization checks for routes and query-level filters in MongoDB to enforce data scoping. On the React frontend, UI elements are dynamically rendered based on the user's role, ensuring consistent security between client and server.

This system demonstrates row-level ownership control, where Editors can modify only their own content, while Admins have global privileges. Additionally, it integrates secure authentication, structured logging, and observability to monitor authorization events.

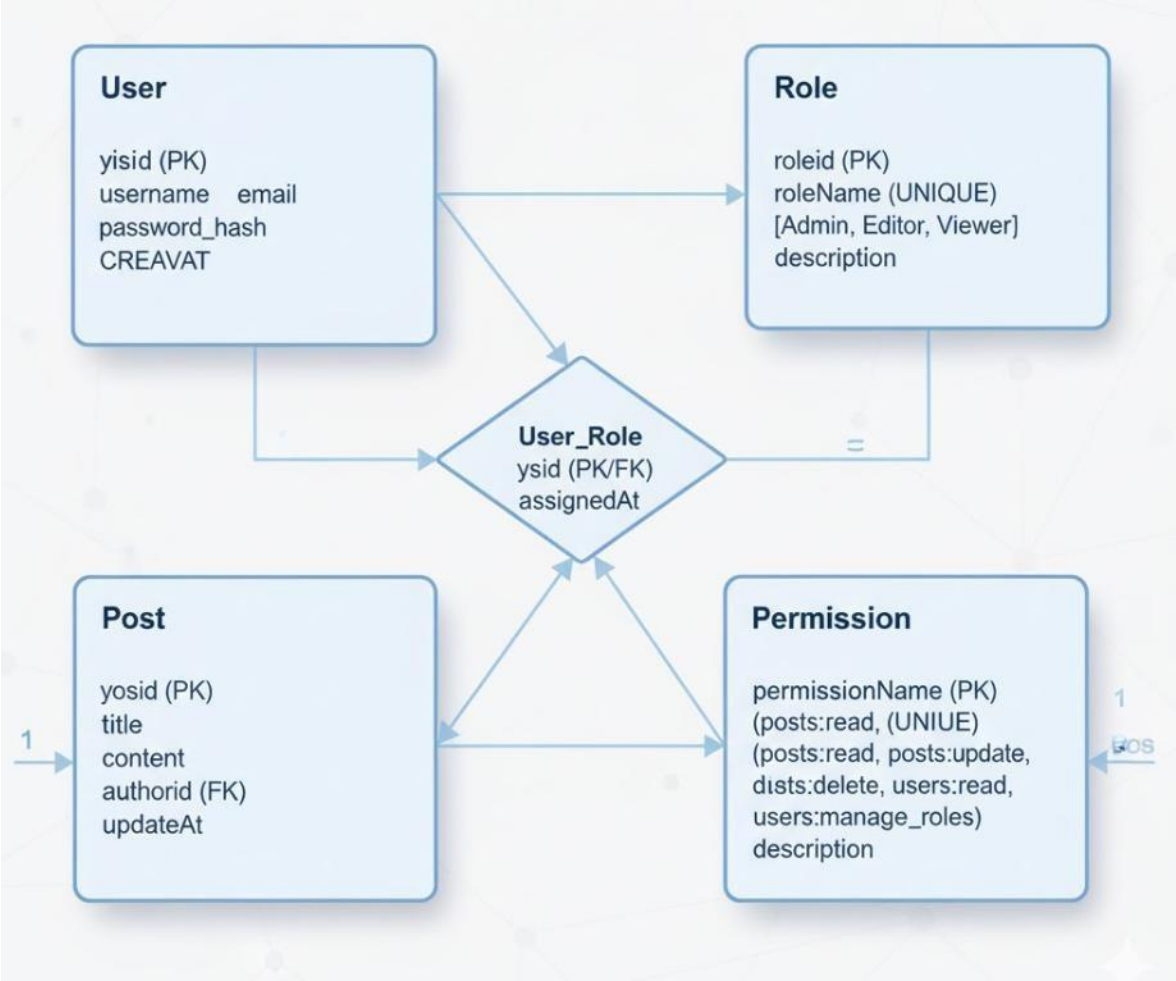
HARDWARE/SOFTWARE REQUIREMENTS

Hardware:
- Processor: Intel i5 or higher
- RAM: 8 GB minimum
- Hard Disk: 500 GB or higher
- Monitor: 15” color monitor
- Keyboard and Mouse

Software:
- Operating System: Windows 10 / Ubuntu 22.04
- Backend: Node.js (Express.js)
- Frontend: React.js
- Database: MongoDB
- Tools: Visual Studio Code, Postman, Git, Docker
- Browser: Google Chrome or Firefox

ER DIAGRAM

The ER diagram consists of the following main entities: User, Role, Permission, and Post. Their relationships define which roles can perform actions on posts and manage other users.



DATABASE SCHEMA

The database uses **MongoDB** with three main collections — **users**, **roles**, and **posts** — to implement fine-grained Role-Based Access Control (RBAC).

---

1. users Collection

Field	Type	Description
_id	ObjectId	Unique user ID
username	String	Login name (unique)
password	String	Encrypted password
email	String	Optional user email
roleId	ObjectId → roles._id	Assigned role
createdAt, updatedAt	Date	Timestamps

➡️ Each user is linked to one role.

---

2. roles Collection

Field	Type	Description
_id	ObjectId	Role ID
name	String	Role name (Admin, Editor, Viewer)
permissions	Array	Allowed actions
description	String	Role summary

➡️ Defines access levels and permissions.

---

3. posts Collection

Field	Type	Description
_id	ObjectId	Post ID
title, content	String	Post details
authorId	ObjectId → users._id	Post owner
status	String	Draft/Published
createdAt, updatedAt	Date	Timestamps

➡️ Each post belongs to one user; Editors can modify only their own posts.

Relationships:

- User ↔ Role → One-to-One
- Role ↔ Permissions → One-to-Many
- User ↔ Post → One-to-Many

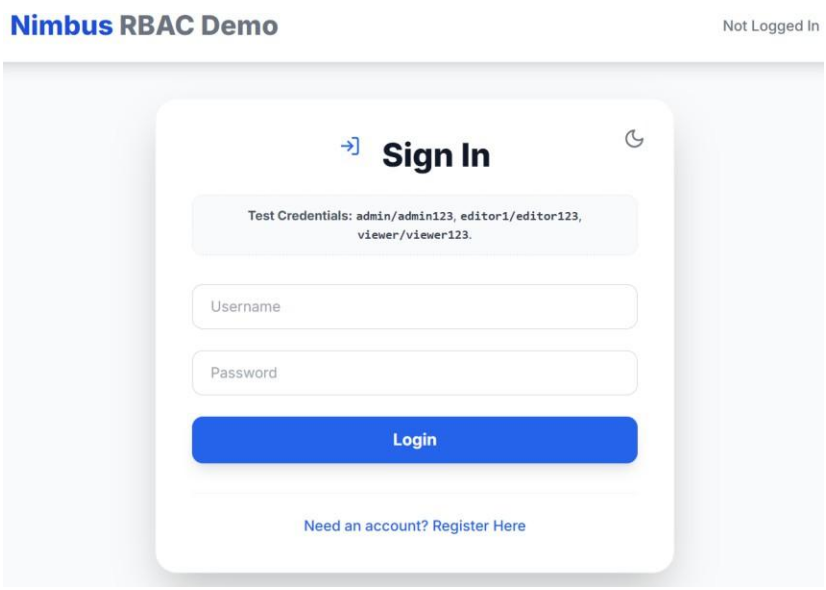
FRONT-END SCREENS

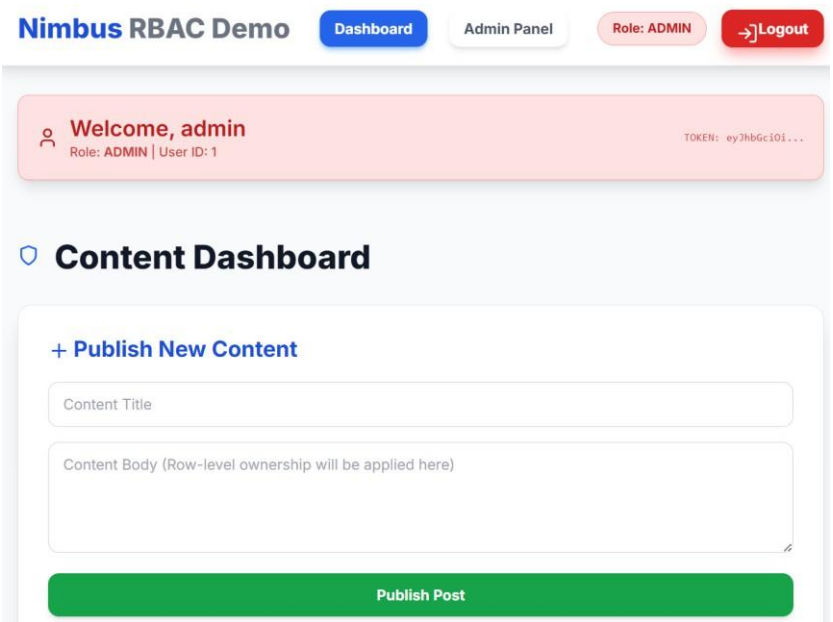
The frontend contains the following UI components:

