Capstone Project

23CSP-339 - Full Stack-I

Final Project Evaluation Guidelines

Role Based Access Control

Project Title

Role-Based Access Control (RBAC) in a MERN application

A PROJECT REPORT

Submitted by

NAME OF THE CANDIDATE(S)

Anish kumar(23BIS70123) Dipali Rawat(23BIS70134)

in partial fulfilment for the award of the degree of

IN

BRANCH OF STUDY

Chandigarh University

Nov 2025

BONAFIDE CERTIFICATE

Certified that this project report **"Role-Based Access Control (RBAC) in a MERN application"** is the Bonafide work of **"Anish kumar &Dipali rawat"** who carried out the project work under my/our supervision.

SIGNATURE

SIGNATURE

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

Project Title

Role-Based Access Control (RBAC) in a MERN application

Project Description

Design and implement fine-grained Role-Based Access Control (RBAC) in a MERN application where Admin, Editor, and Viewer roles govern which API actions and UI capabilities are allowed. Build a Node.js (Express) backend with JWT-based authentication that encodes role claims and enforces authorization via middleware and query-level filters in MongoDB. In React, reflect effective permissions by enabling/disabling controls, guarding routes, and hiding restricted data. Deliver seeded users/roles and demonstrate row-level ownership checks (e.g., Editors can modify their own content, Admins can manage all).

Hardware/Software Requirements

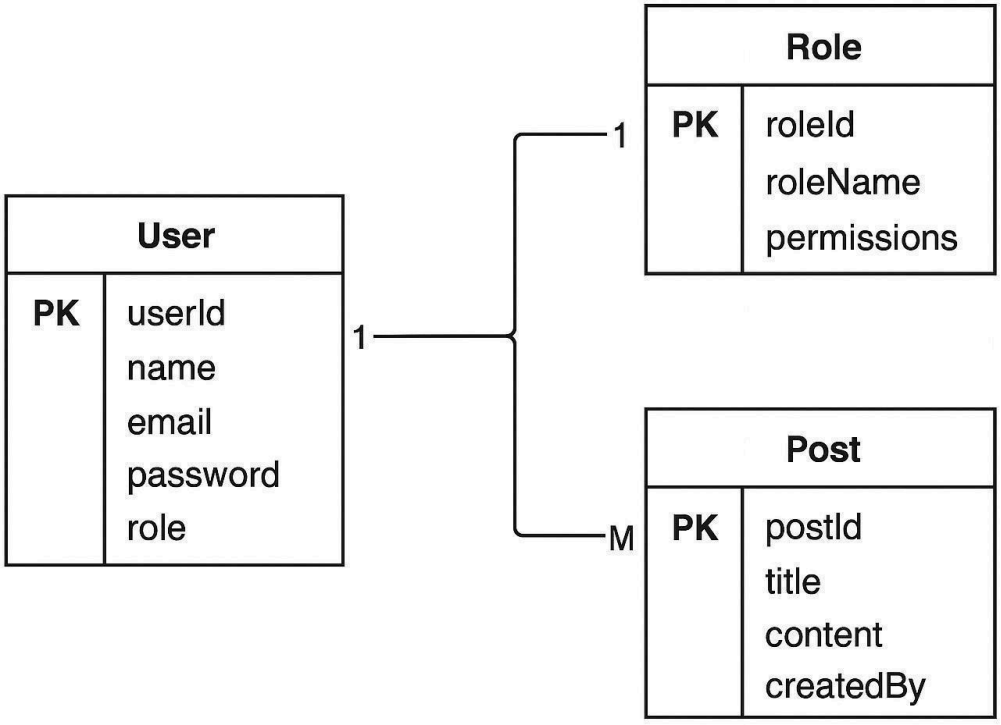
Hardware Requirements

- Processor: Intel i5 or higher
- RAM: Minimum 8 GB
- Storage: 20 GB available
- OS: Windows / Linux / macOS

Software Requirements

- Node.js (v18 or higher)
- MongoDB Community Server
- React.js (v18)
- Express.js
- Visual Studio Code
- Postman for API testing

ER Diagram



Database Schema

Users Collection

```
{
 "_id": "ObjectId",
 "name": "John Doe",
 "email":       "john@example.com",
 "password":        "$2b$10$hashed",
 "role": "editor"
}
```

Roles Collection

```
{
 "_id": "ObjectId",
 "roleName": "editor",
 "permissions": ["read", "create", "update_own"]
}
```

Posts Collection

```
{
 "_id": "ObjectId", "title":
 "Sample Post",
 "content": "This is an example post.", "createdBy":
 "user_id_ref"
}
```

Backend Implementation (Node.js + Express)

Authentication

- Login endpoint verifies credentials.
- JWT generated with user ID and role.
- Token attached in headers for API calls.

Authorization Middleware

Example:

```
function authorizeRoles(...allowedRoles) {
  return (req, res, next) => {
```

```
    if (!allowedRoles.includes(req.user.role)) {
      return res.status(403).json({ message: "Access denied" });
    }
    next();
  };
}
```

Ownership Check Example

```
const post = await Post.findById(req.params.id);
if (req.user.role === 'editor' && post.createdBy.toString() !== req.user.id) {
  return res.status(403).json({ message: "Not allowed to modify others' content" });
}
```

Front-End Screens

Role-Based UI Rendering

```
{user.role === 'admin' && <button onClick={deletePost}>Delete</button>}
{user.role === 'editor' && post.createdBy === user.id && <button>Edit</button>}
```

Protected Routes

```
<Route
  path="/admin"
  element={
    user.role === 'admin' ? <AdminPanel /> : <Navigate to="/unauthorized" />
  }
/>
```

JWT Token Structure

```
{
  "id": "647a1f...",
  "email": "user@example.com",
  "role": "editor",
  "iat": 1718807000,
  "exp": 1718810600
}
```

Output Screens and Reports

Below are the key **output screens** of the project that demonstrate the working of fine-grained Role-Based Access Control in the MERN application.

1. Login Page

- The user enters email and password.
- Upon successful login, a JWT token is generated and stored in localStorage.
- The dashboard is loaded based on the user's role.

---

2. Admin Dashboard

- Admin can view all posts created by any user.
- Admin has full access: Create, Update, Delete any record.

- “Manage Users” option is visible only for Admins.

---

3. Editor Dashboard

- Editors can create and modify their own posts.
- If they try to edit another user’s post, the system denies access with a 403

message.
- “Edit” and “Delete” buttons appear only for owned posts.

---

4. Viewer Dashboard

- Viewers have read-only access.
- They can see all posts but cannot create, edit, or delete.
- Buttons for restricted actions are disabled or hidden.

---

5. Unauthorized Access Page

- When a user tries to access a restricted route, they are redirected to this page.
- Displays message: “Access Denied – You do not have permission to view this page.”

---

6. Post Creation Page

- Editors and Admins can create posts through this form.
- Data is sent via POST request to /api/posts.

---

7. Post Management (Admin View)

- Admin sees all posts along with author information.
- Delete and Edit buttons are available for all posts*”)*

Limitations

1. Token Expiry & Re-login:
   Users must re-login after JWT expiry; no auto-refresh mechanism.
2. Static Role Management:
   Roles and permissions are hardcoded with no dynamic configuration.
3. No Audit Logging:
   Sensitive user actions are not recorded or tracked.
4. Limited Error Handling:
   API responses lack detailed, user-friendly feedback.
5. No Two-Factor Authentication (2FA):
   Login security relies only on credentials.

Future Scope

1. Dynamic Role Management:
   Add an admin interface for creating and managing roles and permissions dynamically.
2. Audit & Logging System:
   Record and monitor all critical user actions with timestamps.
3. Token Refresh Mechanism:

Implement refresh tokens for seamless session management.

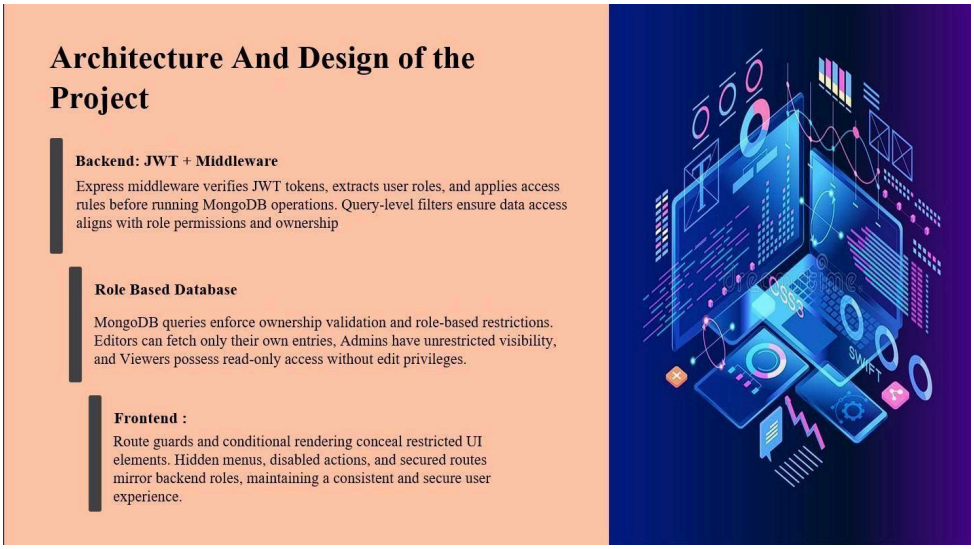
4. Two-Factor Authentication (2FA):
   Integrate optional OTP or Authenticator-based verification.
5. Enhanced UI/UX:
   Improve the interface with responsive, role-based dashboards using modern UI frameworks.

GitHub URL

https://github.com/Anish122316/full-stack-develope

ment-project-2025.git

PPT SLIDES:



Importance of RBAC in MERN

Security

Prevent unauthorized access to sensitive data and operations through role-based constraints at every layer.

Scalability

Manage permissions for growing teams without duplicating code—centralize rules in JWT claims and middleware.

Compliance

Demonstrate audit trails and enforce least-privilege access required by security standards and regulations.

Execution: Backend Process and Working

JWT Token

Embed user ID, role, and optionally resource ownership into the JWT payload. Sign using a secret key and transmit to the client.
userId:
unique identifier
role: "Admin", "Editor", or "Viewer"
ownedResources: array of IDs

Express Middleware

Validate JWT, extract claims, and attach user context to the request object. Authorization middleware verifies roles before executing route handlers.

Implementation: Frontend

MongoDB Ownership

Query filters enforce row-level access. Editors match query where userId equals their ID; Admins bypass filters.

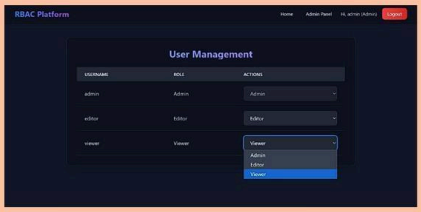
React Route Guards

Protected routes check user role and redirect unauthorized users. PrivateRoute component wraps admin/editor pages.

Conditional UI

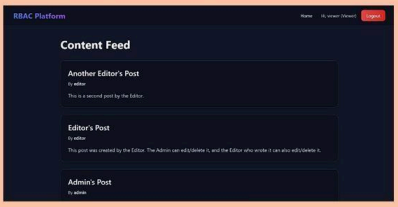
Disable edit/delete buttons for Viewers. Hide admin panels from non-admins. Show only permitted menu items based on decoded JWT.

