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

A PROJECT REPORT

Submitted by

Name- Madhav

UID- 23BAI70107

in partial fulfilment for the award of the degree of

Computer Science Engineering with

specialization **IN**

Artificial Intelligence and Machine Learning (AIML)

Chandigarh University

NOVEMBER, 2025

BONAFIDE CERTIFICATE

Certified that this project report **“Role-Based Access Control (RBAC) in a MERN application”** is the bonafide work of **“Madhav”** who carried out the project work under my/our supervision.

Table of Content

PROBLEM STATEMENT

Develop a secure and fine-grained **Role-Based Access Control (RBAC)** system within a **MERN stack** application that differentiates permissions among three user roles — **Admin, Editor, and Viewer**. The backend, built using **Node.js and Express**, should implement **JWT-based authentication** to store and verify user roles, enforcing authorization through middleware and conditional access in MongoDB queries. On the **React frontend**, user permissions must dynamically influence the interface by controlling access to routes, components, and UI actions. The system should also include seeded demo users and showcase ownership validation, where **Editors** can only edit their own data while **Admins** have complete management authority.

Core Functionalities:

- **Role & Permission Mapping** – Create a configuration or collection that defines capabilities for each role across all system operations.
- **JWT Authentication** – Implement short-term access tokens containing user information and roles, with secure handling through cookies or local storage.
- **Access Control Middleware** – Integrate middleware that verifies permissions and restricts unauthorized access by default.
- **Data Filtering by Role** – Apply access control rules at the database query level to ensure users can only interact with permitted data.
- **Frontend Access Management** – Use route guards and conditional rendering to control visibility and availability of restricted UI elements.
- **Admin Console** – Build an interface for administrators to manage users, assign roles, and monitor system activity.
- **Security and Validation** – Implement strict input validation, password hashing, rate limiting, and apply CORS and CSRF protections.
- **Monitoring & Logging** – Maintain structured logs and metrics to track authorization attempts and support troubleshooting.
- **Testing Framework** – Conduct unit, integration, and end-to-end tests to ensure consistent role-based behaviour.
- **Development Setup & Seeding** – Provide initialization scripts to generate sample data and predefined user roles for testing and demonstration.

PROJECT DESCRIPTION

AuthFlow – MERN Role-Based Access Control (RBAC) System is a secure and scalable web application developed using the **MERN stack** — **MongoDB, Express.js, React.js, and Node.js**. It integrates a fine-grained **Role-Based Access Control** mechanism to manage user permissions and access levels across both backend APIs and frontend interfaces.

The application defines three main roles — **Admin, Editor, and Viewer** — each with specific capabilities for performing operations such as creation, modification, viewing, and deletion of data. The backend is powered by **Node.js and Express.js**, utilizing **JWT-based authentication** to verify user identity and embed role claims. Authorization middleware ensures that each API request complies with defined role permissions, enforcing ownership validation and preventing unauthorized access.

On the **frontend**, **React.js** provides a dynamic and responsive interface where role-based route protection and conditional component rendering are implemented. UI elements such as buttons, menus, and forms are automatically shown or hidden according to the user's role. The **MongoDB** database stores user accounts, roles, and permission sets, with query-level filtering and indexing to maintain secure and efficient data operations.

The system also includes an **Admin Panel** for user and role management, allowing administrators to monitor and update user access seamlessly. Security best practices such as input validation, password hashing, CORS and CSRF protection, and rate limiting are implemented to enhance overall system resilience. Logging and monitoring modules capture important authentication and access events to support auditing and analysis.

Overall, **AuthFlow** demonstrates a robust and maintainable approach to authentication, authorization, and access control within a modern full-stack environment, ensuring both **security and usability** in web application design.

HARDWARE / SOFTWARE REQUIREMENTS

The AuthFlow MERN RBAC project is a full-stack web application that requires a standard development environment suitable for executing both client-side and server-side technologies. The following hardware and software configurations are recommended for smooth operation and optimal performance of the system.