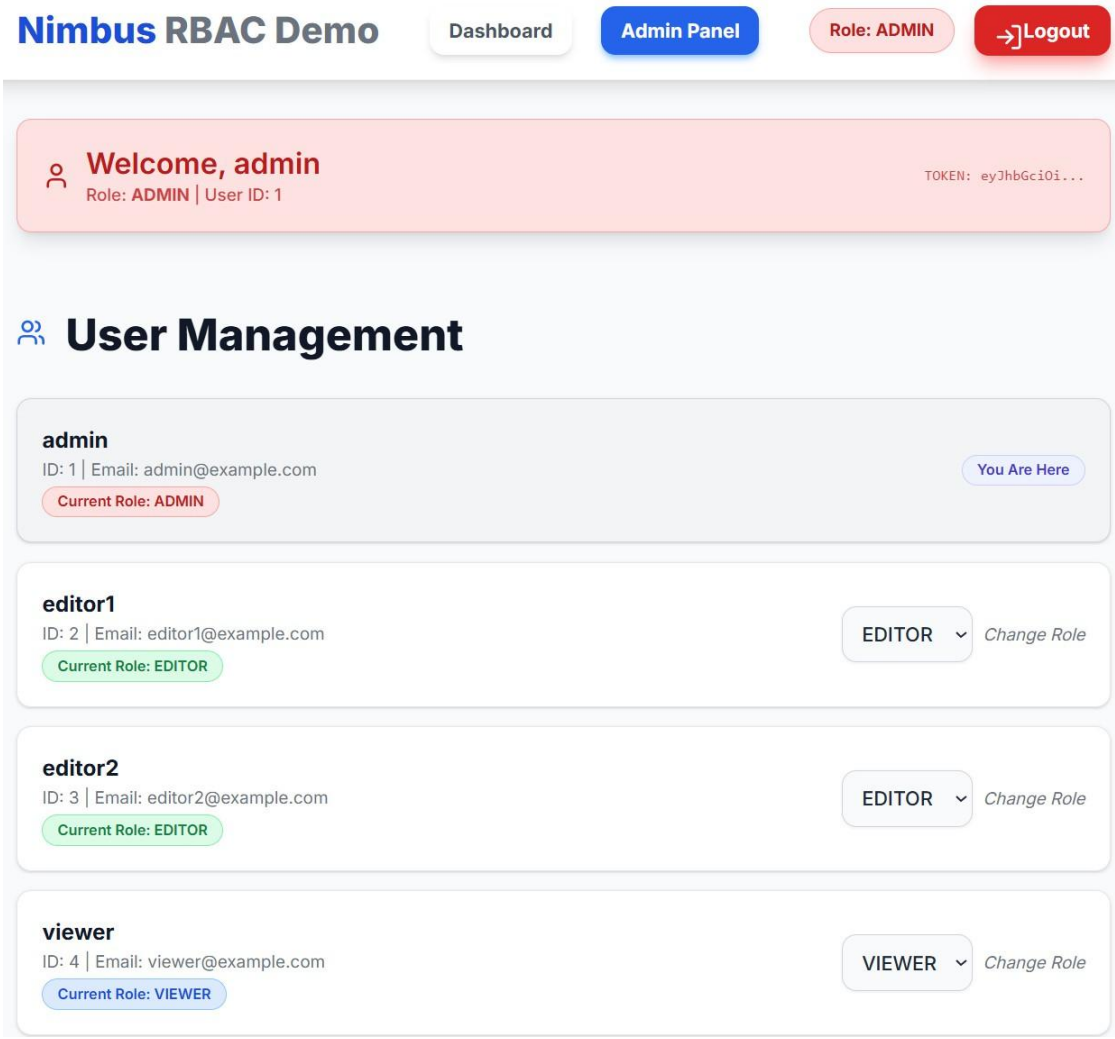
1. Login Page – Secure authentication with JWT

2. Dashboard – Displays user's accessible data

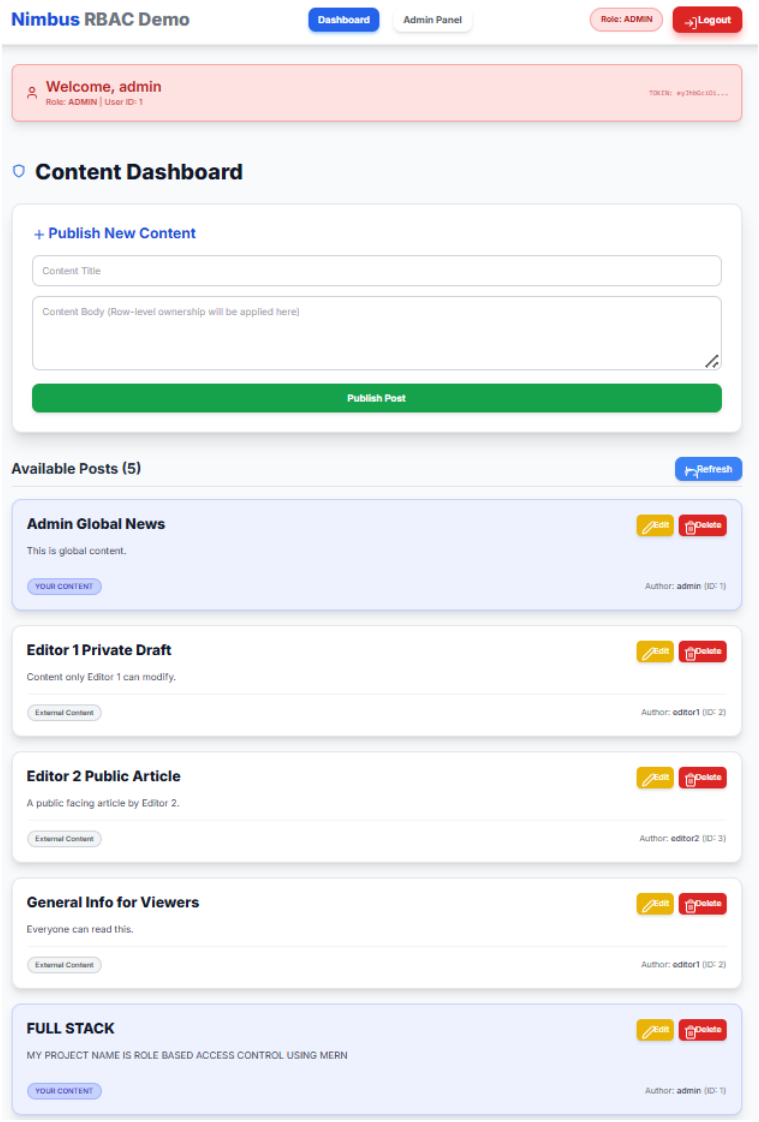
3. Admin Panel – Role management and audit logs

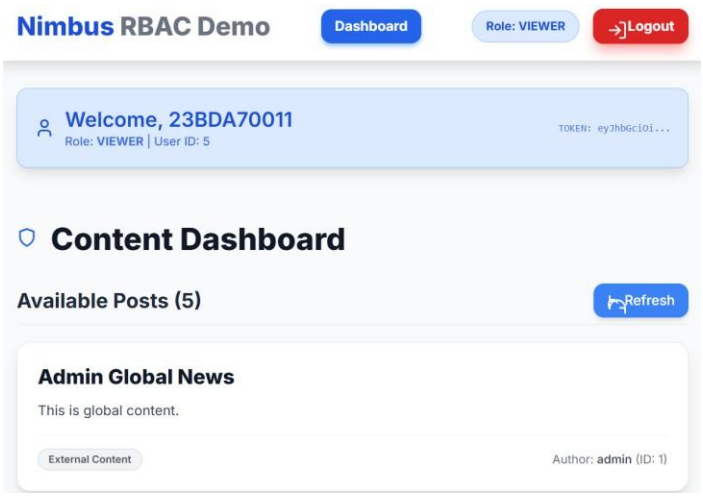





Nimbus RBAC Demo
Dashboard
Admin Panel
Role: ADMIN
Logout
Welcome, admin
Role: ADMIN | User ID: 1
TOKEN: eyJhbGciOi...
User Management
admin
ID: 1 | Email: admin@example.com
Current Role: ADMIN
You Are Here
editor1
ID: 2 | Email: editor1@example.com
Current Role: EDITOR
EDITOR
Change Role
editor2
ID: 3 | Email: editor2@example.com
Current Role: EDITOR
EDITOR
Change Role
viewer
ID: 4 | Email: viewer@example.com
Current Role: VIEWER
VIEWER
Change Role

OUTPUT SCREENS





LIMITATIONS & FUTURE SCOPE

LIMITATIONS

1. Static role structure with no dynamic updates.
2. Limited flexibility in permission management.
3. Basic audit logging without detailed tracking.
4. No support for external authentication systems.
5. Limited scalability for large-scale deployments.