Final Implementation:



Final Implementation:(role based controls)

Admin controls

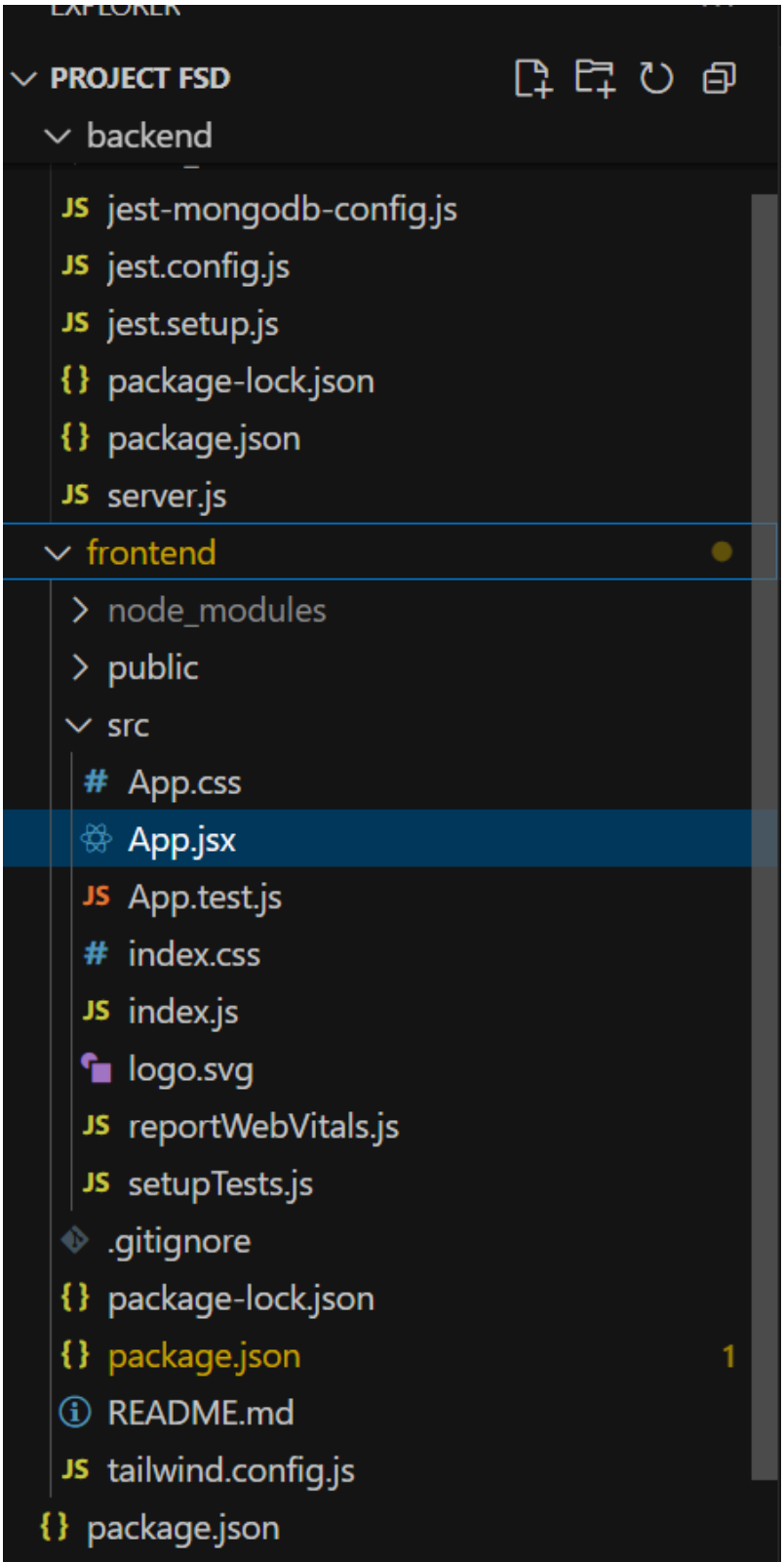


Viewer controls

Editor controls

Project Code

Project layout (recommended)



BACKEND

backend/server.js

```
const express = require('express'); const
mongoose = require('mongoose'); const
bcrypt = require('bcryptjs'); const jwt =
require('jsonwebtoken');
const cookieParser = require('cookie-parser');
const cors = require('cors');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
const { body, validationResult } =
require('express-validator');

// --- Configuration ---
const PORT = process.env.PORT || 5001;
const MONGO_URI =
'mongodb://localhost:27017/rbac-db';
const ACCESS_TOKEN_SECRET = 'your-
access-token-secret-key-CHANGE-ME'; const
REFRESH_TOKEN_SECRET = 'your-
refresh-token-secret-key-CHANGE-ME'; const
ACCESS_TOKEN_EXPIRATION = '15m';
const REFRESH_TOKEN_EXPIRATION = '7d';

const ROLES = { Admin:
  'Admin', Editor: 'Editor',
  Viewer: 'Viewer',
};

const PERMISSIONS = {
  [ROLES.Admin]: {
    content: ['create', 'read', 'update_all',
'delete_all'],
    users: ['read', 'update', 'delete'], auditlog:
    ['read'], // Added permission for
audit log
  },
  [ROLES.Editor]: {
    content: ['create', 'read', 'update_own',
'delete_own'],
    users: [],
    auditlog: [],
  },
  [ROLES.Viewer]: {
    content: ['read'], users:
    [],
    auditlog: [],
  },
};

const app = express();

// --- Middleware ---
app.use(cors({
  origin: 'http://localhost:3000',
  credentials: true,
}));
app.use(express.json()); app.use(cookieParser());
```

```
app.use(morgan('dev'));

// --- Database Connection ---
mongoose.connect(MONGO_
URI)
  .then(() => {
    console.log('MongoDB
    connected
successfully.');
    seedDatabase();
  })
  .catch((err) =>
console.error('MongoDB connection
error:', err));

// --- Mongoose Schemas ---
const userSchema = new
mongoose.Schema({ username: { type:
String, required: true, unique: true, index:
true },
  password: { type: String, required: true },
role: { type: String, enum:
Object.values(ROLES), default:
ROLES.Viewer
},
}, { timestamps: true });

userSchema.pre('save',                async
  function(next)    {    if    (this.isNew    ||
  this.isModified('password')) { try {
     const         salt         =         await
     bcrypt.genSalt(10);  this.password
     = await
bcrypt.hash(this.password, salt);
     console.log(`[Hashing]: Password
     for
'${this.username}' has been
     hashed.`); next();
   } catch (error) {
     next(error);
   }
  }
    els
    e {
    ne
    xt(
    );
  }
});

const User =
mongoose.model('User',
userSchema);

const contentSchema = new
  mongoose.Schema({ title: { type: String,
  required: true },
  body: { type: String, required: true },
author: { type:
mongoose.Schema.Types.ObjectId, ref:
'User', required: true },
}, { timestamps: true });

const Content =
mongoose.model('Content',
contentSchema);

      // --- ADD NEW SCHEMA FOR AUDIT
                                    LOG --
-
const auditLogSchema =
new mongoose.Schema({
  type: {
    type: String,
                  enum: ['ROLE_CHANGE',
                        'USER_LOGIN',
```

```
'USER_REGISTER'],
    required: true
  },
  actor: { type: mongoose.Schema.Types.ObjectId,
ref: 'User' },
// Who did the action
  target: { type:
mongoose.Schema.Types.ObjectId, ref: 'User' },
// Who was affected
  details: {
    oldRole: String,
    newRole: String,
  }
}, { timestamps: true });

const AuditLog = mongoose.model('AuditLog',
auditLogSchema);
// --- END OF NEW SCHEMA ---

// --- Utility Functions ---
const createToken = (payload, secret, expiresIn)
=> {
  return jwt.sign(payload, secret, { expiresIn });
};

const sendAuthTokens = (res, user) => {
const userData = { id: user._id, username:
user.username, role: user.role };

  const accessToken = createToken(userData,
ACCESS_TOKEN_SECRET,
ACCESS_TOKEN_EXPIRATION);
  const refreshToken = createToken({ id: user._id
}, REFRESH_TOKEN_SECRET,
REFRESH_TOKEN_EXPIRATION);

  res.cookie('refreshToken', refreshToken, {
    httpOnly: true,
    secure: process.env.NODE_ENV ===
'production',
    sameSite: 'strict',
    maxAge: 7 * 24 * 60 * 60 * 1000,
  });

  res.json({
    message: 'Authentication successful',
    accessToken,
    user: userData,
  });
};

// --- Rate Limiter ---
const authLimiter = rateLimit({ windowMs:
  15 * 60 * 1000,
  max: 10,
  message: 'Too many authentication attempts
from this IP, please try again after 15 minutes',
});

// --- Auth Middleware ---
const authenticate = (req, res, next) => {
const authHeader =
req.headers['authorization'];
```

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

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

  try {
    const decoded =
jwt.verify(token,
ACCESS_TOKEN_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: 'Token is
not valid.' });
  }
};

const authorize = (action) => {
  return (req, res, next) => {
    const [resource,
requiredPermission] =
action.split(':');
    const userRole =
    req.user.role; const
    userPermissions =
PERMISSIONS[userRole]?.[resource] || [];

    const hasPermission =
userPermissions.includes(requiredPermissio
n) ||
                        userPermissions.includes(`${r
equ iredPermission}_all`);

    if
      (!hasPermission
      ) { if
(userPermissions.includes(`${requiredPermi
ssio n}_own`)) {
        req.isOwnershipCheckRequired =
        true; return next();
      }
      return res.status(403).json({ message:
'Forbidden: You do not have permission.'
});
    }

    next();
  };
};

// --- API Routes ---

// 1. Auth Routes
app.post('/api/auth/register',
authLimiter, [
  body('username', 'Username must be at least 3
characters long').isLength({ min: 3
}).trim().escape(),
  body('password', 'Password must be at
least 6 characters long').isLength({ min: 6 }),
body('role', 'A valid role is
required').isIn([ROLES.Editor,
ROLES.Viewer]), // Validate incoming role
], async (req, res) => {

  const errors =
  validationResult(req); if
  (!errors.isEmpty()) {
```

```
    return res.status(400).json({ errors:
errors.array() });
  }

  try {
    const { username, password, role } = req.body;
// Get role from request
    let user = await User.findOne({ username }); if
    (user) {
      return res.status(400).json({ message: 'User
already exists.' });
    }

    // VALIDATE THE ROLE - Do not allow
registering as Admin
    if (role === ROLES.Admin &&
process.env.NODE_ENV !== 'test') {
      return res.status(400).json({ message: 'Cannot
register as Admin.' });
    }

    user = new User({
      username, password,
      role: role || ROLES.Viewer // Use provided
role, default to Viewer
    });
    await user.save();