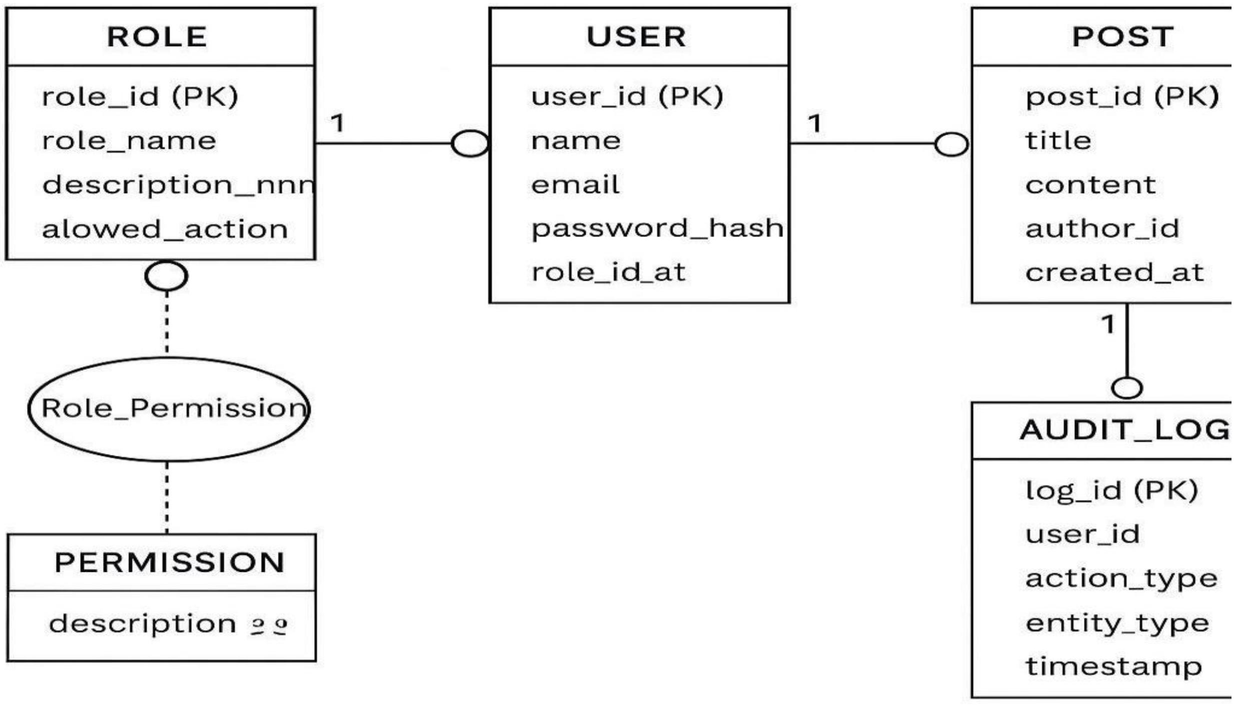
Hardware Requirements

Component	Minimum Specification	Recommended Specification
Processor (CPU)	Intel Core i3 (2.0 GHz or higher)	Intel Core i5/i7 (3.0 GHz or higher)
RAM	4 GB	8 GB or more
Hard Disk Storage	256 GB HDD	512 GB SSD
Display	1366 × 768 resolution	1920 × 1080 Full HD
Keyboard & Mouse	Standard Input Devices	Standard Input Devices
Network	10/100 Mbps Internet Connection	High-Speed Broadband Connection
Power Backup	Optional	UPS recommended for server environment

Software Requirements

Category	Specification
Operating System	Windows 10 / Windows 11 / Ubuntu 22.04 LTS
Frontend Framework	React.js (Version 18 or above)
Backend Framework	Node.js with Express.js (Node v18 or above)
Database	MongoDB (Version 6.0 or above)
Authentication Library	JSON Web Token (JWT)
State Management	Redux Toolkit
Development IDE	Visual Studio Code
Containerization	Docker & Docker Compose
API Testing Tool	Postman
Version Control System	Git & GitHub
Package Manager	npm (Node Package Manager)
Browser	Google Chrome / Mozilla Firefox
Documentation Tool	Microsoft Word / Google Docs

ER DIAGRAM



Relationships

1. Role → User → One-to-Many

- Each **Role** can be assigned to multiple **Users**.
- This defines access levels across users, such as Admin, Editor, and Viewer.
- **Example:** The **Admin** role can be associated with many users having administrative privileges.