FUTURE SCOPE

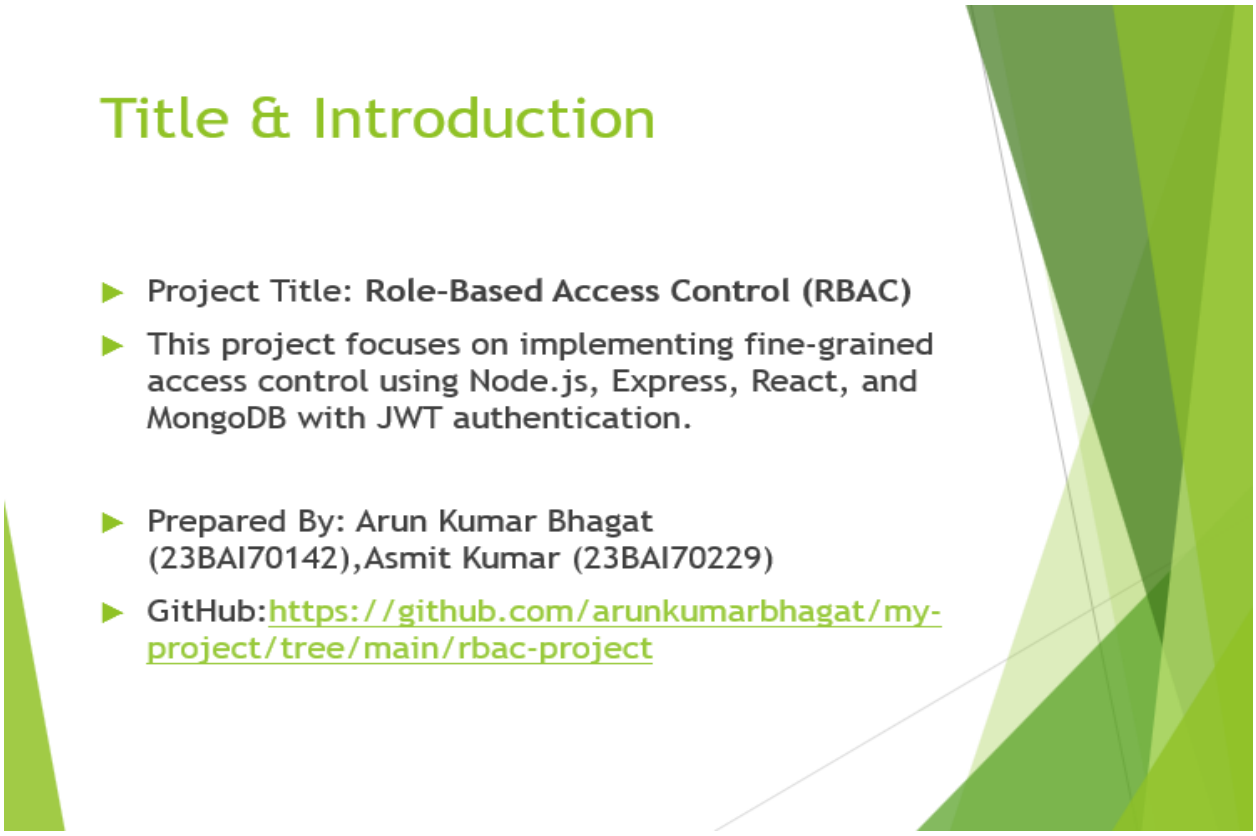
1. Add dynamic role and permission management.
2. Implement advanced audit logging and analytics.
3. Integrate OAuth2.0 and LDAP for enterprise SSO.
4. Include AI-based anomaly and threat detection.
5. Develop a comprehensive admin analytics dashboard.

GITHUB URL

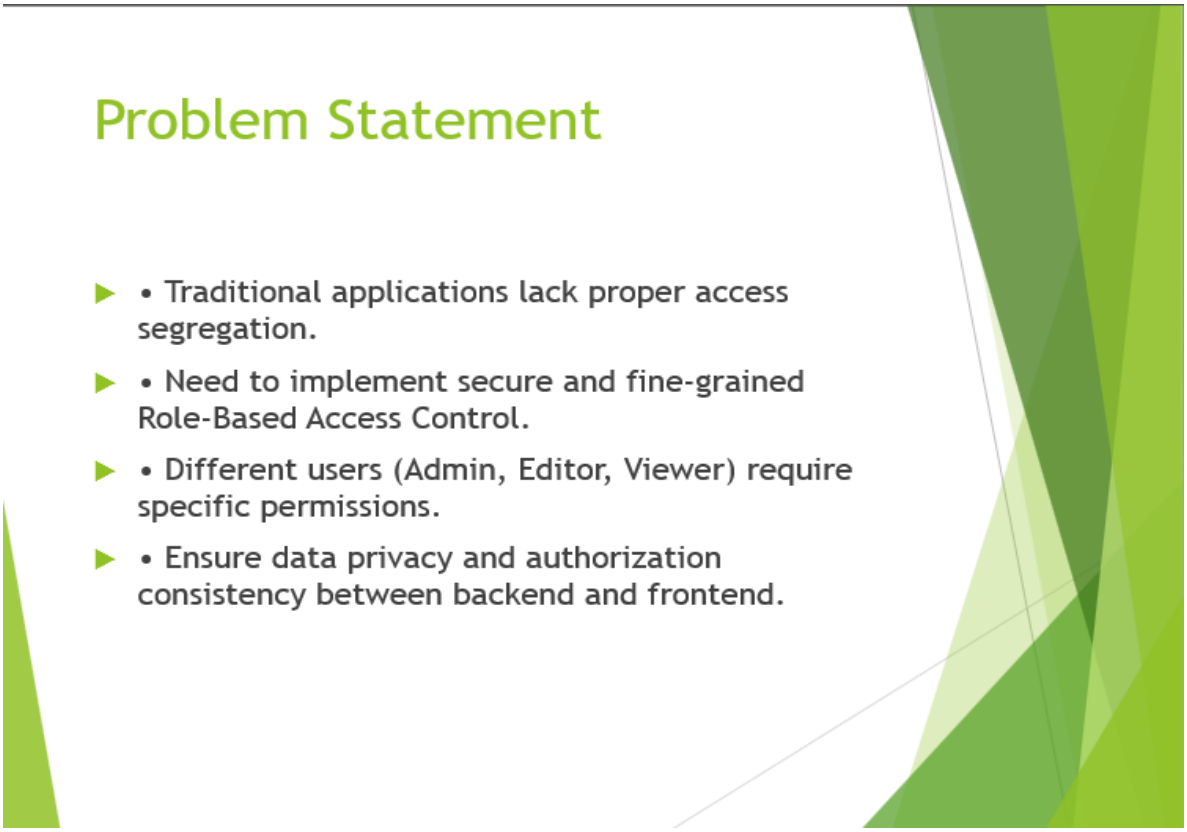
GitHub Repository: https://github.com/arunkumarbhagat/my-project/tree/main/rbac-project