    // --- ADD AUDIT LOG ---
    await new AuditLog({
      type: 'USER_REGISTER',
      actor: user._id, // User is the actor
      target: user._id // and the target
    }).save();

    sendAuthTokens(res, user);
  } catch (error) {
    res.status(500).json({ message: 'Server error
during registration.', error: error.message });
  }
});

app.post('/api/auth/login', authLimiter, [
body('username', 'Username is
required').notEmpty().trim().escape(),
body('password', 'Password is
required').notEmpty(),
], async (req, res) => {

  const errors = validationResult(req); if
  (!errors.isEmpty()) {
    return res.status(400).json({ errors:
errors.array() });
  }

  try {
    const { username, password } = req.body;
    const user = await User.findOne({ username
});
    if (!user) {
      console.log(`[Login Attempt]: User
'${username}' not found.`);
```

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

    console.log(`[Login Attempt]: Found
user '${username}'. Comparing
passwords...`);
    const isMatch = await
bcrypt.compare(password, user.password);

    if (!isMatch) {
      console.log(`[Login Attempt]: Password
for '${username}' does NOT match.`);
      return res.status(400).json({
message: 'Invalid credentials.' });
    }

    console.log(`[Login Attempt]: Password
for '${username}' matched. Logging in.`);

    // --- ADD AUDIT LOG ---
    await new AuditLog({
      type:
      'USER_LOGIN',
      actor:
      user._id,
      target:
      user._id
    }).save();

    sendAuthTokens(res, user);
  } catch (error) {
    res.status(500).json({ message: 'Server
error during login.', error: error.message });
  }
});

app.post('/api/auth/refresh', (req, res)
=> { const refreshToken =
req.cookies.refreshToken;
  if (!refreshToken) {
    return res.status(401).json({ message: 'No
refresh token provided.' });
  }

  try {
    const decoded =
jwt.verify(refreshToken,
REFRESH_TOKEN_SECRET);

    User.findById(decoded.id).then(user
      => { if (!user) {
        return res.status(401).json({
message: 'Invalid refresh token.' });
      }

      const userData = { id: user._id,
username: user.username, role: user.role };
      const accessToken =
createToken(userData,
ACCESS_TOKEN_SECRET,
ACCESS_TOKEN_EXPIRATION);

      res.json({ accessToken });
    });
  } catch (err) {
    res.status(401).json({ message:
'Invalid refresh token.' });
  }
```

```
});

app.post('/api/auth/logout', (req, res) => {
  res.cookie('refreshToken', '', { httpOnly:
  true,
    expires: new Date(0),
  });
  res.status(200).json({ message: 'Logged out
successfully.' });
});

// 2. Content Routes app.use('/api/content',
authenticate);

app.get('/api/content', authorize('content:read'),
async (req, res) => {
  try {
    const content = await
Content.find().populate('author',
'username').sort({ createdAt: -1 });
    res.json(content);
  } catch (error) {
    res.status(500).json({ message: 'Server error.',
error: error.message });
  }
});

app.get('/api/content/:id',
authorize('content:read'), async (req, res) => { try
{
    const content = await
Content.findById(req.params.id).populate('auth
or', 'username');
    if (!content) {
      return res.status(404).json({ message: 'Content
not found.' });
    }
    res.json(content);
  } catch (error) {
    res.status(500).json({ message: 'Server error.',
error: error.message });
  }
});

app.post('/api/content',
authorize('content:create'), [
body('title', 'Title is
required').notEmpty().trim().escape(),
body('body', 'Body is
required').notEmpty().trim().escape(),
], async (req, res) => {

  const errors = validationResult(req); if
  (!errors.isEmpty()) {
    return res.status(400).json({ errors:
errors.array() });
  }

  try {
    const { title, body } = req.body;

const content = new Content({
title,
  body,
```

```
    author: req.user.id,
   });
   await content.save();
   res.status(201).json(conte
   nt);
  } catch (error) {
   res.status(500).json({ message: 'Server
error.', error: error.message });
  }
});

app.put('/api/content/:id',
authorize('content:update'), [
body('title', 'Title is
required').notEmpty().trim().escap
e(), body('body', 'Body is
required').notEmpty().trim().escap
e(),
], async (req, res) => {

  const errors =
  validationResult(req); if
  (!errors.isEmpty()) {
   return res.status(400).json({
errors: errors.array() });
  }

  try {
   const { title, body } =
   req.body; let content =
   await
Content.findById(req.params.id);

   if (!content) {
    return res.status(404).json({
message: 'Content not found.' });
   }

   if (req.isOwnershipCheckRequired
&& content.author.toString() !==
req.user.id) {
    return res.status(403).json({ message:
'Forbidden: You can only update your own
content.' });
   }

   content.title =
   title; content.body
   = body; await
   content.save();

   res.json(content);
  } catch (error) {
   res.status(500).json({ message: 'Server error.',
error: error.message });
  }
});

app.delete('/api/content/:id',
authorize('content:delete'), async (req, res)
=> { try {
   const content = await
Content.findById(req.params.
id);

   if (!content) {
    return res.status(404).json({
message: 'Content not found.' });
   }
```

```
    if (req.isOwnershipCheckRequired &&
content.author.toString() !== req.user.id) {
      return res.status(403).json({ message:
'Forbidden: You can only delete your own
content.' });
    }

    await
Content.findByIdAndDelete(req.params.id);

    res.json({ message: 'Content deleted
successfully.' });
  } catch (error) {
    res.status(500).json({ message: 'Server error.',
error: error.message });
  }
});

// 3. Admin Routes app.use('/api/users',
authenticate, authorize('users:read'));

app.get('/api/users', async (req, res) => {
  try {
    const users = await User.find().select('-
password');
    res.json(users);
  } catch (error) {
    res.status(500).json({ message: 'Server error.',
error: error.message });
  }
});

 app.put('/api/users/:id/role',
authorize('users:update'), [ body('role',
'A valid role is
required').isIn(Object.values(ROLES)),
], async (req, res) => {

  const errors = validationResult(req); if
  (!errors.isEmpty()) {
    return res.status(400).json({ errors:
errors.array() });
  }

  try {
    const { role } = req.body;
    const user = await
User.findById(req.params.id);

    if (!user) {
      return res.status(404).json({ message: 'User
not found.' });
    }

    const oldRole = user.role; // Store old role
    user.role = role;
    await user.save();

    // --- ADD AUDIT LOG ---
    await new AuditLog({ type:
      'ROLE_CHANGE',
      actor: req.user.id, // The Admin who made
```

```
the change
      target: user._id, // The user who was
changed
      details: {
        oldRole: oldRole,
        newRole: user.role
      }
    }).save();

    res.json(user);
  } catch (error) {
    res.status(500).json({ message: 'Server
error.', error: error.message });
  }
});

// 4. Audit Log Route
app.get('/api/audit-logs',
authenticate,
authorize('auditlog:read'), async (req, res) =>
  { try {
    const logs = await AuditLog.find()
      .populate('actor', 'username') // Get
username of the actor
      .populate('target', 'username') //
Get username of the target
      .sort({ createdAt: -1 })       // Newest first
      .limit(20);                    // Get last 20 events

    // Format the data to perfectly match
the frontend's mock data structure
    const formattedLogs = logs.map(log => ({
      _id: log._id,
      timestamp:
      log.createdAt, type:
      log.type,
      adminUsername:
log.actor?.username, // Will be null if
actor/target deleted
      targetUsername:
      log.target?.username, oldRole:
      log.details?.oldRole, newRole:
      log.details?.newRole,
    }));

    res.json(formattedLogs);
  } catch (error) {
    res.status(500).json({ message: 'Server error
fetching audit logs.', error: error.message });
  }
});

// --- Server Start ---
if (process.env.NODE_ENV !== 'test') {
  app.listen(PORT, () => {
  console.log(`Server running on port
${PORT}`);
  });
}

// --- Database Seeding ---
async function
seedDatabase() { try {
    await User.deleteMany({});
    await
    Content.deleteMany({});
    await AuditLog.deleteMany({}); // Clear
old logs on restart
```

```
  const adminUser = new User({
    username: 'admin', password:
    'adminpassword', role: ROLES.Admin,
  });
  const editorUser = new User({
    username: 'editor', password:
    'editorpassword', role: ROLES.Editor,
  });
  const viewerUser = new User({
    username: 'viewer', password:
    'viewerpassword', role: ROLES.Viewer,
  });

  await     adminUser.save();     await
  editorUser.save();                 await
  viewerUser.save();
```

```
  const content1 = new Content({
    title: 'Admin\'s Post',
    body: 'This post was created by the Admin.
Only the Admin can edit or delete this.', author:
    adminUser._id,
  });
  const content2 = new Content({
    title: 'Editor\'s Post',
    body: 'This post was created by
the Editor. The Admin can
edit/delete it, and the Editor who
wrote it can also edit/delete it.',
    author: editorUser._id,
  });
  const content3 = new
    Content({ title:
    'Another Editor\'s
    Post',
    body: 'This is a second post by the
    Editor.', author: editorUser._id,
  });

  await
  content1.s
  ave();
  await
  content2.s
  ave();
  await
  content3.s
  ave();

  console.log('Database seeded
  successfully.');
 } catch (error) {
  console.error('Error seeding
database:', error.message);
 }
}

// --- ADD THIS AT THE VERY END ---
module.exports = app;
```

---

backend/package.json

```
{
 "name": "backend",
 "version": "1.0.0",
 "main": "server.js",
 "scripts": {
  "test": "cross-env NODE_ENV=test jest --
runInBand",
  "start": "node server.js"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "description": "",
 "dependencies": {
 "bcryptjs": "^3.0.2",
  "cookie-parser": "^1.4.7",
  "cors": "^2.8.5",
  "express": "^5.1.0",
  "express-rate-limit": "^8.1.0",
  "express-validator": "^7.3.0",
  "jsonwebtoken": "^9.0.2",
  "mongoose": "^8.19.2",
  "morgan": "^1.10.1"
 },
 "devDependencies": {
  "@shelf/jest-mongodb":
  "^4.3.2", "cross-env":
  "^7.0.3",
  "jest": "^29.7.0",
  "mongodb-memory-server":
  "^9.4.0", "supertest": "^7.0.0"
 }
}
```

FRONTEND

Use create-react-app or Vite. The code below assumes create-react-app.

frontend/src/App.jsx

```
import React, { useState, useEffect,
createContext, useContext } from 'react';
import {
  BrowserRouter,
  Routes,
  Route,
  Link,
  useNavigate,
```

```
  Outlet, useLocation,
  Navigate,
  useParams,
} from 'react-router-dom'; import
axios from 'axios';

// --- Configuration ---

const ROLES = {
  Admin: 'Admin',
  Editor: 'Editor',
  Viewer: 'Viewer',
};

// Simplified permission definitions (for frontend
checks)
const PERMISSIONS = {
  [ROLES.Admin]: {
    content: ['create', 'read', 'update_all',
'delete_all'],
    users: ['read', 'update', 'delete'], auditlog:
    ['read'],
  },
  [ROLES.Editor]: {
    content: ['create', 'read', 'update_own',
'delete_own'],
    users: [],
    auditlog: [],
  },
  [ROLES.Viewer]: {
    content: ['read'],
    users: [],
    auditlog: [],
  },
};

// --- Mock Data ---
const MOCK_AUDIT_LOGS = [
    { _id: '1', timestamp: '2023-10-27T10:30:00Z',
eventType: 'USER_LOGIN', user: { username:
'admin' }, details: { ip: '192.168.1.1' } },
    { _id: '2', timestamp: '2023-10-27T10:35:00Z',
eventType: 'ROLE_UPDATE', user: { username:
'admin' }, targetUser: { username: 'editor' },
details: { from: 'Editor', to: 'Viewer' }
},
    { _id: '3', timestamp: '2023-10-27T10:40:00Z',
eventType: 'USER_REGISTER', user: {
username: 'newUser' }, details: { role: 'Viewer' }
},
    { _id: '4', timestamp: '2023-10-27T09:00:00Z',
eventType: 'USER_LOGIN', user: { username:
'editor' }, details: { ip: '10.0.0.5' } },
    { _id: '5', timestamp: '2023-10-26T18:15:00Z',
eventType: 'ROLE_UPDATE', user: { username:
'admin' }, targetUser: { username: 'viewer' },
details: { from: 'Viewer', to: 'Editor' }
},
    { _id: '6', timestamp: '2023-10-27T11:00:00Z',
eventType: 'USER_LOGIN', user: { username:
'admin' }, details: { ip: '192.168.1.1' } },
];
```

```
// --- Theme Classes (Hardcoded Dark) ---
const themeClasses = {
  layout: "min-h-screen font-sans
transition- colors duration-300
bg-gray-900",
  nav: "sticky top-0 z-30 shadow-sm border-b
bg-black/20 backdrop-blur-lg text-white border-
white/10",
  card: "rounded-xl bg-black/20
backdrop-blur- lg border border-white/20
shadow-xl",
  input: "w-full px-4 py-2.5 border rounded-lg
focus:outline-none focus:ring-2 transition-all
bg- gray-700/50 border-white/20 text-white
placeholder-gray-300 focus:ring-blue-500
focus:border-blue-500",
  text: "text-white",
  textMuted:
  "text-gray-300",
  textLabel:
  "text-gray-100",
  textHeading: "text-3xl font-bold text-
transparent bg-clip-text bg-gradient-to-r
from- blue-400 to-purple-400",
  aurora: "absolute top-0 left-0 w-full
h-full overflow-hidden -z-10",
  solidBg: "bg-gray-800",
  solidBorder:
  "border-gray-700"
};

// --- Auth Context ---
const AuthContext = createContext(null);

// --- Axios API Client ---
const apiClient = axios.create({
  baseURL:
  'http://localhost:5001/api',
  withCredentials: true,
});

apiClient.interceptors.request.
  use( (config) => {
    const { accessToken } =
JSON.parse(localStorage.getItem('auth') ||
'{}');
    if (accessToken) {
      config.headers['Authorization'] =
      `Bearer
${accessToken}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

apiClient.interceptors.response.
  use( (response) => response,
  async (error) => {
    const originalRequest =
    error.config; if
    (error.response?.status === 401
    &&
!originalRequest._retry) {
      originalRequest._retry =
      true; try {
        const { data } = await
apiClient.post('/auth/refres
h');
        const newAccessToken = data.accessToken;

        let authData =
JSON.parse(localStorage.getItem('auth') ||
'{}');
```

```
        authData.accessToken = newAccessToken;
        localStorage.setItem('auth',
JSON.stringify(authData));

        originalRequest.headers['Authorization'] =
`Bearer ${newAccessToken}`;
        return apiClient(originalRequest);
      } catch (refreshError) {
        localStorage.removeItem('auth');
        window.location.href = '/login'; return
        Promise.reject(refreshError);
      }
    }
    return Promise.reject(error);
  }
);

// --- AuthProvider Component --- const
AuthProvider = ({ children }) => { const
[user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => { try
    {
      const authData =
JSON.parse(localStorage.getItem('auth'));
      if (authData && authData.user) {
        setUser(authData.user);
      }
    } catch (error) {
      console.error('Failed to parse auth data from
localStorage');
    }
    setLoading(false);
  }, []);

  const login = (userData) => {
    const authData = {
      accessToken: userData.accessToken,
      user: userData.user,
    };
    localStorage.setItem('auth',
JSON.stringify(authData));
    setUser(userData.user);
  };

  const logout = async () => { try
    {
      await apiClient.post('/auth/logout');
    } catch (error) { console.error('Logout
      failed', error);
    }
    localStorage.removeItem('auth');
    setUser(null);
  };
  const value = { user, login, logout, loading };
  return (
    <AuthContext.Provider value={value}>
      {!loading && children}
    </AuthContext.Provider>
```