2. User → Post → One-to-Many

- A single **User** can create multiple **Posts**, but each **Post** belongs to only one **User**.
- This enforces ownership — only the author can edit or delete their posts.
- **Example:** An **Editor** can create multiple posts, each linked to their user account.

3. User → ActivityLog → One-to-Many

- A **User** can generate multiple **Activity Logs**, recording their actions in the system.
- This helps maintain transparency and auditability within the application.
- **Example:** Login, Post Creation, or Role Change actions are recorded in the **ActivityLog** table.

4. Role → Permission → One-to-Many (Conceptual)

- Each **Role** contains a set of **Permissions** that define what operations can be performed.
- Permissions act as fine-grained control elements, enabling or restricting actions.
- **Example:**
  - **Admin** → create, read, update, delete
  - **Editor** → create, read, update (own)
  - **Viewer** → read only

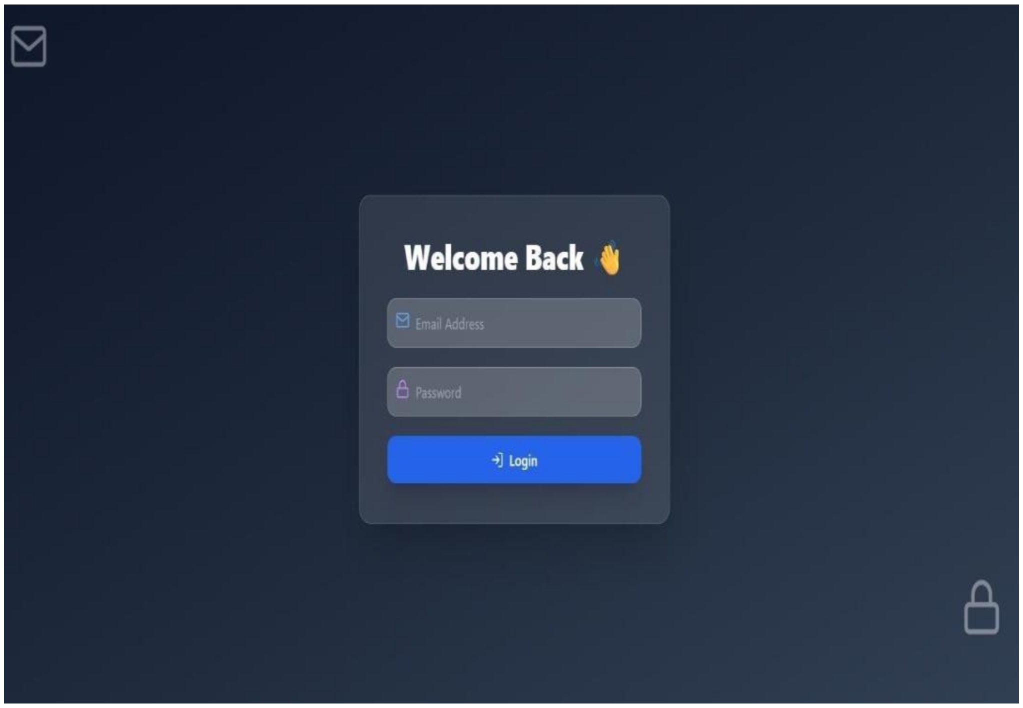
FRONT-END SCREENS

The front-end interface of the AuthFlow MERN RBAC system has been developed using React.js to provide an intuitive and responsive user experience. The interface dynamically adapts based on the logged-in user's role — *Admin*, *Editor*, or *Viewer*. This role-based UI behavior ensures that each user sees only the functionalities they are permitted to access.