PPT SLIDES

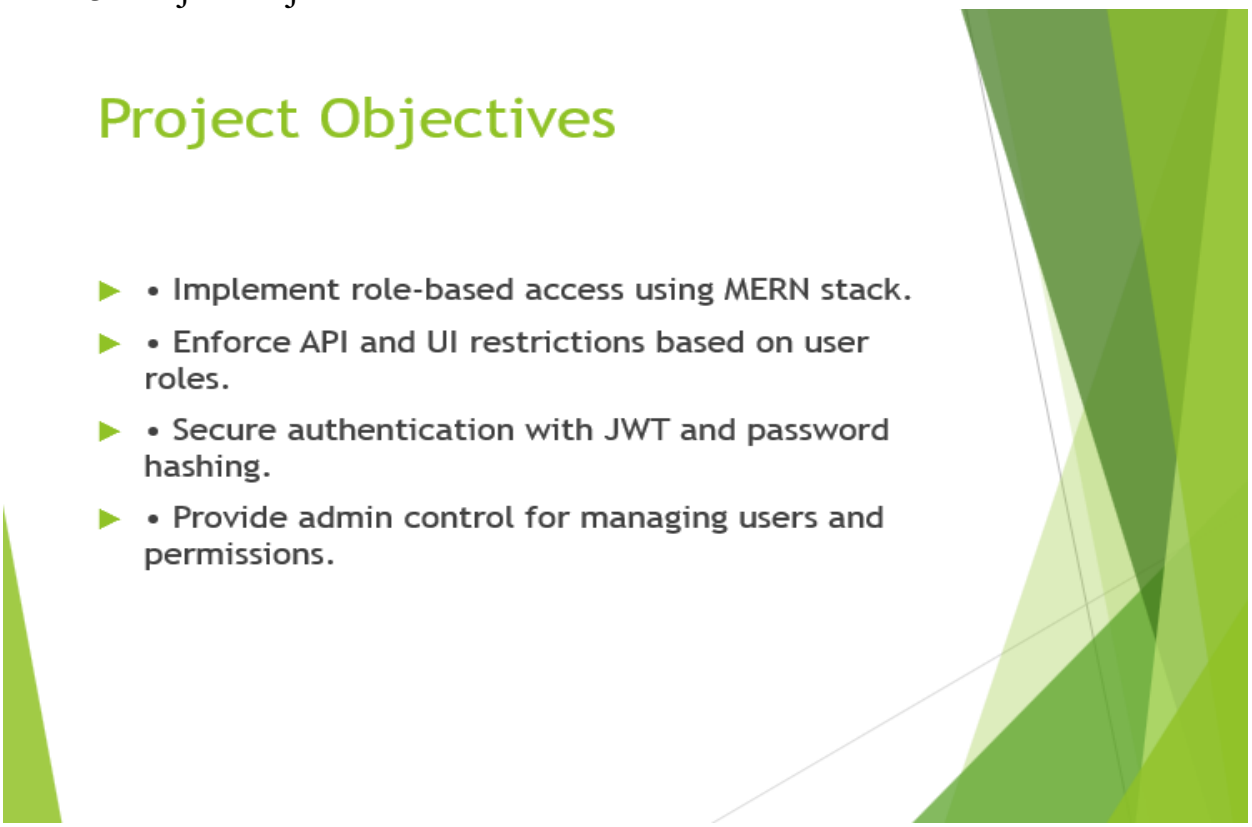
Slide 1: Title and Introduction



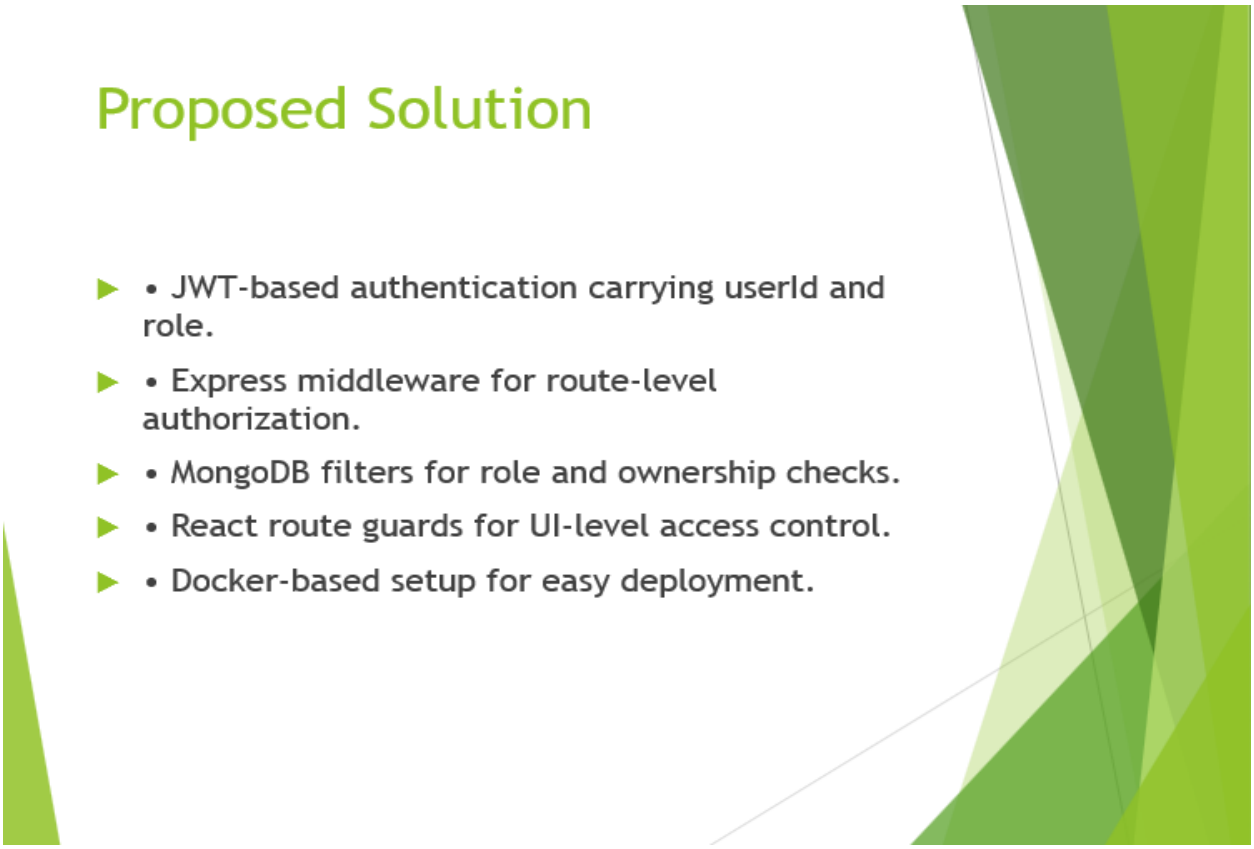
Slide 2: Problem Statement



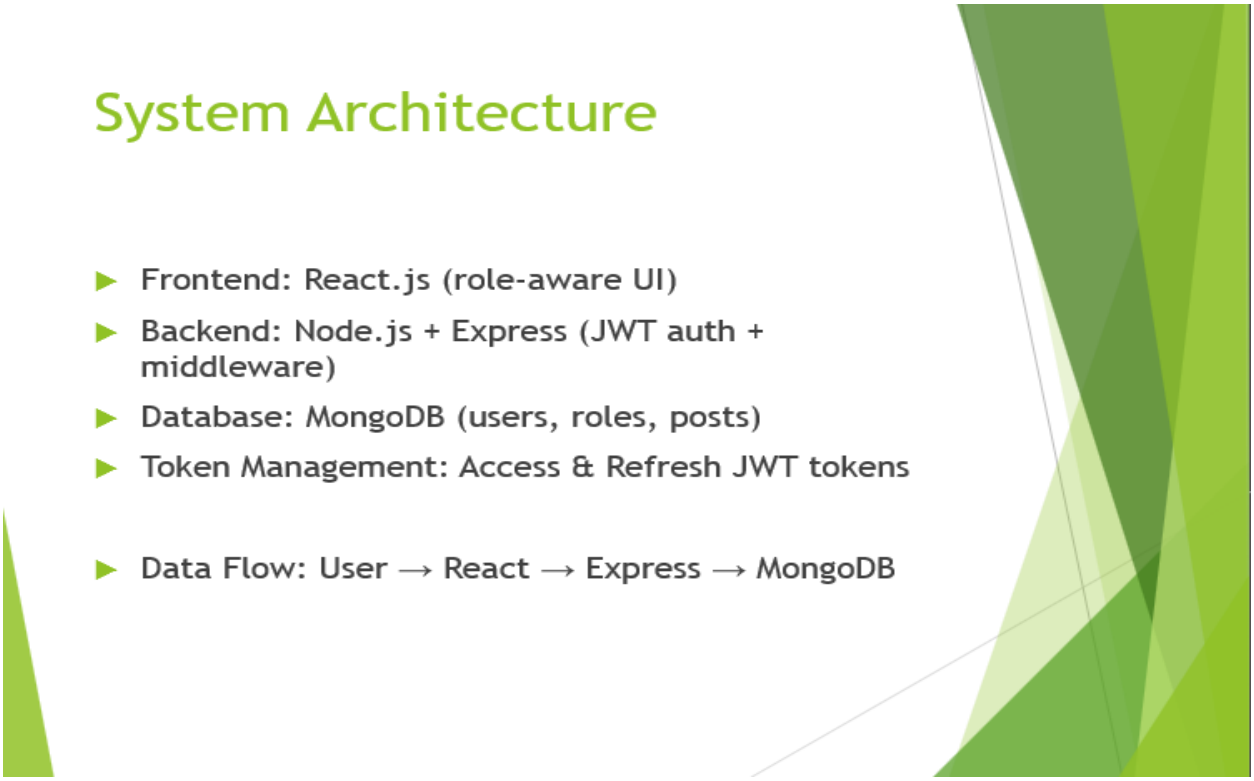
Slide 3: Project Objectives



slide 4: Proposed Solution

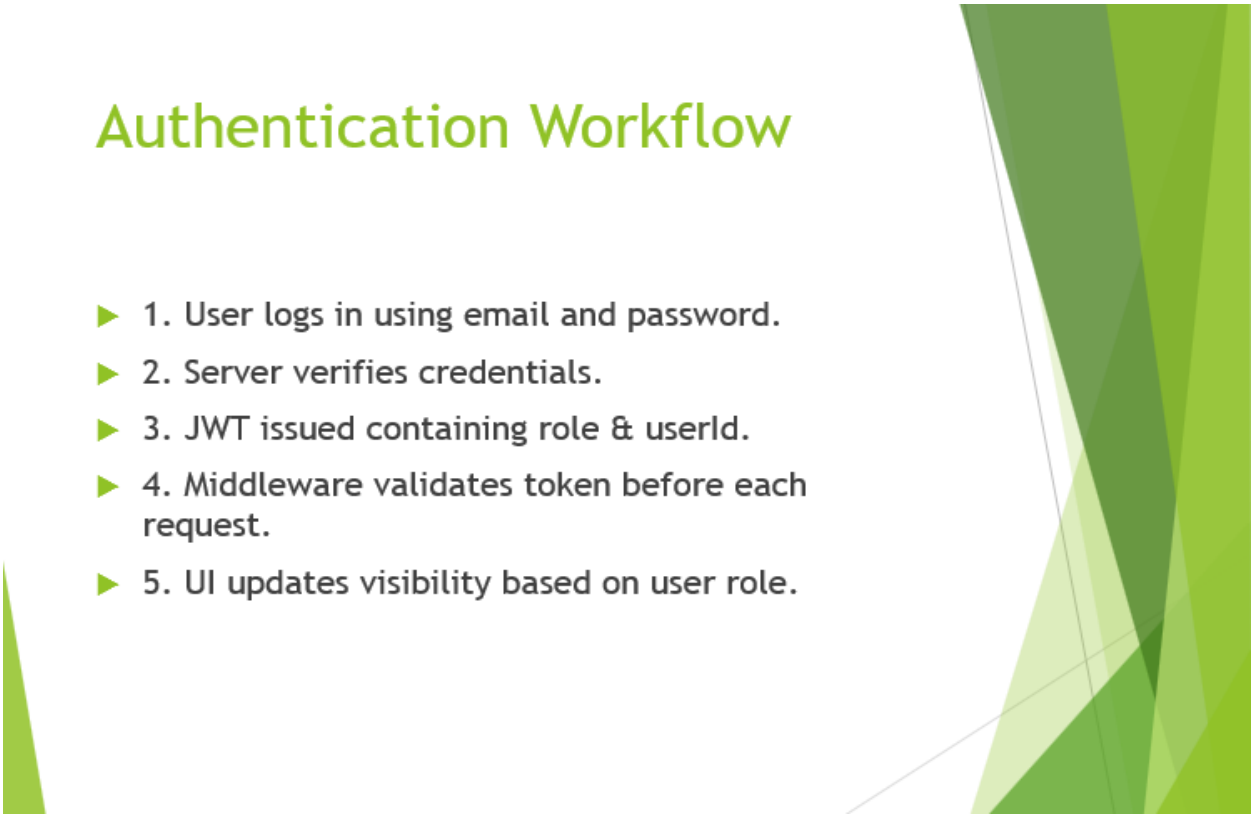


slide 5: System Architecture

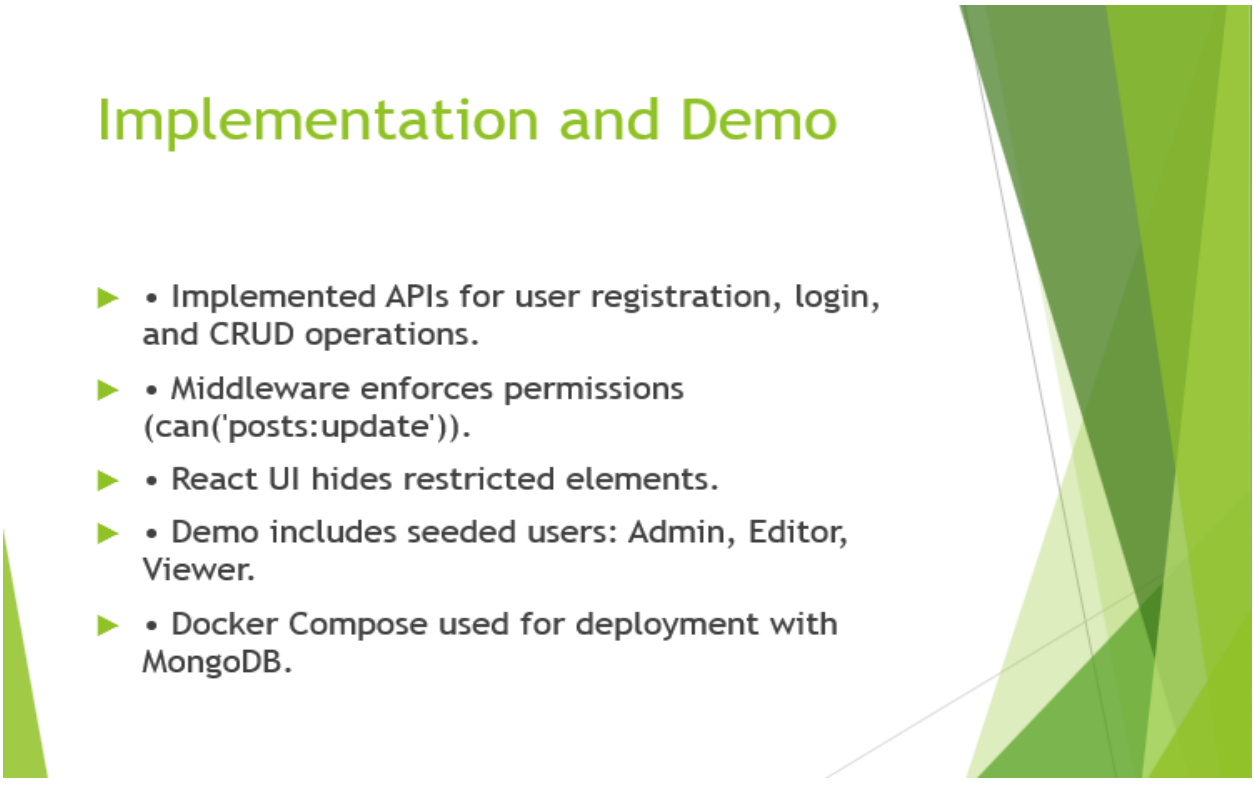


Slide 6: Security & Validation

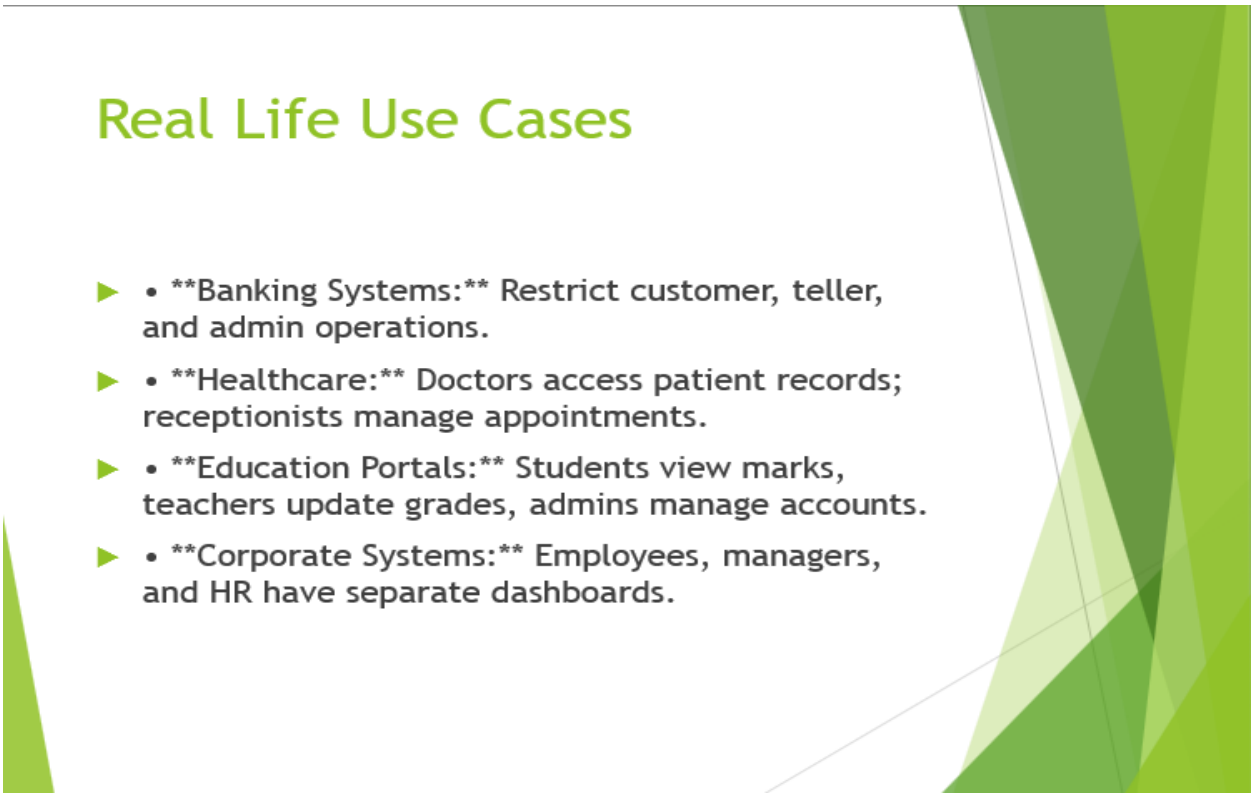
Slide 7: Authentication Workflow



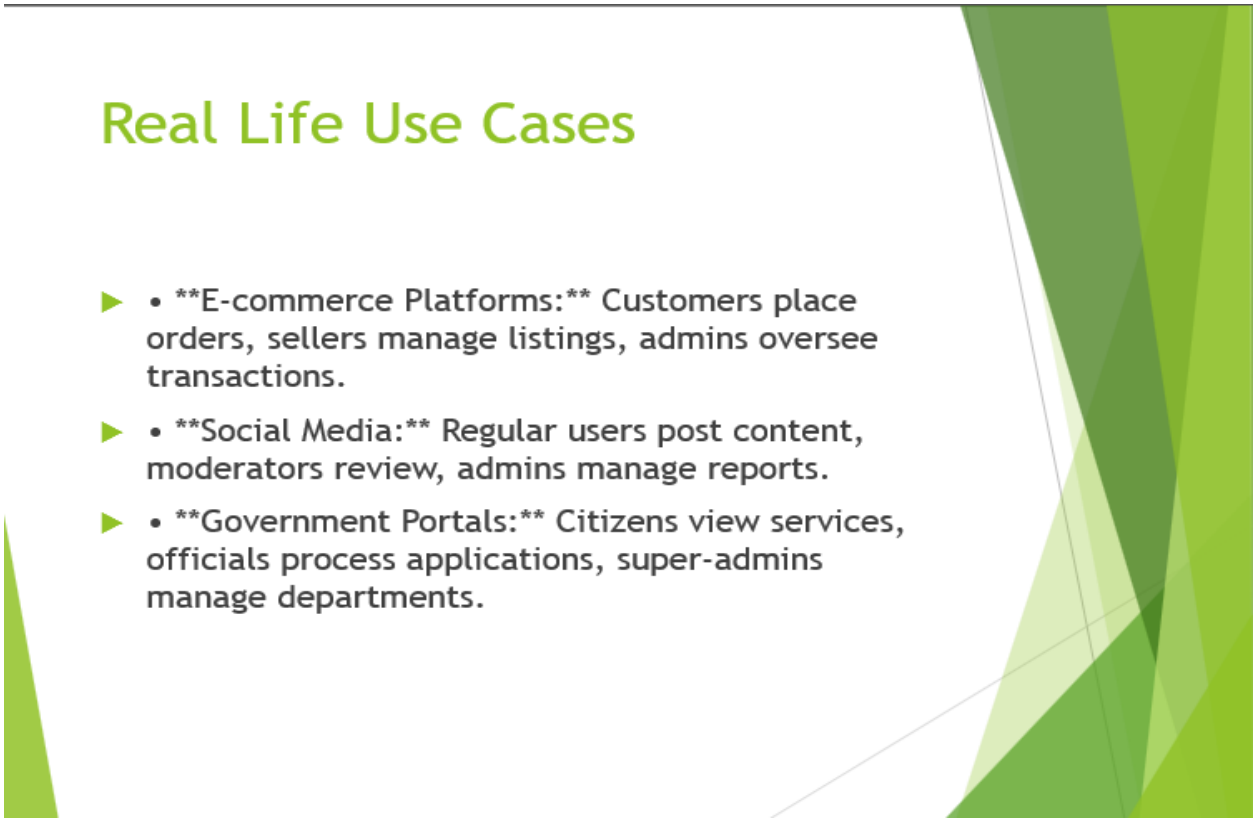
Slide 8: Implementation and Demo



Slide 9: Real Life Use Cases

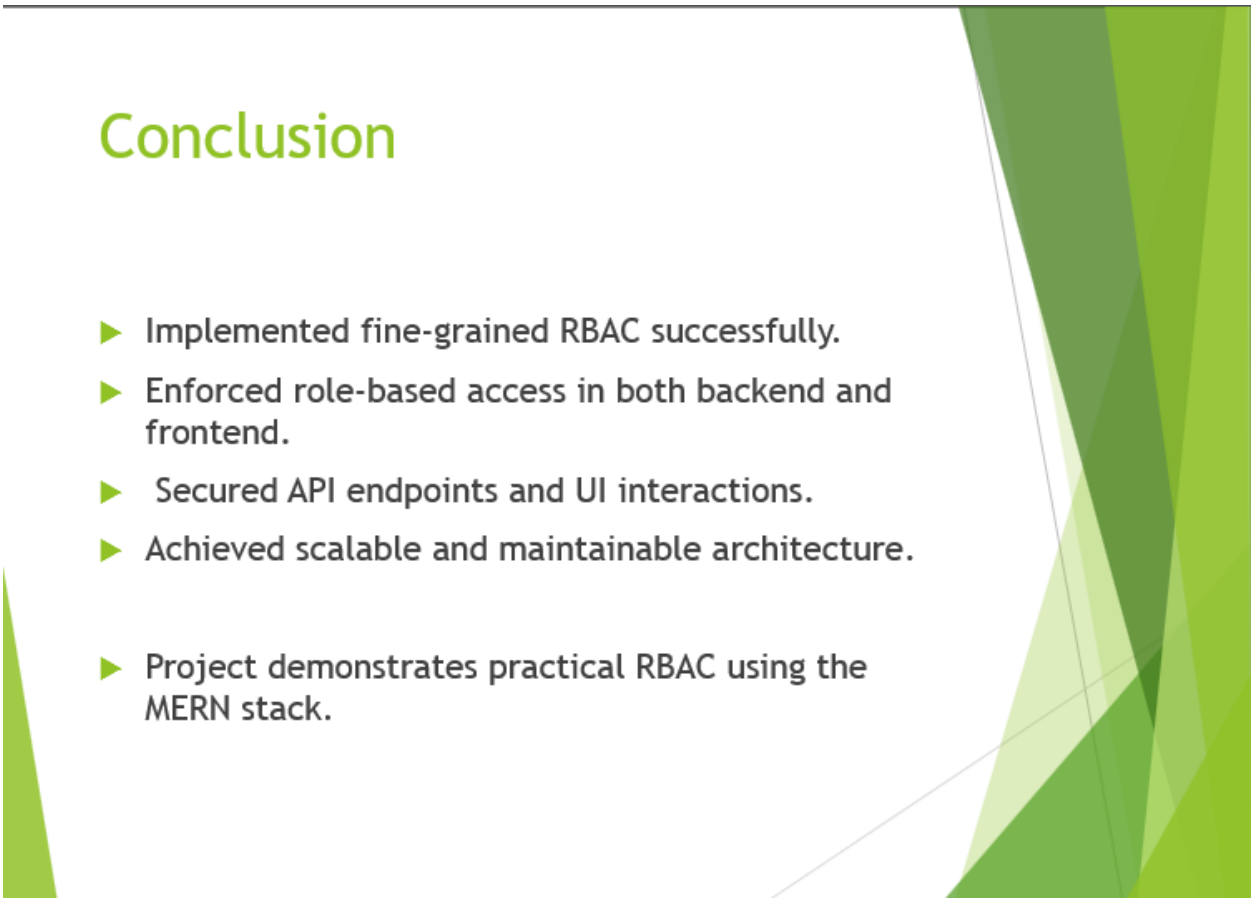


Slide 10: Real Life Use Cases



Slide 11: Limitations and Future Enhancements

Slide 12: Conclusion



Slide 13: Thank You

PROJECT CODE

BACKEND (SERVER.JS)

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const cors = require('cors');

// --- Configuration ---
const JWT_SECRET =
'supersecurejwtsecret'; // In a real app,
use environment variables!
const JWT_EXPIRATION = '1h';
const PORT = 3001;

// --- Mock Database (Simulating
Mongoose/MongoDB) ---
let users = [];
let posts = [];
let nextUserId = 1;
let nextPostId = 1;

// --- RBAC Permission Matrix (Core
Feature) ---
const PERMISSIONS = {
    ADMIN: {
        'users': ['create', 'read', 'update',
'delete'],
        'posts': ['create', 'read', 'update',
'delete'],
        'admin': ['manage_roles']
    },
    EDITOR: {
        'posts': ['create', 'read'],
        'self_posts': ['update', 'delete'], //
Can update/delete only their own posts