);

```
};

// --- Auth Hooks ---
const useAuth = () =>
{
  return useContext(AuthContext);
};

const usePermissions = () =>
  { const { user } = useAuth();

  const can = (action, subject)
    => { if (!user) return false;

    const [resource,
requiredPermission] = action.split(':');
    const userPermissions =
PERMISSIONS[user.role]?.[resource] ||
[];

    if
(userPermissions.includes(requiredPermiss
ion)
||
userPermissions.includes(`${requiredPermis
sion
}_all`)) {
      return true;
    }

    if
(userPermissions.includes(`${requiredPermi
ssio n}_own`)) {
      if
        (!subject
        ) {
        return
        true;
      }
      return subject.authorId === user.id;
    }

    return false;
  };

  const canRead = (resource = 'content') =>
can(`${resource}:read`);
  const canCreate = (resource = 'content') =>
can(`${resource}:create`);
  const canUpdate = (subject, resource =
'content') => can(`${resource}:update`,
subject); const canDelete = (subject, resource
=
'content') => can(`${resource}:delete`,
subject); const canManageUsers = () =>
can('users:read');
  const canViewAuditLog = () =>
can('auditlog:read');

  return    {    can,    canRead,    canCreate,
canUpdate, canDelete, canManageUsers,
canViewAuditLog, userRole: user?.role,
userId: user?.id };
};

// --- UI Components ---

// Modal
const Modal = ({ title, children, onClose }) =>
  { return (
    <div className="fixed inset-0 bg-black/60
backdrop-blur-sm z-40 flex items-center
justify-
```

```
center p-4">
    <div className={`relative w-full max-w-md
p-6 ${themeClasses.card}`}>
      <h3 className={`text-xl font-semibold mb- 4
${themeClasses.text}`}>{title}</h3>
      <div
className={themeClasses.textMuted}>
       {children}
      </div>
      <button onClick={onClose}
       className="absolute top-4 right-4 text-
gray-400 hover:text-gray-200"
      >
       <svg className="w-6 h-6" fill="none"
stroke="currentColor" viewBox="0 0 24
24"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth="2"
d="M6 18L18 6M6 6l12 12"></path></svg>
      </button>
    </div>
   </div>
  );
};

// Aurora Background
const AuroraBackground = () => { return
  (
   <div className={themeClasses.aurora}>
    <style>{`
     .aurora-blur { filter:
      blur(100px);
     }
     .aurora-1 { position:
      absolute; width:
      500px; height:
      500px;
      background: radial-gradient(circle,
rgba(139, 92, 246, 0.4) 0%, rgba(139, 92, 246, 0)
70%);
      animation: aurora-anim-1 20s infinite
alternate;
     }
     .aurora-2 { position:
      absolute; width:
      400px; height:
      400px;
      background: radial-gradient(circle, rgba(59,
130, 246, 0.4) 0%, rgba(59, 130, 246, 0)
70%);
      animation: aurora-anim-2 22s infinite
alternate;
     }
     .aurora-3 { position:
      absolute; width:
      300px; height:
      300px;
      background: radial-gradient(circle,
rgba(236, 72, 153, 0.4) 0%, rgba(236, 72, 153, 0)
70%);
      animation: aurora-anim-3 18s infinite
alternate;
     }
```

```
      @keyframes
       aurora-anim-1 { 0% {
       top: 10%; left: 10%; }
       100% { top: 30%; left: 60%; }
      }
      @keyframes
       aurora-anim-2 { 0% {
       top: 50%; left: 70%; }
       100% { top: 40%; left: 20%; }
      }
      @keyframes
       aurora-anim-3 { 0% {
       top: 80%; left: 40%; }
       100% { top: 60%; left: 80%; }
      }
    `}</style>
    <div className="aurora-blur">
     <div className="aurora-1"></div>
     <div className="aurora-2"></div>
     <div className="aurora-3"></div>
    </div>
   </div>
  );
};

// --- Page Components ---

// Layout
const Layout = () => {
  const { user, logout } = useAuth();
  const navigate = useNavigate();
  const { canManageUsers } = usePermissions();

  const handleLogout = async () =>
   { await logout();
   navigate('/login');
  };

  return (
   <div className={themeClasses.layout}>
    <AuroraBackground />
    <nav className={themeClasses.nav}>
     <div className="max-w-7xl mx-auto
px-4 sm:px-6 lg:px-8">
      <div className="flex justify-between h-
16">
       <div className="flex items-center">
        <Link to="/" className="text-2xl
font- bold text-transparent bg-clip-text
bg-gradient- to-r from-blue-500
to-purple-500">
         RBAC Platform
        </Link>
       </div>
       <div className="flex items-center space-
x-4">
        <Link to="/"
className="text-gray-300 hover:text-white
px-3 py-2 rounded-md text-sm font-medium
transition-colors">
         Home
        </Link>
        {canManageUsers() && (
         <Link to="/admin"
className="text- gray-300 hover:text-white
px-3 py-2 rounded- md text-sm
font-medium transition-colors">
          Admin Panel
         </Link>
```

```
          )}
          {user ? (
            <>
              <span
className={`${themeClasses.textMuted} text-
sm`}>Hi, {user.username} ({user.role})</span>
              <button onClick={handleLogout}
                className="bg-gradient-to-r from-
red-500 to-red-600 text-white px-4 py-2
rounded-lg text-sm font-medium shadow-md
hover:from-red-600 hover:to-red-700 transition-
all duration-300 transform hover:scale-105"
              >
                Logout
              </button>
            </>
          ) : (
            <>
              <Link to="/login"
                className="text-gray-300 bg-
white/10 hover:bg-white/20 px-4 py-2 rounded-
lg text-sm font-medium transition-all duration-
300"
              >
                Login
              </Link>
              <Link
                to="/register"
                className="bg-gradient-to-r from-
blue-500 to-blue-600 text-white px-4 py-2
rounded-lg text-sm font-medium shadow-md
hover:from-blue-600 hover:to-blue-700
transition-all duration-300 transform
hover:scale-105"
              >
                Sign Up
              </Link>
            </>
          )}
        </div>
      </div>
    </nav>
    <main className="py-10 relative z-10">
      <div className="max-w-7xl mx-auto px-4
sm:px-6 lg:px-8">
        <Outlet />
      </div>
    </main>
  </div>
 );
};

// Home Page (Content List)
const HomePage = () => {
 const [content, setContent] = useState([]);
 const [error, setError] = useState(null);
 const { canCreate } = usePermissions();

 const fetchContent = async () => {
   try {
```

```
      setError(null);
      const { data } = await
apiClient.get('/content');
      setContent(data);
    } catch (err) {
      setError('Failed to fetch
      content.'); console.error(err);
    }
  };

  useEffect(()
    => {
    fetchConten
    t();
  }, []);

  if (error) {
    return <p className="text-red-500 text-
center">{error}</p>;
  }

  return (
    <div className="max-w-4xl mx-auto">
      <div className="flex justify-between
items- center mb-8">
        <h1 className={`text-4xl font-bold
${themeClasses.text}`}>Content Feed</h1>
        {canCreate() && (
          <Link
            to="/creat
            e"
            className="bg-gradient-to-r from-
green-500 to-green-600 text-white px-5
py-2.5 rounded-lg shadow-md
hover:from-green-600 hover:to-green-700
transition-all duration-300 transform
hover:scale-105"
          >
            + Create New Post
          </Link>
        )}
      </div>
      <div className="space-y-6">
        {content.length > 0 ? content.map((item) =>
(
          <ContentItem key={item._id}
item={item} onDelete={fetchContent} />
        )) : (
          <p
className={`${themeClasses.textMuted}
text- center`}>No content yet. Be the first to
create a post!</p>
        )}
      </div>
    </div>
  );
};

// Content Item
const ContentItem = ({ item, onDelete }) => {
  const { canUpdate, canDelete } =
usePermissions();
  const [showConfirm, setShowConfirm] =
useState(false);
  const [deleteError,
setDeleteError] = useState(null);
```

```
  const subject = { authorId: item.author?._id };
  const canEdit = canUpdate(subject);
  const canRemove = canDelete(subject);

  const handleDelete = async () => {
   try {
    setDeleteError(null); await
apiClient.delete(`/content/${item._id}`);
    setShowConfirm(false);
    onDelete();
   } catch (err) {
    setDeleteError('Failed to delete post.');
    console.error(err);
    setShowConfirm(false);
   }
  };

  return (
   <>
    <div className={`p-6 transition-all
duration-300 ${themeClasses.card}`}>
     <div className="flex justify-between
items-start">
      <div>
       <h2 className={`text-2xl font-semibold
${themeClasses.text} mb-2`}>{item.title}</h2>
       <p
className={`${themeClasses.textMuted} text- sm
mb-4`}>
        By <span className={`font-medium
${themeClasses.textLabel}`}>{item.author?.user
name || 'Unknown Author'}</span>
       </p>
      </div>
      <div className="flex space-x-2 flex-
shrink-0 ml-4">
       {canEdit && (
        <Link to={`/edit/${item._id}`}
         className="text-sm bg-gradient-to-r
from-yellow-400 to-yellow-500 text-white px-4
py-2 rounded-lg shadow-md hover:from-yellow-
500 hover:to-yellow-600 transition-all duration-
300 transform hover:scale-105"
         title="Edit this post"
        >
         Edit
        </Link>
       )}
       {canRemove && (
        <button
         onClick={() => setShowConfirm(true)}
         className="text-sm bg-gradient-to-r
from-red-500 to-red-600 text-white px-4 py-2
rounded-lg shadow-md hover:from-red-600
hover:to-red-700 transition-all duration-300
transform hover:scale-105"
         title="Delete this post"
        >
         Delete
        </button>
                                                )}
```

```
        </div>
       </div>
       <p
className={`${themeClasses.textMut
ed}
leading-relaxed`}>{item.body}</p>
       {deleteError && <p
className="text-red- 500 text-sm
mt-4">{deleteError}</p>}
     </div>

     {showConfirm && (
       <Modal title="Delete Post?" onClose={()
=> setShowConfirm(false)}>
         <p>Are you sure you want to delete
the post titled "{item.title}"? This action
cannot be undone.</p>
         <div className="flex justify-end
space-x- 3 mt-6">
           <button
             onClick={() =>
             setShowConfirm(false)}
             className="px-4 py-2 rounded-lg
             bg-
gray-200 text-gray-800 hover:bg-gray-300"
           >
             Cancel
           </button>
           <button
             onClick={handleDele
             te}
             className="px-4 py-2 rounded-lg
bg- red-600 text-white hover:bg-red-700"
           >
             Delete
           </button>
         </div>
       </Modal>
     )}
   </>
  );
};

// Form Wrapper Component
const AuthFormWrapper = ({ title, children })
=> {
  return (
    <div className={`max-w-md mx-auto
p-8 md:p-10 ${themeClasses.card}`}>
      <h2 className={`text-center mb-8
${themeClasses.textHeading}`}>
        {title}
      </h2>
      {children}
    </div>
  );
};

// Login Page
const LoginPage = () => {
const [username, setUsername] =
useState(''); const [password,
setPassword] = useState(''); const [error,
setError] = useState(null);
const { login } = useAuth();
const navigate =
useNavigate();