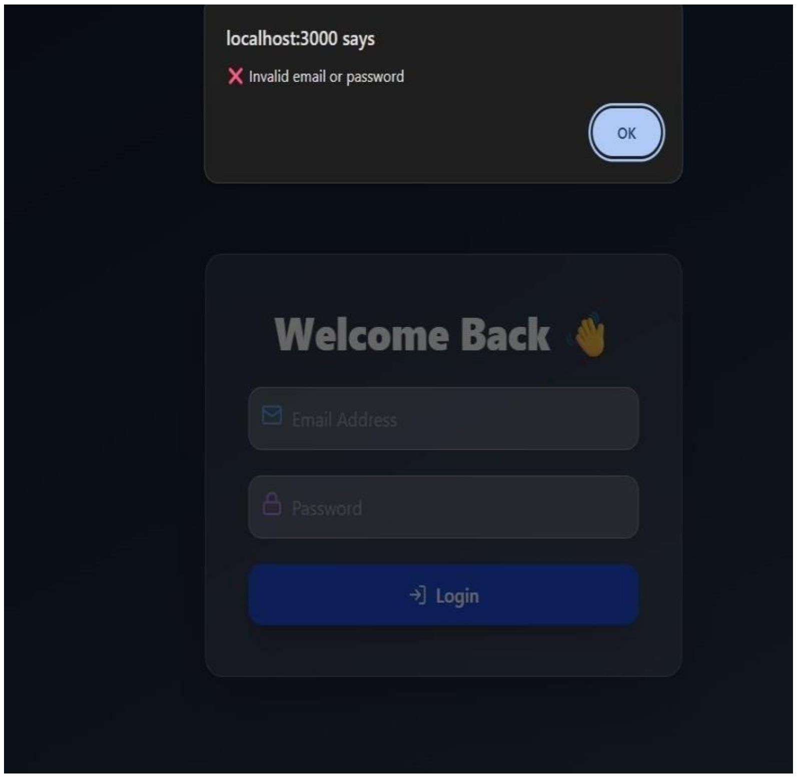
The React frontend integrates React Router DOM for navigation, Redux Toolkit for state management, and Axios for API communication with the backend.

All UI components are designed with reusability and security in mind, ensuring restricted actions are visually disabled or hidden from unauthorized users.