        'users': ['read']
    },
    VIEWER: {
        'posts': ['read']
    }
};

// --- Security Helpers ---
const hashPassword = (password) =>
bcrypt.hashSync(password, 10);

// --- Seeding Data (Key Feature: Seed &
Dev Setup) ---
const seedDatabase = () => {
    // 1. Create Users
    users = [
        { id: nextUserId++, username:
'admin', password:
hashPassword('admin123'), role:
'ADMIN', email: 'admin@example.com'
},
        { id: nextUserId++, username:
'editor1', password:
hashPassword('editor123'), role:
'EDITOR', email:
'editor1@example.com' },
        { id: nextUserId++, username:
'editor2', password:
hashPassword('editor123'), role:
'EDITOR', email:
'editor2@example.com' },
        { id: nextUserId++, username:
'viewer', password:
hashPassword('viewer123'), role:
'VIEWER', email:
'viewer@example.com' },
```

```
    ];

    // 2. Create Posts
    const editor1Id = users.find(u =>
u.username === 'editor1').id;
    const editor2Id = users.find(u =>
u.username === 'editor2').id;

    posts = [
        { id: nextPostId++, title: 'Admin
Global News', content: 'This is global
content.', authorId: users.find(u =>
u.username === 'admin').id, author:
'admin', createdAt: new Date() },
        { id: nextPostId++, title: 'Editor 1
Private Draft', content: 'Content only
Editor 1 can modify.', authorId:
editor1Id, author: 'editor1', createdAt:
new Date() },
        { id: nextPostId++, title: 'Editor 2
Public Article', content: 'A public facing
article by Editor 2.', authorId: editor2Id,
author: 'editor2', createdAt: new Date()