const handleSubmit = async (e) => {
  e.preventDefault();
```

```
    setError(null); try
    {
      const { data } = await
apiClient.post('/auth/login', { username, password
});
      login(data);
      if (data.user.role === ROLES.Admin) {
        navigate('/admin');
      } else {
        navigate('/');
      }
    } catch (err) {
      setError(err.response?.data?.message ||
'Invalid username or password.');
      console.error(err);
    }
  };

  const fillDemoUser = (user, pass) => {
    setUsername(user); setPassword(pass);
  };

  return (
    <AuthFormWrapper title="Welcome
Back!">
      <form onSubmit={handleSubmit}>
        {error && <p className="text-red-500
text-center mb-4 text-sm">{error}</p>}
        <div className="mb-4">
          <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="username">Username</label>
          <input type="text"
            id="username"
            value={username}
            onChange={(e) =>
setUsername(e.target.value)}
            className={themeClasses.input}
          />
        </div>
        <div className="mb-6">
          <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="password">Password</label>
          <input
            type="password"
            id="password"
            value={password}
            onChange={(e) =>
setPassword(e.target.value)}
            className={themeClasses.input}
          />
        </div>
        <button
          type="submit"
          className="w-full bg-gradient-to-r from-
blue-500 to-blue-600 text-white px-4 py-2.5
rounded-lg shadow-md hover:from-blue-600
hover:to-blue-700 transition-all duration-300
transform hover:scale-105"
        >
```

```
        Login
      </button>
    </form>
    <div className="mt-6 text-center">
      <p className={`mb-3
${themeClasses.textMuted} text-sm`}>Or try
a demo user:</p>
      <div className="grid grid-cols-3 gap-3">
        <button
          onClick={() =>
fillDemoUser('admin', 'adminpassword')}
          className="px-4 py-2 rounded-lg
text- white font-medium bg-gradient-to-r
from- purple-500 to-indigo-500 shadow-md
hover:scale-105 transform transition-all"
        >
          Admin
        </button>
        <button
          onClick={() =>
fillDemoUser('editor', 'editorpassword')}
          className="px-4 py-2 rounded-lg
text- white font-medium bg-gradient-to-r
from-cyan- 500 to-blue-500 shadow-md
hover:scale-105 transform transition-all"
        >
          Editor
        </button>
        <button
          onClick={() =>
fillDemoUser('viewer', 'viewerpassword')}
          className="px-4 py-2 rounded-lg
text- white font-medium bg-gradient-to-r
from-green- 500 to-teal-500 shadow-md
hover:scale-105 transform transition-all"
        >
          Viewer
        </button>
      </div>
    </div>
    <p className={`mt-6 text-center text-sm
${themeClasses.textMuted}`}
      > Don't have an
      account?{' '}
      <Link to="/register" className="font-
medium text-blue-500 hover:text-blue-400">
        Sign Up
      </Link>
    </p>
  </AuthFormWrapper>
 );
};

// Register Page
const RegisterPage = () => {
  const [username, setUsername] =
useState(''); const [password, setPassword]
= useState(''); const [role, setRole] =
useState(ROLES.Viewer);
  const [error, setError] = useState(null);
  const { login } = useAuth();
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
```

```
    e.preventDefault();
    setError(null);
    try {
      const { data } = await
apiClient.post('/auth/register', { username,
password, role });
      login(data);
      navigate('/');
    } catch (err) {
      setError(err.response?.data?.message ||
'Failed to register. Username might be taken.');
      console.error(err);
    }
  };

  return (
    <AuthFormWrapper title="Create Your
Account">
      <form onSubmit={handleSubmit}>
        {error && <p className="text-red-500
text-center mb-4 text-sm">{error}</p>}
        <div className="mb-4">
          <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="username">Username</label>
          <input type="text"
            id="username"
            value={username}
            onChange={(e) =>
setUsername(e.target.value)}
            className={themeClasses.input}
          />
        </div>
        <div className="mb-4">
          <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="password">Password</label>
          <input
            type="password"
            id="password"
            value={password}
            onChange={(e) =>
setPassword(e.target.value)}
            className={themeClasses.input}
          />
        </div>

        <div className="mb-6">
          <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="role">Register as:</label>
          <select id="role"
            value={role}
            onChange={(e) => setRole(e.target.value)}
            className={themeClasses.input}
          >
            <option className="bg-gray-700 text-
white" value={ROLES.Viewer}>Viewer</option>
            <option className="bg-gray-700 text-
white" value={ROLES.Editor}>Editor</option>
          </select>
        </div>

        <button
          type="sub
          mit"
          className="w-full bg-gradient-to-r
from- green-500 to-green-600 text-white px-4
py-2.5 rounded-lg shadow-md
hover:from-green-600 hover:to-green-700
transition-all duration-300 transform
hover:scale-105"
        >
          Register
        </button>
      </form>
      <p className={`mt-6 text-center text-sm
${themeClasses.textMuted}`}>
        Already have an account?{'
        '}
        <Link to="/login" className="font-
medium text-blue-500
hover:text-blue-400">
          Login
        </Link>
      </p>
    </AuthFormWrapper>
  );
};

// AdminDashboard Page
const AdminDashboard = () => {
  const AdminButton = ({ to, title,
description, icon }) => (
    <Lin
      k
      to={
      to}
      className={`block p-6
transition-all duration-300 transform
hover:scale-105 hover:shadow-2xl
${themeClasses.card}`}
    >
      <div className="flex items-center
space-x- 4">
        <div className="p-3 rounded-full bg-
gradient-to-r from-blue-500 to-purple-500
text- white">
          {icon}
        </div>
        <div>
          <h3 className={`text-xl font-semibold
${themeClasses.text}`}>{title}</h3>
          <p
className={themeClasses.textMuted}>{des
cript ion}</p>
        </div>
      </div>
    </Link>
```

```
  );

  return (
    <div className="max-w-4xl mx-auto">
      <h2 className={`text-4xl font-bold
mb-8 text-center
${themeClasses.textHeading}`}>Ad
min Dashboard</h2>
      <div className="grid grid-cols-1
md:grid- cols-2 gap-6">
```

```
        <AdminButton to="/"
          title="Manage Content Feed"
          description="View, edit, and delete all
posts."
          icon={<svg className="w-6 h-6"
fill="none" stroke="currentColor" viewBox="0 0
24 24"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth="2"
d="M19 20H5a2 2 0 01-2-2V6a2 2 0 012-2h10a2
2 0 012 2v1m2 13a2 2 0 01-2-2V7m2 13a2 2 0
002-2V9a2 2 0 00-2-2h-2m-4-3H9M7 16h6M7
12h6M7 8h6"></path></svg>}
        />
        <AdminButton
          to="/admin/users"
          title="Manage Users"
          description="View all users and change
their roles."
          icon={<svg className="w-6 h-6"
fill="none" stroke="currentColor" viewBox="0 0
24 24"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth="2"
d="M12 4.354a4 4 0 110 5.292M15 21H3v-1a6 6
0 016-6h6m6 3a9 9 0 11-18 0 9 9 0 0118
0z"></path></svg>}
        />
        <AdminButton
          to="/admin/audit" title="View
          Audit Log"
          description="See a log of important
system events."
          icon={<svg className="w-6 h-6"
fill="none" stroke="currentColor" viewBox="0 0
24 24"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth="2"
d="M9 12h6m-6 4h6m2 5H7a2 2 0 01-2-2V5a2 2
0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0
01.293.707V19a2 2 0 01-2 2z"></path></svg>}
        />
      </div>
    </div>
  );
};

// Admin Page (User Management)
const AdminPage = () => {
  const [users, setUsers] = useState([]); const
[error, setError] = useState(null); const
[editingRole, setEditingRole] = useState({});
  const [updateError, setUpdateError] =
useState(null);
  const { userId: adminUserId } =
usePermissions();

  const fetchUsers = async () => { try
    {
      setError(null);
      const { data } = await apiClient.get('/users');
      setUsers(data);
    } catch (err) {
      setError('Failed to fetch users.');
```

```
    console.error(err);
   }
  };

  useEffect(()
   => {
   fetchUsers()
   ;
  }, []);

  const handleRoleChange = (userId, newRole)
=> {
   setEditingRole(prev => ({ ...prev,
[userId]: newRole }));
  };

  const handleSaveRole = async (userId) =>
   { const newRole = editingRole[userId];
   if (!newRole) return;

   setUpdateError(nu
   ll); try {
    await
apiClient.put(`/users/${userId}/role`, { role:
newRole });
     setEditingRole(prev => {
      const newState = { ...prev
      }; delete
      newState[userId]; return
      newState;
     });
     fetchUsers();
   } catch (err) {
     setUpdateError('Failed to update
     role.'); console.error(err);
   }
  };

  if (error) {
   return <p className="text-red-500 text-
center">{error}</p>;
  }

  return (
   <div className={`max-w-4xl mx-auto p-8
${themeClasses.card}`}>
    <h2 className={`text-center mb-6
${themeClasses.textHeading}`}>
User Management</h2>
    {updateError && <p
className="text-red- 500 text-center mb-4
text- sm">{updateError}</p>}
    <div className="overflow-x-auto">
     <table className="w-full table-auto
min- w-max">
      <thead>
     <tr className={`bg-gray-700/50 text-left
          ${themeClasses.textLabel} uppercase
                                    text-sm`}>
          <th className="px-6 py-3 font-
semibold">Username</th>
          <th className="px-6 py-3 font-
semibold">Role</th>
          <th className="px-6 py-3 font-
semibold">Actions</th>
         </tr>
        </thead>
        <tbody
```

```
className={themeClasses.textMuted}>
          {users.map((user) => (
            <tr key={user._id} className={`border- b
${themeClasses.solidBorder} hover:bg-gray-
700/50`}>
              <td className="px-6 py-
4">{user.username}</td>
              <td className="px-6 py-
4">{user.role}</td>
              <td className="px-6 py-4 flex items-
center space-x-3">
                <select value={editingRole[user._id]
                  ||
user.role}
                  onChange={(e) =>
handleRoleChange(user._id, e.target.value)}
                  disabled={user._id ===
adminUserId}
                  className={`${themeClasses.input}
${user._id === adminUserId ? 'bg-gray-600/50' :
''}`}
                >
                  {Object.values(ROLES).map((role)
=> (
                    <option key={role}
                      value={role}
                      className="bg-gray-700 text-
white"
                    >
                      {role}
                    </option>
                  ))}
                </select>