1. Login_Page



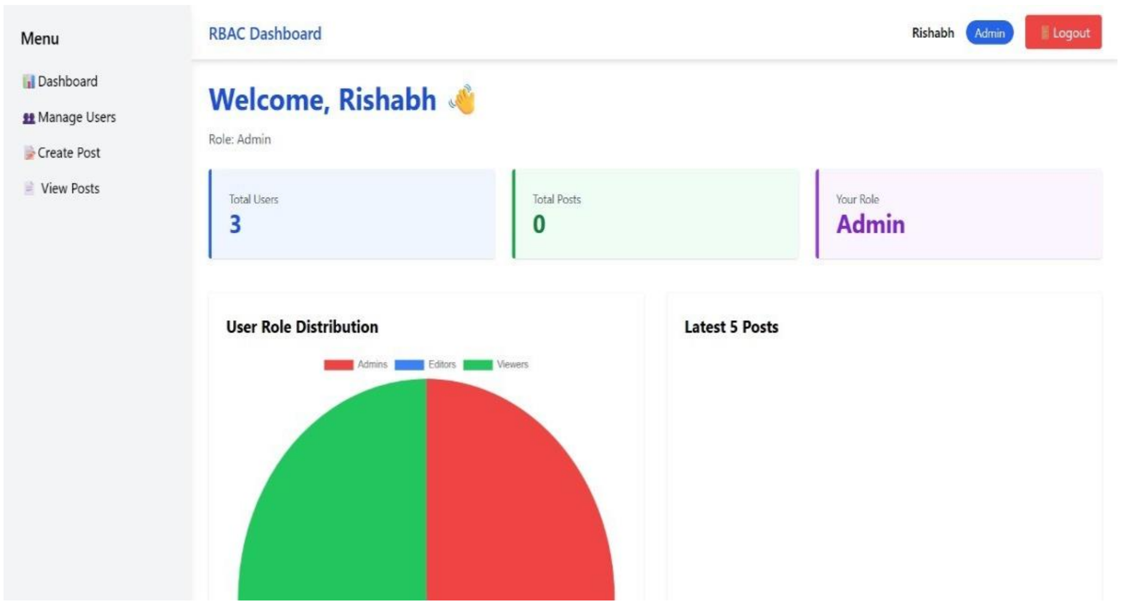
2. Access Denied Page



localhost:3000 says

❌ Invalid email or password

3. Admin Dashboard



Welcome, Rishabh 👋

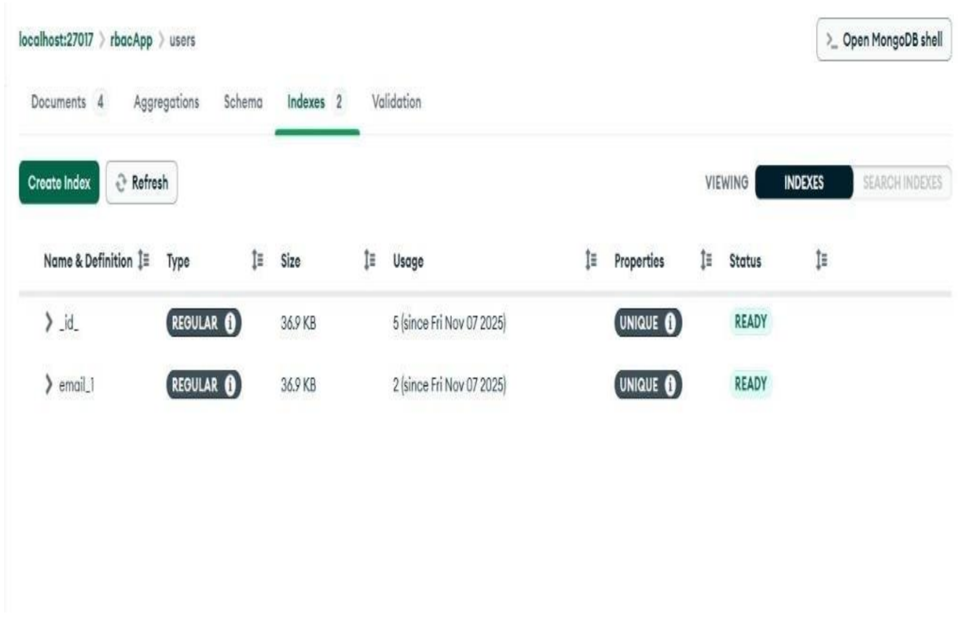
Role: Admin

Total Users 3

Total Posts 0

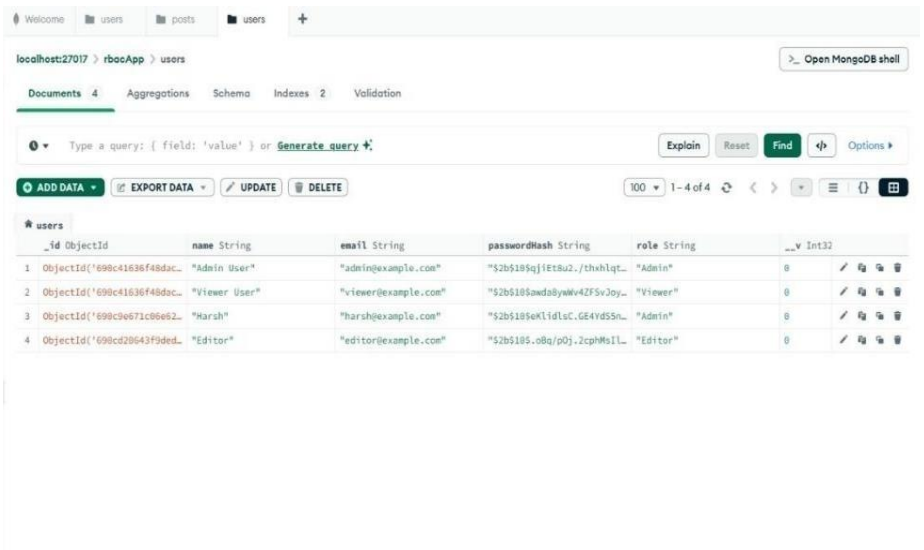
Your Role Admin

USER DATABASE



localhost:27017 › rbacApp › users

Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	36.9 KB	5 (since Fri Nov 07 2025)	UNIQUE	READY
email_1	REGULAR	36.9 KB	2 (since Fri Nov 07 2025)	UNIQUE	READY