},
        { id: nextPostId++, title: 'General
Info for Viewers', content: 'Everyone can
read this.', authorId: editor1Id, author:
'editor1', createdAt: new Date() },
    ];
    console.log(`Database seeded with
${users.length} users and
${posts.length} posts.`);
};
seedDatabase();

// --- Express App Setup ---
const app = express();
app.use(express.json());

// CORS setup for frontend (Key
Feature: Security)
app.use(cors({
    origin: 'http://localhost:5173', //
Replace with your React app URL if
needed
    methods: ['GET', 'POST', 'PUT',
'DELETE'],
    allowedHeaders: ['Content-Type',
'Authorization'],
}));

// --- Middleware: Authentication (Auth
Tokens) ---
const authMiddleware = (req, res, next)
=> {
    const authHeader =
req.headers.authorization;
    if (!authHeader ||
!authHeader.startsWith('Bearer ')) {
        console.log('Authorization check
failed: Missing or invalid header.');
        return res.status(401).send({ error:
'Access Denied. No token provided.' });
    }

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

    try {
        const decoded = jwt.verify(token,
JWT_SECRET);
        // Attach user info (including role
and userId) to the request
        req.user = {
            id: decoded.userId,
            role: decoded.role,
            username: decoded.username //
Used for logging
        };
```

```
        next();
    } catch (ex) {
        console.log('JWT verification
failed:', ex.message);
        return res.status(400).send({ error:
'Invalid token.' });
    }
};