                {editingRole[user._id] &&
editingRole[user._id] !== user.role && (
                  <button onClick={()
                    =>
handleSaveRole(user._id)}
                    className="px-4 py-2 rounded-lg
bg-green-600 text-white text-sm hover:bg-green-
700"
                  >
                    Save
                  </button>
                )}
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  </div>
 );
};

// AuditLogPage
// AuditLogPage
const AuditLogPage = () => {
  const [logs, setLogs] = useState([]); const
  [error, setError] = useState(null);
  const [loading, setLoading] = useState(true);
```

```
  // Fetch function
  const fetchLogs = async () =>
   { try {
     setError(null);
     const { data } = await
apiClient.get('/audit- logs'); // calls
http://localhost:5001/api/audit-logs
     // Optional: if backend returns timestamps,
convert to ISO or leave as-is
     // Ensure newest first (backend already
sorts, but just in case)
     const sorted = Array.isArray(data) ?
data.sort((a, b) => new Date(b.timestamp
|| b.createdAt) - new Date(a.timestamp ||
a.createdAt)) : [];
     setLogs(sorted
     );
     setLoading(fal
     se);
   } catch (err) {
     console.error('Failed to fetch audit
logs', err);
     setError(err.response?.data?.messag
e || 'Failed to fetch audit logs.');
     setLoading(false);
   }
  };

  useEffect(()
   => {
   fetchLogs();

   // Polling interval: update every 5
seconds (tune as needed)
   const interval = setInterval(fetchLogs, 5000);

   // Cleanup on unmount
   return () => clearInterval(interval);
  }, []);

  const renderLogDetails = (log) => {
   // Match your backend fields
(server.js formats logs with type,
adminUsername, targetUsername,
oldRole, newRole)
   switch (log.type) {
     case 'ROLE_CHANGE':
       return `${log.adminUsername ||
'Someone'} changed ${log.targetUsername ||
'a user'}'s role from ${log.oldRole || 'N/A'} to
${log.newRole || 'N/A'}.`;
     case 'USER_LOGIN':
       return `${log.adminUsername ||
log.targetUsername || 'User'} logged in.`;
     case 'USER_REGISTER':
       return `${log.adminUsername ||
log.targetUsername || 'User'} registered a new
account.`;
     default:
       return 'Performed an action.';
   }
  };

  return (
   <div className={`max-w-4xl mx-auto p-8
${themeClasses.card}`}>
   <h2 className={`text-center mb-6
 ${themeClasses.textHeading}`}>Audit
                Log
```

```
(Recent)</h2>

    {loading && <p                          });
className={`${themeClasses.textM
uted} text- center`}>Loading audit
logs...</p>}
    {error && <p
className="text-red-500 text-
center mb-4 text-sm">{error}</p>}


    <div className="space-y-4">
      {logs.length > 0 ? logs.map(log => (
        <div key={log._id || log.id} className="p- 4
rounded-lg bg-gray-700/50">
          <p className={themeClasses.text}>
            <span className="font-semibold text-
blue-400">{log.adminUsername ||
log.targetUsername || 'System'}</span>
{renderLogDetails(log)}
          </p>
          <p className={`text-sm
${themeClasses.textMuted} mt-1`}>
            {new Date(log.timestamp ||
log.createdAt).toLocaleString()}
          </p>
        </div>
      )) : (
        !loading && <p
className={themeClasses.textMuted}>No audit
logs found.</p>
      )}
    </div>
  </div>
 );
};


// Form Wrapper for Content
const ContentFormWrapper = ({ title, children
}) => {
 return (
  <div className={`max-w-2xl mx-auto p-8
md:p-10 ${themeClasses.card}`}>
    <h2 className={`text-center mb-8
${themeClasses.textHeading}`}>
      {title}
    </h2>
    {children}
  </div>
 );
};

// CreatePostPage
const CreatePostPage = () => { const
 [title, setTitle] = useState('');
 const [body, setBody] = useState(''); const
 [error, setError] = useState(null); const
 navigate = useNavigate();
                                    }
                                    try {
                                     await apiClient.post('/content', {
                                     title, body

                                     navigate('/');
                                    } catch (err) {
                                     setError(err.response?.d
                                     ata?.message ||


const handleSubmit = async (e) => {
 e.preventDefault();
 setError(null);
 if (!title || !body) {
  setError('Title and body are required.');
  return;
```

```
'Failed to create post.');
    console.error(err);
  }
 };

 return (
  <ContentFormWrapper title="Create
New Post">
    <form onSubmit={handleSubmit}>
     {error && <p
className="text-red-500 text-center
mb-4 text-sm">{error}</p>}
      <div className="mb-4">
       <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="title">Title</label>
       <input
        type="te
        xt"
        id="title"
        value={tit
        le}
        onChange={(e)
=>
setTitle(e.target.value)
}
        className={themeClasses.input}
       />
      </div>
      <div className="mb-6">
       <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="body">Body</label>
       <textarea
        id="body"
        rows="10"
        value={body}
        onChange={(e)
        =>
setBody(e.target.value)}
        className={themeClasses.input}
       />
      </div>
      <button
       type="subm
       it"
       className="w-full bg-gradient-to-r
from- blue-500 to-blue-600 text-white px-4
py-2.5 rounded-lg shadow-md
hover:from-blue-600 hover:to-blue-700
transition-all duration-300 transform
hover:scale-105"
      >
       Publish Post
      </button>
    </form>
  </ContentFormWrapper>
 );
};
```

```
// EditPostPage
const EditPostPage = () => {
 const [title, setTitle] =
 useState(''); const [body, setBody]
 = useState('');
```

```
  const [error, setError] = useState(null); const
  [loading, setLoading] = useState(true); const
  navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
   const fetchPost = async () => { try {
      const { data } = await
apiClient.get(`/content/${id}`);
      setTitle(data.title);
      setBody(data.body);
      setLoading(false);
    } catch (err) {
      setError('Failed to fetch post data. You may
not have permission to edit this.');
      console.error(err);
      setLoading(false);
    }
   };
   fetchPost();
  }, [id]);

  const handleSubmit = async (e) => {
   e.preventDefault();
   setError(null);
   if (!title || !body) {
    setError('Title and body are required.');
    return;
   }
   try {
    await apiClient.put(`/content/${id}`, { title,
body });
    navigate('/');
   } catch (err) {
    setError(err.response?.data?.message ||
'Failed to update post.');
    console.error(err);
   }
  };

  if (loading) {
   return <div className={`text-center
${themeClasses.textMuted}`}>Loading
post...</div>;
  }

  if (error && !title) {
    return <p className="text-red-500 text-
center">{error}</p>;
  }

  return (
   <ContentFormWrapper title="Edit Post">
    <form onSubmit={handleSubmit}>
     {error && <p className="text-red-500
text-center mb-4 text-sm">{error}</p>}
      <div className="mb-4">
       <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="title">Title</label>
                                    <input type="text"
```

```
          id="title"
          value={title}
          onChange={(e
          ) =>
setTitle(e.target.value)}
          className={themeClasses.input}
        />
       </div>
       <div className="mb-6">
        <label className={`block mb-2 font-
medium ${themeClasses.textLabel}`}
htmlFor="body">Body</label>
        <textarea
         id="body"
         rows="10"
         value={body}
         onChange={(e
         ) =>
setBody(e.target.value)}
         className={themeClasses.input}
        />
       </div>
       <button
        type="sub
        mit"
        className="w-full bg-gradient-to-r
from- blue-500 to-blue-600 text-white px-4
py-2.5 rounded-lg shadow-md
hover:from-blue-600 hover:to-blue-700
transition-all duration-300 transform
hover:scale-105"
       >
        Update Post
       </button>
     </form>
   </ContentFormWrapper>
  );
};

// Unauthorized Page
const UnauthorizedPage = () => {
  return (
   <div className={`text-center p-10
max-w-lg mx-auto
${themeClasses.card}`}>
     <h1 className="text-5xl font-bold
text- transparent bg-clip-text
bg-gradient-to-r from- red-500
to-yellow-500">Access Denied</h1>
     <p className={`mt-6 text-xl
${themeClasses.textMuted}`}>You do not
have the required permissions to view this
page.</p>
     <Link to="/" className="mt-8
inline-block bg-gradient-to-r from-blue-500
to-blue-600 text- white px-6 py-3 rounded-lg
shadow-md hover:from-blue-600
hover:to-blue-700 transition-all
duration-300 transform hover:scale-105">
       Go Back to Home
     </Link>
   </div>
  );
};

// --- Protected Route Component ---
const ProtectedRoute = ({ allowedRoles })
  => { const { user } = useAuth();
  const location = useLocation();
  const { userRole } = usePermissions();
```

```
  if (!user) {
    return <Navigate to="/login"
state={{ from: location }} replace
/>;
  }

  if (allowedRoles &&
!allowedRoles.includes(userRole)
) {

    return <Navigate to="/unauthorized" state={{
from: location }} replace />;
  }

  return <Outlet />;
};

// --- App Component --- export
default function App() { return (
   <AuthProvider>
     <BrowserRouter>
       <Routes>
         <Route path="/" element={<Layout />}>

           {/* Public Routes */}
           <Route path="/login"
element={<LoginPage />} />
           <Route path="/register"
element={<RegisterPage />} />
           <Route path="/unauthorized"
element={<UnauthorizedPage />} />

           {/* Protected Routes (All logged-in users)
*/}
                                                  />
                                                  */}
                                                  <Route
                                                  element={<ProtectedRoute />}>
                                                    <Route index
                                                    element={<HomePage />}

                                                  </Route>

                                                  {/* Protected Routes (Editors &
                                                  Admins)

                                                  <Route
                                                  element={<ProtectedRoute
allowedRoles={[ROLES.Ad
min, ROLES.Editor]} />}>
             <Route path="create"
element={<CreatePostPage />}
/>
             <Route
path="edit/:id"
element={<EditPostPage />}
/>
           </Route>

           {/* Protected Routes (Admins Only) */}
           <Route element={<ProtectedRoute
allowedRoles={[ROLES.Admin]} />}>
             <Route path="admin"
element={<AdminDashboard />}
/>
             <Route path="admin/users"
element={<AdminPage />} />
             <Route path="admin/audit"
element={<AuditLogPage />} />
           </Route>

         </Route>
       </Routes>
     </BrowserRouter>
   </AuthProvider>
  );
}
```