localhost:27017 › rbacApp › users

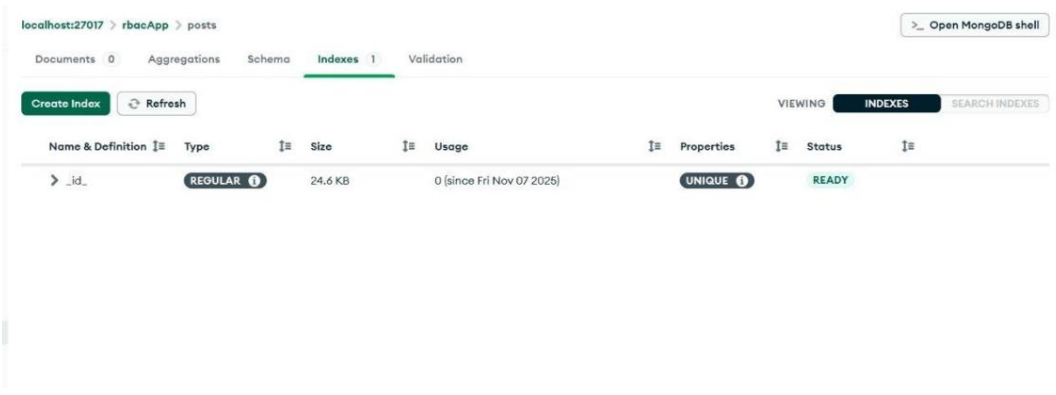
Type a query: { field: 'value' } or Generate query

1 – 4 of 4

users

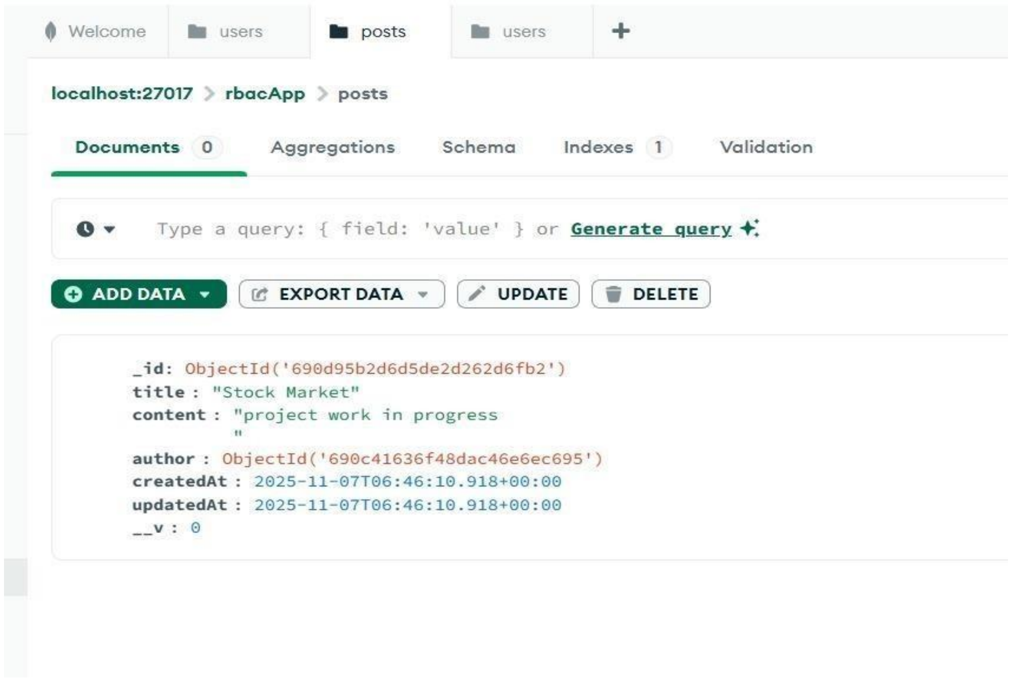
	_id ObjectId	name String	email String	passwordHash String	role String	__v Int32
1	ObjectId('690c41636f48dac…	"Admin User"	"admin@example.com"	"$2b$10$qjiEt8u2./thxhlqt…	"Admin"	0
2	ObjectId('690c41636f48dac…	"Viewer User"	"viewer@example.com"	"$2b$10$awda8ywWv4ZFSvJoy…	"Viewer"	0
3	ObjectId('690c9e671c06e62…	"Harsh"	"harsh@example.com"	"$2b$10$eKlidlsC.GE4YdS5n…	"Admin"	0
4	ObjectId('690cd20643f9ded…	"Editor"	"editor@example.com"	"$2b$10$.o8q/pOj.2cphMsIl…	"Editor"	0