// --- Middleware: Authorization (API
Enforcement) ---
/**
 * Express middleware to check if the
authenticated user has the required
permission.
 * @param {string} resource - e.g.,
'posts', 'users', 'admin'
 * @param {string} action - e.g., 'create',
'read', 'update', 'delete', 'manage_roles'
 * @param {boolean}
[isOwnerCheck=false] - If true, checks
for 'self_' permission or relies on logic
below.
 */
const authorize = (resource, action,
isOwnerCheck = false) => {
    return (req, res, next) => {
        const role = req.user.role;
        const rolePermissions =
PERMISSIONS[role];

        if (!rolePermissions) {
            console.warn(`Attempted access
by user ${req.user.id} with invalid role:
${role}`);
            return res.status(403).send({
error: 'Forbidden. Invalid Role
Configuration.' });
        }

        let hasPermission = false;

        // 1. Standard Check (e.g., ADMIN
can 'posts:delete')
        if (rolePermissions[resource] &&
rolePermissions[resource].includes(actio
n)) {
            hasPermission = true;
        }

        // 2. Ownership Check (Key
Feature: Row-level Security)
        if (isOwnerCheck) {
            // Check for explicit 'self'
permission (e.g., EDITOR's
'self_posts:update')
            const selfResource =
`self_${resource}`;
            if (rolePermissions[selfResource]
&&
rolePermissions[selfResource].includes(
action)) {
                // Permission granted, now the
route handler must check the authorId
                hasPermission = true;
                req.isOwnerCheckRequired =
true; // Flag for route handler
            }
        }

        if (hasPermission) {
            next();
        } else {
            console.log(`Authorization
Denial (403): User ${req.user.username}
(Role: ${role}) failed check for
${resource}:${action}`);
```

```
            // Structured Log for
Observability
            console.error(JSON.stringify({
                level: 'error',
                message:
'AuthorizationDenied',
                userId: req.user.id,
                username: req.user.username,
                role: role,
                requiredPermission:
`${resource}:${action}`,
                timestamp: new
Date().toISOString(),
                correlationId: req.headers['x-
request-id'] || 'N/A'
            }));
            res.status(403).send({ error:
`Forbidden. Role '${role}' cannot
perform '${action}' on '${resource}'.` });
        }
    };
};

// --- Route: Authentication ---
app.post('/api/auth/register', async (req,
res) => {
    const { username, password, email,
role } = req.body;
    // Input validation/sanitization
(minimal check)
    if (!username || !password) return
res.status(400).send('Username  and
password are required.');
    if (users.some(u => u.username ===
username)) return
res.status(400).send('User already
exists.');

    const newUser = {
        id: nextUserId++,
        username,
        password:
hashPassword(password),
        email: email ||
`${username}@example.com`,
        // Default role is VIEWER unless
Admin is registering them (Admin path
is below)
        role: ['ADMIN', 'EDITOR',
'VIEWER'].includes(role) ? role :
'VIEWER',
    };
    users.push(newUser);

    const token = jwt.sign(
        { userId: newUser.id, role:
newUser.role, username:
newUser.username },
        JWT_SECRET,
        { expiresIn: JWT_EXPIRATION }
    );

    res.send({ token, role: newUser.role,
userId: newUser.id, username:
newUser.username });
});

app.post('/api/auth/login', async (req,
res) => {
    const { username, password } =
req.body;
    const user = users.find(u =>
u.username === username);

    if (!user ||
!bcrypt.compareSync(password,
user.password)) {
```

```
        return res.status(400).send('Invalid username or password.');
    }

    // Key Feature: Auth Tokens (JWT with role, userId)
    const token = jwt.sign(
        { userId: user.id, role: user.role, username: user.username },
        JWT_SECRET,
        { expiresIn: JWT_EXPIRATION }
    );

    res.send({
        token,
        role: user.role,
        userId: user.id,
        username: user.username,
        message: `Welcome, ${user.username} (${user.role})!`
    });
});

// --- Route: Posts API (Data Scoping / CRUD) ---

// CREATE Post
app.post('/api/posts', authMiddleware, authorize('posts', 'create'), (req, res) => {
    const { title, content } = req.body;
    const newPost = {
        id: nextPostId++,
        title,
        content,
        authorId: req.user.id,
        author: req.user.username,
        createdAt: new Date()
    };
    posts.push(newPost);
    res.status(201).send(newPost);
});

// READ Posts (Key Feature: Data Scoping)
app.get('/api/posts', authMiddleware, authorize('posts', 'read'), (req, res) => {
    const userRole = req.user.role;
    let filteredPosts = [...posts];

    // This is the core Data Scoping / Query-level Filter logic
    if (userRole === 'VIEWER' || userRole === 'EDITOR') {
        // In a real MongoDB query: Post.find({ isPublic: true }) or similar.
        // Mock: Assume all posts are visible for demonstration of the READ permission.
        // If we wanted to restrict access, we would filter here.
        // For simplicity, READ is global access for all authenticated users who pass authorize.
    }

    res.send(filteredPosts.map(p => ({
        id: p.id,
        title: p.title,
        content: p.content,
        author: p.author,
        authorId: p.authorId,
        createdAt: p.createdAt
    })));
});

// UPDATE Post (Key Feature: Ownership Predicates)
```

```
app.put('/api/posts/:id', authMiddleware, authorize('posts', 'update', true), (req, res) => {
    const postId = parseInt(req.params.id);
    const postIndex = posts.findIndex(p => p.id === postId);

    if (postIndex === -1) return res.status(404).send('Post not found.');

    const post = posts[postIndex];

    // Check 1: Did the authorize middleware flag an ownership requirement? (i.e., user is an Editor)
    if (req.isOwnerCheckRequired) {
        // Check 2: Actual row-level security check (authorId === userId)
        if (post.authorId !== req.user.id) {
            console.log(`Authorization Denial (403): User ${req.user.username} attempted to update post ${postId} belonging to ${post.authorId}`);
            return res.status(403).send({ error: 'Forbidden. You can only update your own posts.' });
        }
    }
    // Admin, who doesn't trigger req.isOwnerCheckRequired for 'posts:update', bypasses this check.

    // Update the post
    posts[postIndex] = {
        ...post,
        title: req.body.title || post.title,
        content: req.body.content || post.content,
        updatedAt: new Date()
    };

    res.send(posts[postIndex]);
});

// DELETE Post (Ownership Predicates)
app.delete('/api/posts/:id', authMiddleware, authorize('posts', 'delete', true), (req, res) => {
    const postId = parseInt(req.params.id);
    const postIndex = posts.findIndex(p => p.id === postId);
app.put('/api/admin/users/:id/role', authMiddleware, authorize('admin', 'manage_roles'), (req, res) => {
    const userId = parseInt(req.params.id);
    const { newRole } = req.body;

    if (!['ADMIN', 'EDITOR', 'VIEWER'].includes(newRole)) {
        return res.status(400).send('Invalid role provided.');
    }

    const userIndex = users.findIndex(u => u.id === userId);
    if (userIndex === -1) return res.status(404).send('User not found.');

    const oldRole = users[userIndex].role;
    users[userIndex].role = newRole;

    // Audit Log example (Key Feature: Administration/Observability)
    console.log(`[AUDIT] Role changed for User ID ${userId}
```

```
(${users[userIndex].username}) by
Admin ID ${req.user.id}. Old Role:
${oldRole}, New Role: ${newRole}`);

   res.send({ message: `Role updated to
${newRole} for user
${users[userIndex].username}.` });
});

// --- Server Start ---
app.listen(PORT, () => {
   console.log(`⚡ [Server] Running on
http://localhost:${PORT}`);
   console.log('--- Test Users ---');
   console.log('Admin: admin /
admin123');
   console.log('Editor: editor1 /
editor123');
   console.log('Viewer: viewer /
viewer123');
});
```

PACKAGE.JSON

```
{
 "name": "server",
 "version": "1.0.0",
 "description": "",
 "main": "server.js",
 "scripts": {
  "start": "node server.js",
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "dependencies": {
 "bcryptjs": "^2.4.3",
  "cors": "^2.8.5",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.2"
 }
}
```

React (Frontend of MERN RBAC System)

```
import React, { useState, useEffect, useCallback, useMemo } from 'react';
import { LogIn, User, Edit, Trash2, Plus, Users, Shield } from 'lucide-react';

const API_BASE_URL = 'http://localhost:3001/api';

// --- Permission Matrix ---
const PERMISSIONS = {
  ADMIN: { users: ['create','read','update','delete'], posts: ['create','read','update','delete'], admin: ['manage_roles'] },
  EDITOR: { posts: ['create','read'], self_posts: ['update','delete'], users: ['read'] },
  VIEWER: { posts: ['read'] }
};

// --- Auth Hook ---
const useAuth = () => {
const [user, setUser] = useState(null);
  useEffect(() => { const stored = localStorage.getItem('rbacUser'); if (stored) setUser(JSON.parse(stored)); }, []);
  const login = useCallback((data) => { setUser(data); localStorage.setItem('rbacUser', JSON.stringify(data)); }, []);
const logout = useCallback(() => { setUser(null); localStorage.removeItem('rbacUser'); }, []);
  const can = useCallback((resource, action, ownerId=null) => {
    if (!user?.role) return false;
    const rolePerms = PERMISSIONS[user.role];
    if (rolePerms[resource]?.includes(action)) return true;
    if (ownerId && rolePerms[`self_${resource}`]?.includes(action)) return user.userId === ownerId;
    return false;
  }, [user]);
  return { user, login, logout, can };
};