README (run instructions)

Create a top-level README with these steps:

Prerequisites

- Node.js (v18+ recommended)
- MongoDB running locally or remote connection string

Backend

1. cd backend
2. cp .env.example .env and edit .env (set MONGO_URI and JWT_SECRET)

3. npm install
4. Seed demo roles & users: npm run seed (this seeds 3 users: alice@admin.com/admin123, bob@editor.com/editor123, charlie@viewer.com/viewer123)
5. npm run dev (or npm start) to run server on PORT (default 5000)

Frontend

1. cd frontend
2. npm install
3. (optional) create .env with REACT_APP_API_URL=http://localhost:5000/api

4. npm start (runs on http://localhost:3000)

Test flow

1. Login with seeded users (emails & passwords above)
2. Admin can create/edit/delete any
3. Editor can create and edit own posts only (cannot delete)
4. Viewer can only read

Extra notes / example cURL Login

```
curl -X POST http://localhost:5000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"bob@editor.com","password":"editor123"}'
```

Create Post (Editor)

```
curl -X POST http://localhost:5000/api/posts \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <TOKEN>" \
  -d '{"title":"Hello","content":"My post"}'
```

Get Posts

```
curl -H "Authorization: Bearer <TOKEN>" http://localhost:5000/api/posts
```

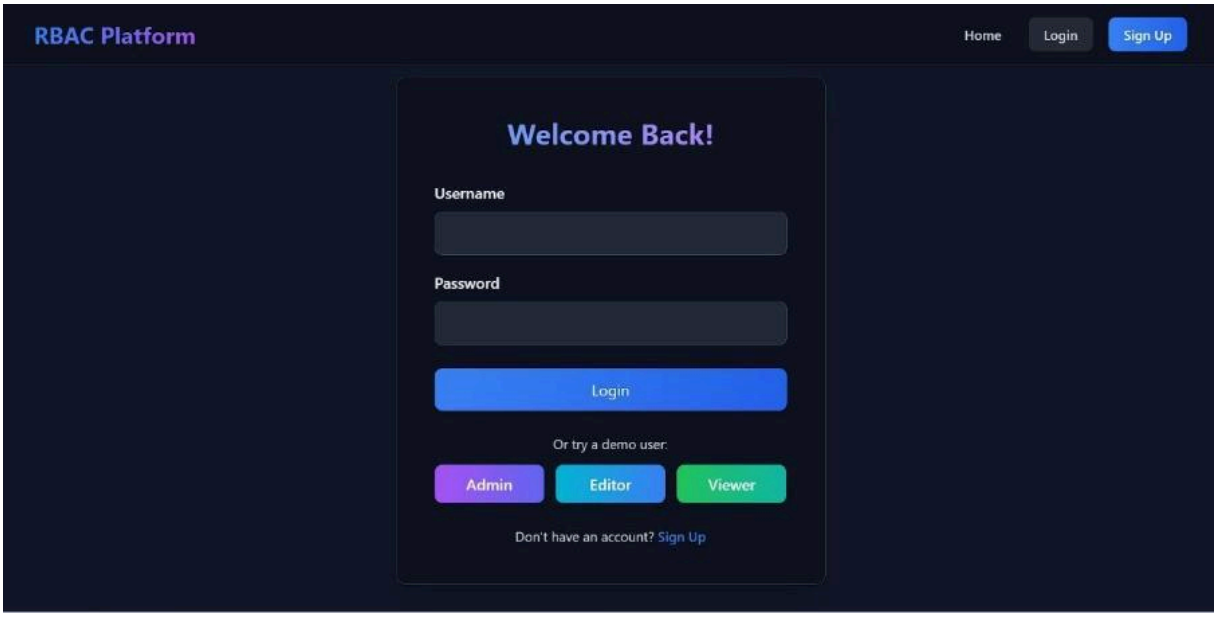
Security & design touches explained (quick)

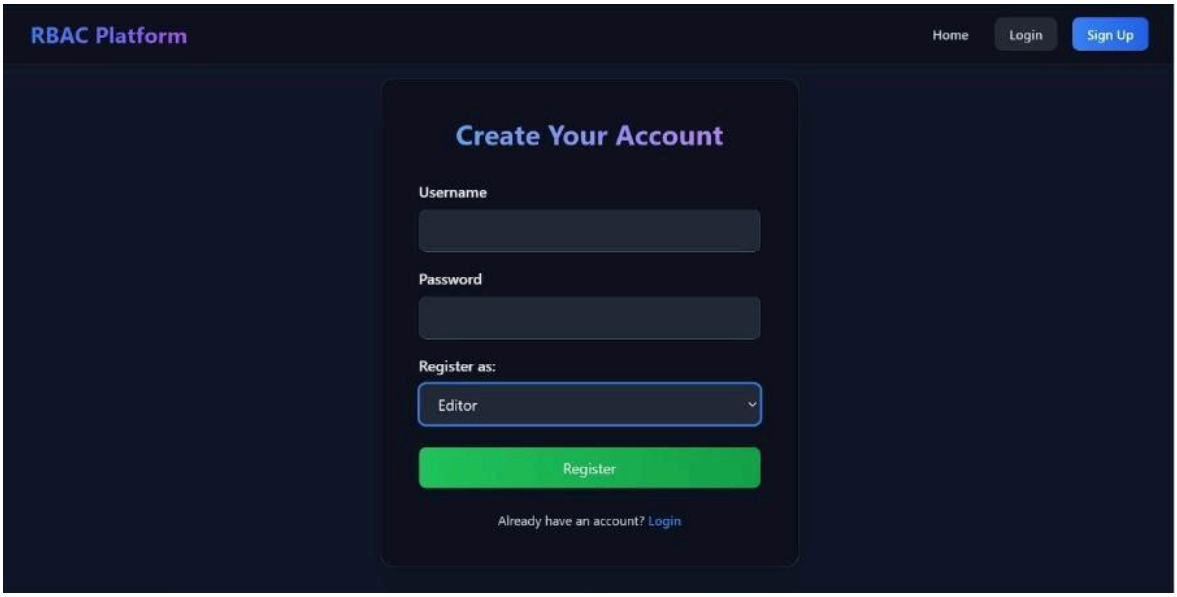
- JWT contains minimal claims: id, email, role. Short lived tokens recommended with refresh tokens for production.
- Backend enforces RBAC via middleware authorizeRoles() and row-level checks in controllers (post.createdBy).
- Query-level filters done in getPosts() to show an editor only their own posts.
- Frontend uses token decode to reflect UI controls and guards routes using ProtectedRoute. Buttons disabled/hidden based on user.role and isOwner.

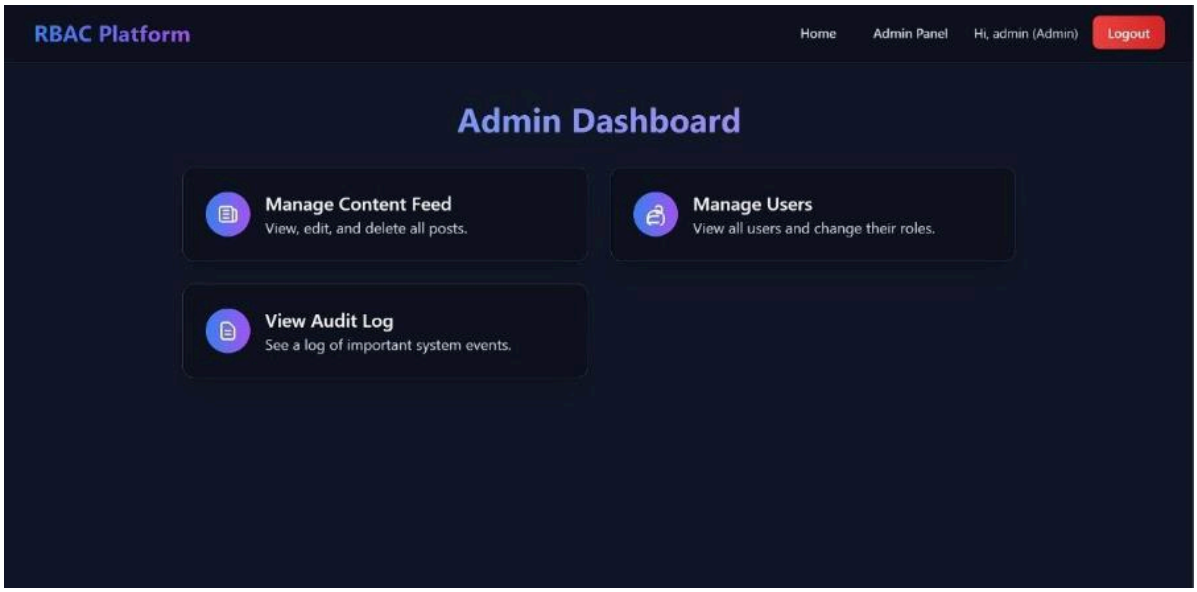
Limitations & improvements (recap)

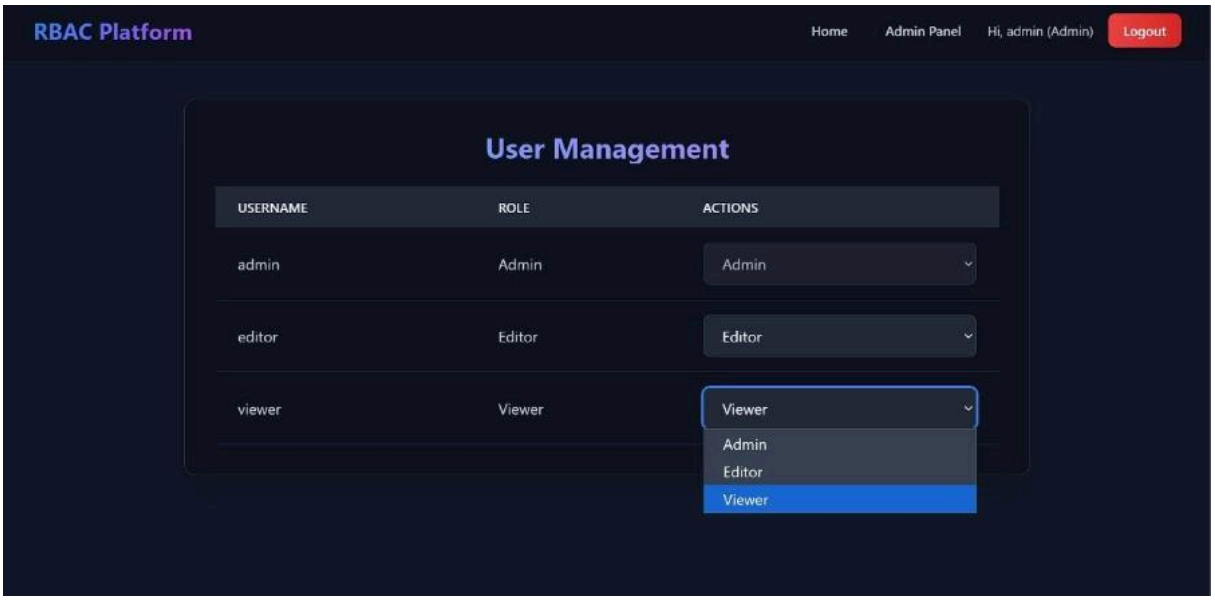
- No refresh tokens; add refresh token flow in production.
- Roles are strings saved on the user — consider persisting advanced permission sets in DB and fetching them.
- Action auditing is missing — add action logs.
- No rate limiting or brute-force protection — add to production.
- Password reset, email verification, 2FA optionally required.

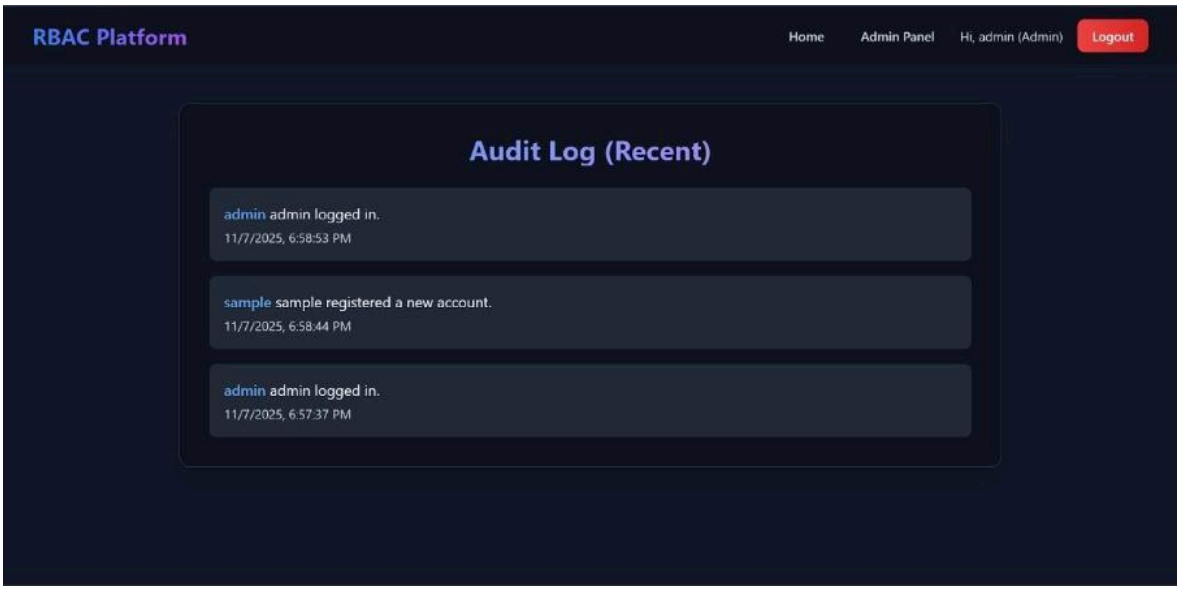
Project Output

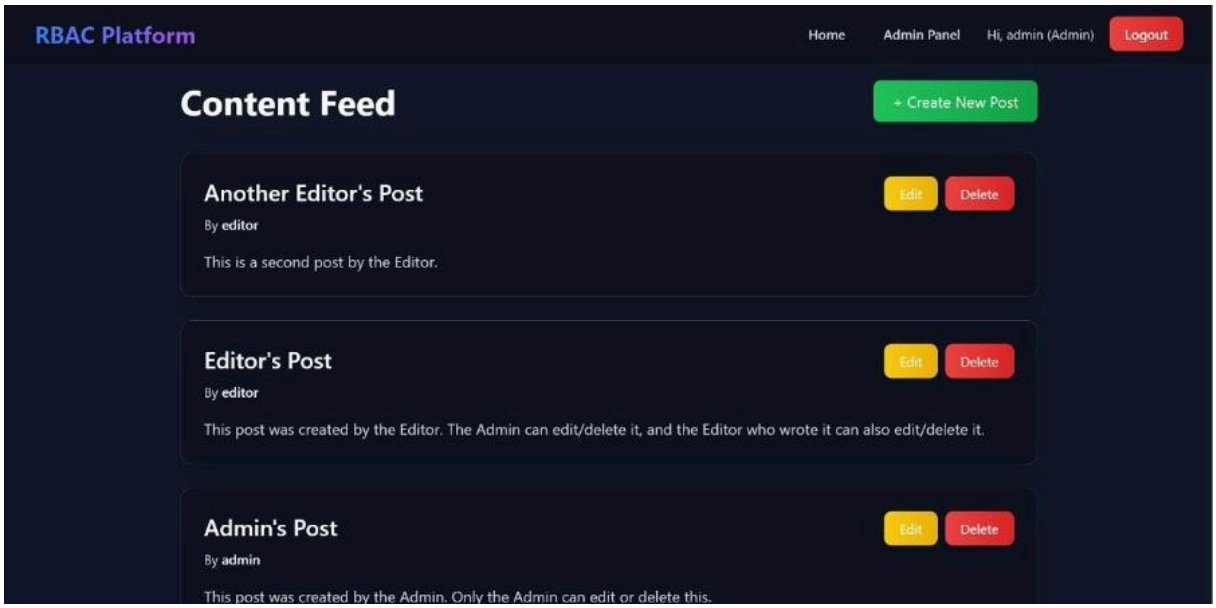


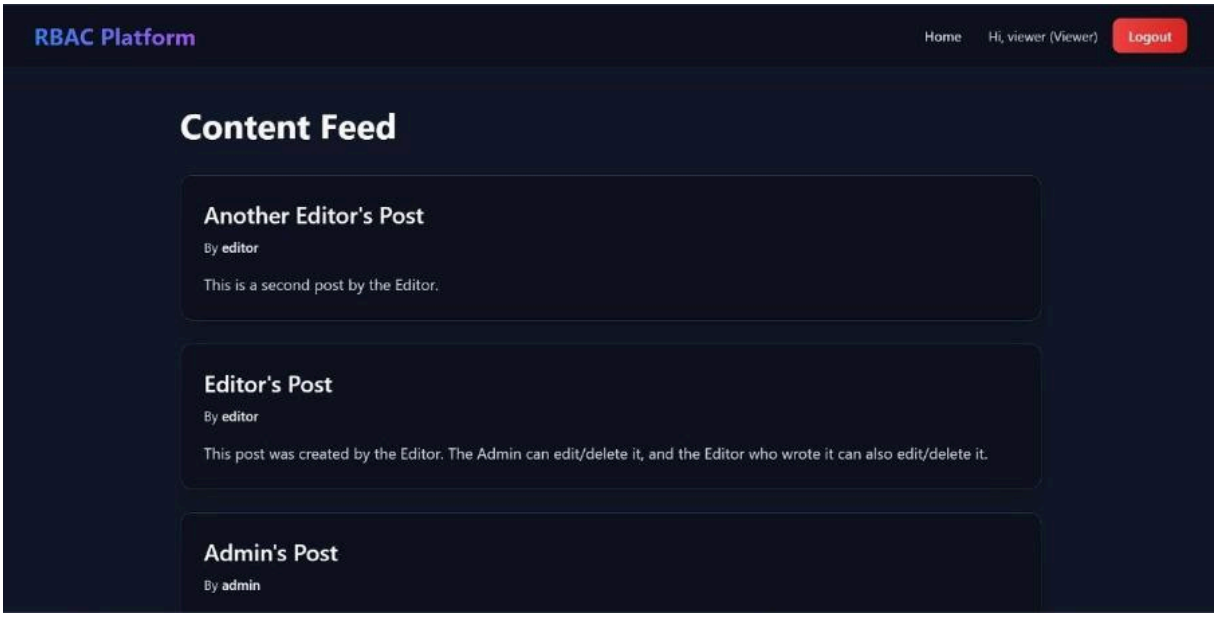


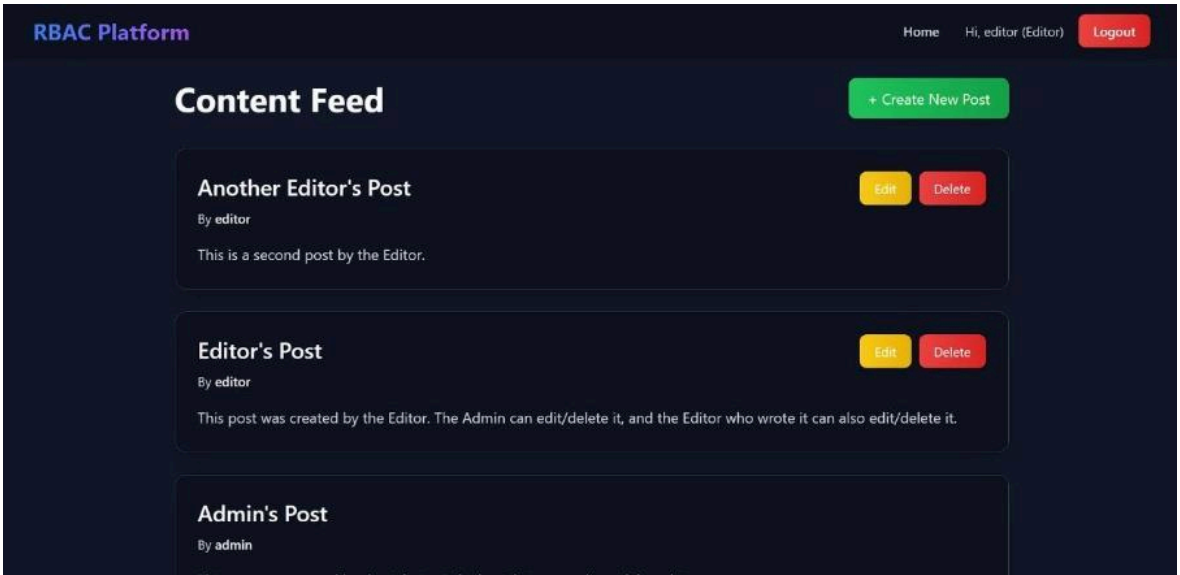

RBAC Platform
Home
Admin Panel
Hi, admin (Admin)
Logout
Admin Dashboard
Manage Content Feed
View, edit, and delete all posts.
Manage Users
View all users and change their roles.
View Audit Log
See a log of important system events.


RBAC Platform
Home
Admin Panel
Hi, admin (Admin)
Logout
User Management
USERNAME
ROLE
ACTIONS
admin
Admin
Admin
editor
Editor
Editor
viewer
Viewer
Viewer
Admin
Editor
Viewer


RBAC Platform
Home
Admin Panel
Hi, admin (Admin)
Logout
Audit Log (Recent)
admin admin logged in.
11/7/2025, 6:58:53 PM
sample sample registered a new account.
11/7/2025, 6:58:44 PM
admin admin logged in.
11/7/2025, 6:57:37 PM


RBAC Platform
Home
Admin Panel
Hi, admin (Admin)
Logout
Content Feed
+ Create New Post
Another Editor's Post
Edit
Delete
By editor
This is a second post by the Editor.
Editor's Post
Edit
Delete
By editor
This post was created by the Editor. The Admin can edit/delete it, and the Editor who wrote it can also edit/delete it.
Admin's Post
Edit
Delete
By admin
This post was created by the Admin. Only the Admin can edit or delete this.


RBAC Platform
Home
Hi, viewer (Viewer)
Logout
Content Feed
Another Editor's Post
By editor
This is a second post by the Editor.
Editor's Post
By editor
This post was created by the Editor. The Admin can edit/delete it, and the Editor who wrote it can also edit/delete it.
Admin's Post
By admin


RBAC Platform
Home
Hi, editor (Editor)
Logout
Content Feed
+ Create New Post
Another Editor's Post
Edit
Delete
By editor
This is a second post by the Editor.
Editor's Post
Edit
Delete
By editor
This post was created by the Editor. The Admin can edit/delete it, and the Editor who wrote it can also edit/delete it.
Admin's Post
By admin

References

These are the **primary online sources** and official documentation consulted during the development and study of the project:

1. MongoDB Documentation – https://www.mongodb.com/docs/
2. Express.js Official Guide – https://expressjs.com/
3. React.js Documentation – https://react.dev/
4. Node.js Official Docs – https://nodejs.org/en/docs
5. JSON Web Token (JWT) Introduction – https://jwt.io/introduction/
6. OWASP Cheat Sheet: Access Control – https://cheatsheetseries.owasp.org/
7. REST API Design Guidelines – Microsoft Developer Network (MSDN)
8. Mongoose ODM Documentation – https://mongoosejs.com/docs/
9. React Router v6 Documentation – https://reactrouter.com/
10. Tailwind CSS Documentation (for frontend styling) – https://tailwindcss.com/docs

Bibliography

These are the **books, research papers, and additional readings** referred for theoretical understanding of RBAC, web security, and MERN architecture:

1. **Sandhu, R., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996).**
   Role-Based Access Control Models. IEEE Computer, 29(2), 38–47.

2. **McDonald, M. (2021).**
   Web Security for Developers. No Starch Press.

3. **Flanagan, D. (2020).**
   JavaScript: The Definitive Guide. O'Reilly Media.

4. **Banks, A., & Porcello, E. (2021).**
   Learning React: Modern Patterns for Developing React Apps. O'Reilly Media.

5. **Subramaniam, V. (2018).**
   Programming Node.js: Building Scalable Network Applications. Pragmatic Bookshelf.

6. **Fowler, M. (2012).**

Patterns of Enterprise Application Architecture. Addison-Wesley Professional.

7. **Fogel, K. (2017).**
   Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly Media.

8. **Singh, N., & Sharma, P. (2020).**
   Practical MongoDB: A Beginner's Guide to NoSQL and Its Implementation. Apress.

9. **Rouse, M. (2022).**
   Understanding Role-Based Access Control (RBAC). TechTarget, Security Definitions.

10. **W3C Consortium (2023).**
    Web Application Security Guidelines. World Wide Web Consortium (W3C).