// --- UI Components ---
const LoadingSpinner = () => <div className="text-center p-10">Loading...</div>;

const AuthView = ({ login }) => {
```

```
  // Login/Register Form + Role-based demo credentials
};

const PostForm = ({ can, refreshPosts }) => {
  // Post creation form, allowed only for users with `posts:create`
};

const PostItem = ({ post, user, can, refreshPosts }) => {
  // Displays post content with Edit/Delete buttons conditionally rendered via `can()`
};

const Dashboard = ({ user, can, refreshPosts, posts }) => {
  // Fetch and display posts; enforce read permissions via `can('posts','read')`
};

const AdminPanel = ({ user, can }) => {
  // User management + Role assignment; accessible only to ADMIN via `can('admin','manage_roles')`
};

// --- Main Application ---
const App = () => {
  const { user, login, logout, can } = useAuth();
  const [posts, setPosts] = useState([]);
  const [currentView, setCurrentView] = useState('dashboard');

  const refreshPosts = useCallback((p) => setPosts(p), []);
  const navItems = useMemo(() => [
    { name: 'Dashboard', view: 'dashboard', show: true },
    { name: 'Admin Panel', view: 'admin', show: can('users','read') },
  ], [can]);

  if (!user) return <AuthView login={login} />;

  return (
    <div className="app">
      <header>
        <h1>RBAC Demo</h1>
        <button onClick={logout}>Logout</button>
      </header>
      <main>
        {currentView === 'admin'
          ? <AdminPanel user={user} can={can} />
```

```
        : <Dashboard user={user}
can={can}
refreshPosts={refreshPosts}
posts={posts} />}
      </main>
    </div>
  );
};

export default App;
```

INDEX.CSS (FOR STYLING)

:

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

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

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

a {
  font-weight: 500;
  color: #646cff;
  text-decoration: inherit;
}
a:hover {
  color: #535bf2;
}

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

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

button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  background-color: #1a1a1a;
  cursor: pointer;
  transition: border-color 0.25s;
}
button:hover {
  border-color: #646cff;
}
button:focus,
button:focus-visible {
  outline: 4px auto -webkit-focus-ring-
color;
}

@media (prefers-color-scheme: light) {
  :root {
```

```
    color: #213547;
    background-color: #ffffff;
  }
  a:hover {
    color: #747bff;
  }
  button {
    background-color: #f9f9f9;
  }
}
```

INDEX.HTML (FOR WEB)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Nimbus RBAC Demo</title>
  <!-- Tailwind CSS CDN -->
  <script
src="https://cdn.tailwindcss.com"></scri
pt>
  <link
href="https://fonts.googleapis.com/css2?
family=Inter:wght@100..900&display=s
wap" rel="stylesheet">
</head>

<body class="font-sans bg-gray-50 min-
h-screen">
  <div id="app-root" class="flex justify-
center items-center h-screen">
    <div class="animate-spin h-12 w-12
border-b-4 border-indigo-500 rounded-
full"></div>
  </div>

  <script type="module">
    /* ---------------- Core RBAC Logic ---
---------- */
    const API_BASE_URL =
"http://localhost:3001/api";
    const PERMISSIONS = {
      ADMIN: { users:
['create','read','update','delete'], posts:
['create','read','update','delete'], admin:
['manage_roles'] },
      EDITOR: { posts: ['create','read'],
self_posts: ['update','delete'], users:
['read'] },
      VIEWER: { posts: ['read'] }
    };

    let user = null, posts = [], currentView
= 'dashboard', theme = 'light';

    function can(resource, action,
ownerId=null) {
      if (!user) return false;
      const rolePerms =
PERMISSIONS[user.role];
      if
(rolePerms[resource]?.includes(action))
return true;
      if (ownerId &&
rolePerms[`self_${resource}`]?.includes
(action))
        return user.userId === ownerId;
      return false;
    }

    /* ---------------- Authentication --------
-------- */
    async function login(username,
password) {
```

```
    const res = await
fetch(`${API_BASE_URL}/auth/login`,
{
      method: 'POST', headers:
{'Content-Type':'application/json'},
      body: JSON.stringify({ username,
password })
    });
    const data = await res.json();
    if (!res.ok) throw new
Error(data.message);
    localStorage.setItem('rbacUser',
JSON.stringify(data));
    user = data;
    renderApp();
  }

  function logout() { user = null;
localStorage.removeItem('rbacUser');
renderApp(); }

  /* ---------------- Theme Toggle --------
-------- */
  function toggleTheme() {
    theme = theme === 'light' ? 'dark' :
'light';

document.documentElement.classList.to
ggle('dark');
  }

  /* ----------------- Rendering ----------
--- */
  function renderHeader() {
    return `
      <header class="bg-white shadow-
md p-4 flex justify-between">
        <h1 class="text-2xl font-bold text-
indigo-600">Nimbus RBAC
Demo</h1>
        ${user ? `<button
onclick="logout()" class="bg-red-600
text-white px-4 py-2 rounded-
lg">Logout</button>` : ''}
      </header>
    `;
  }

  function renderAuth() {
    document.getElementById("app-
root").innerHTML = `
      ${renderHeader()}
      <div class="max-w-md mx-auto
mt-20 bg-white p-8 rounded-xl
shadow">
        <h2 class="text-3xl font-bold mb-
4">Sign In</h2>
        <input id="username"
class="border p-2 w-full mb-3"
placeholder="Username" />
        <input id="password"
type="password" class="border p-2 w-
full mb-3" placeholder="Password" />
        <button
onclick="login(username.value,
password.value)" class="bg-indigo-600
text-white px-4 py-2 rounded w-
full">Login</button>
      </div>
    `;
  }

  function renderDashboard() {
    document.getElementById("app-
root").innerHTML = `
      ${renderHeader()}
      <main class="max-w-4xl mx-auto
p-6">
        <h2 class="text-3xl font-bold mb-
6">Dashboard</h2>
```

```
        <p>Welcome,
<strong>${user.username}</strong>
(${user.role})</p>
        <div class="mt-4 p-4 bg-gray-100
rounded-lg">Posts list will appear
here...</div>
      </main>
    `;
  }

  function renderApp() {
    user ? renderDashboard() :
renderAuth();
  }

document.addEventListener('DOMConte
ntLoaded', () => {
    const stored =
localStorage.getItem('rbacUser');
    if (stored) user =
JSON.parse(stored);
    renderApp();
  });
 </script>
</body>
</html>
```

REFERENCES

[1] MongoDB Documentation – *MongoDB Manual*, [Online]. Available: https://www.mongodb.com/docs/

[2] Express.js API Reference – *Express Web Framework (Node.js)*, [Online]. Available: https://expressjs.com/

[3] React.js Official Documentation – *React Developer Guide*, [Online]. Available: https://react.dev/

[4] Node.js Official Documentation – *Node.js Runtime Environment*, [Online]. Available: https://nodejs.org/

[5] OWASP Authentication Cheat Sheet – *Web Application Security Best Practices*, [Online]. Available: https://owasp.org/

[6] JWT.io – *JSON Web Tokens: Introduction and Libraries*, [Online]. Available: https://jwt.io/introduction/

[7] Mongoose Official Documentation – *MongoDB Object Data Modeling (ODM)*, [Online]. Available: https://mongoosejs.com/docs/

[8] Tailwind CSS Documentation – *Utility-first CSS Framework for Rapid UI Development*, [Online]. Available: https://tailwindcss.com/docs

[9] Lucide React Icons – *Open-source Icon Library for React Apps*, [Online]. Available: https://lucide.dev/

[10] Docker Documentation – *Docker Containers and Compose Overview*, [Online]. Available: https://docs.docker.com/

[11] JSON Web Token (RFC 7519) – *IETF Standard for Secure Authentication*, [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519

[12] W3C Web Security Guidelines – *Secure Web Development and Access Control Standards*, [Online]. Available: https://www.w3.org/Security/

BIBLIOGRAPHY

1. **Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996).** ***Role-Based Access Control Models.*** **IEEE Computer, 29(2), 38–47.**
2. **Ferraiolo, D. F., Kuhn, D. R., & Chandramouli, R. (2003).** ***Role-Based Access Control.*** **Artech House.**
3. **Papa, J. (2022).** ***Building Secure MERN Applications.*** **O'Reilly Media.**
4. **Mozilla Developer Network (MDN).** ***Web Security Essentials.*** **Retrieved from https://developer.mozilla.org**
5. **Stallings, W. (2019).** ***Network Security Essentials: Applications and Standards*** **(6th ed.). Pearson Education.**
6. **Goodrich, M. T., & Tamassia, R. (2018).** ***Introduction to Computer Security.*** **Addison-Wesley.**
7. **Hurst, J. (2021).** ***Mastering Node.js Security: Best Practices for Protecting Web Applications.*** **Packt Publishing.**
8. **Subramanian, V., & Kumar, R. (2023).** ***Full-Stack Development with MongoDB, Express, React, and Node.*** **Apress.**
9. **Chowdhury, D., & Shanmugasundaram, S. (2020).** ***Practical Role-Based Access Control for Modern Web Applications.*** **IEEE Access Journal.**
10. **OWASP Foundation. (2023).** ***OWASP Top 10: Security Risks for Web Applications.*** **Retrieved from https://owasp.org**