POST DATABASE



localhost:27017 › rbacApp › posts

Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	24.6 KB	0 (since Fri Nov 07 2025)	UNIQUE	READY



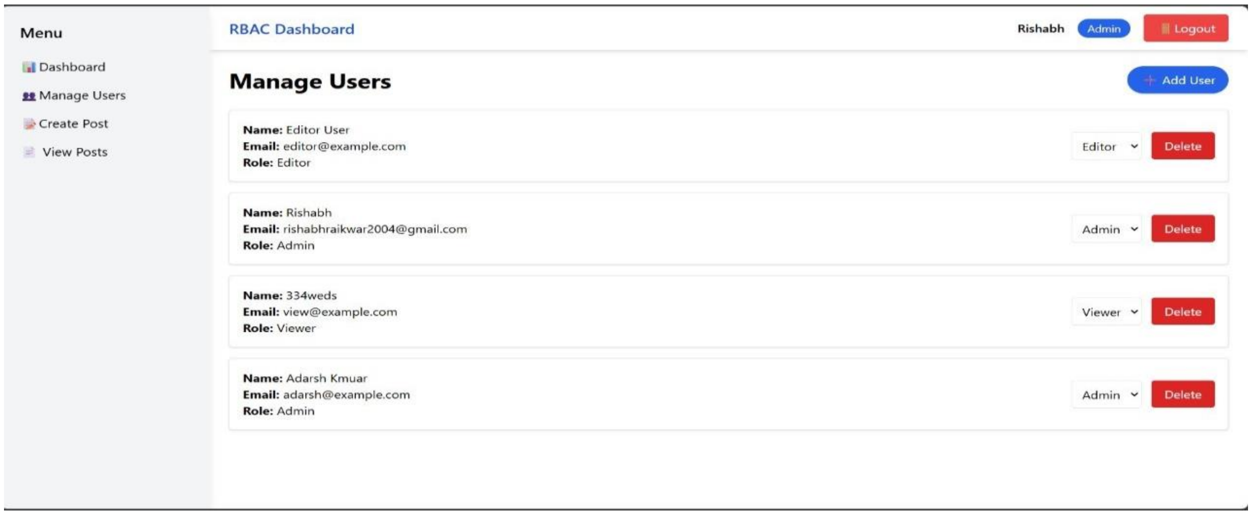
```
_id: ObjectId('690d95b2d6d5de2d262d6fb2')
title : "Stock Market"
content : "project work in progress
          "
author : ObjectId('690c41636f48dac46e6ec695')
createdAt : 2025-11-07T06:46:10.918+00:00
updatedAt : 2025-11-07T06:46:10.918+00:00
__v : 0
```

OUTPUT SCREENS

The **output screens** illustrate the successful execution of the system functionalities for different user roles — *Admin*, *Editor*, and *Viewer*. They validate the proper implementation of **authentication**, **authorization**, and **role-based permissions** across the MERN application. Each output screen demonstrates how the interface and API responses change according to the user's assigned privileges.

1. Admin Dashboard Output

2. Role Updation



Manage Users

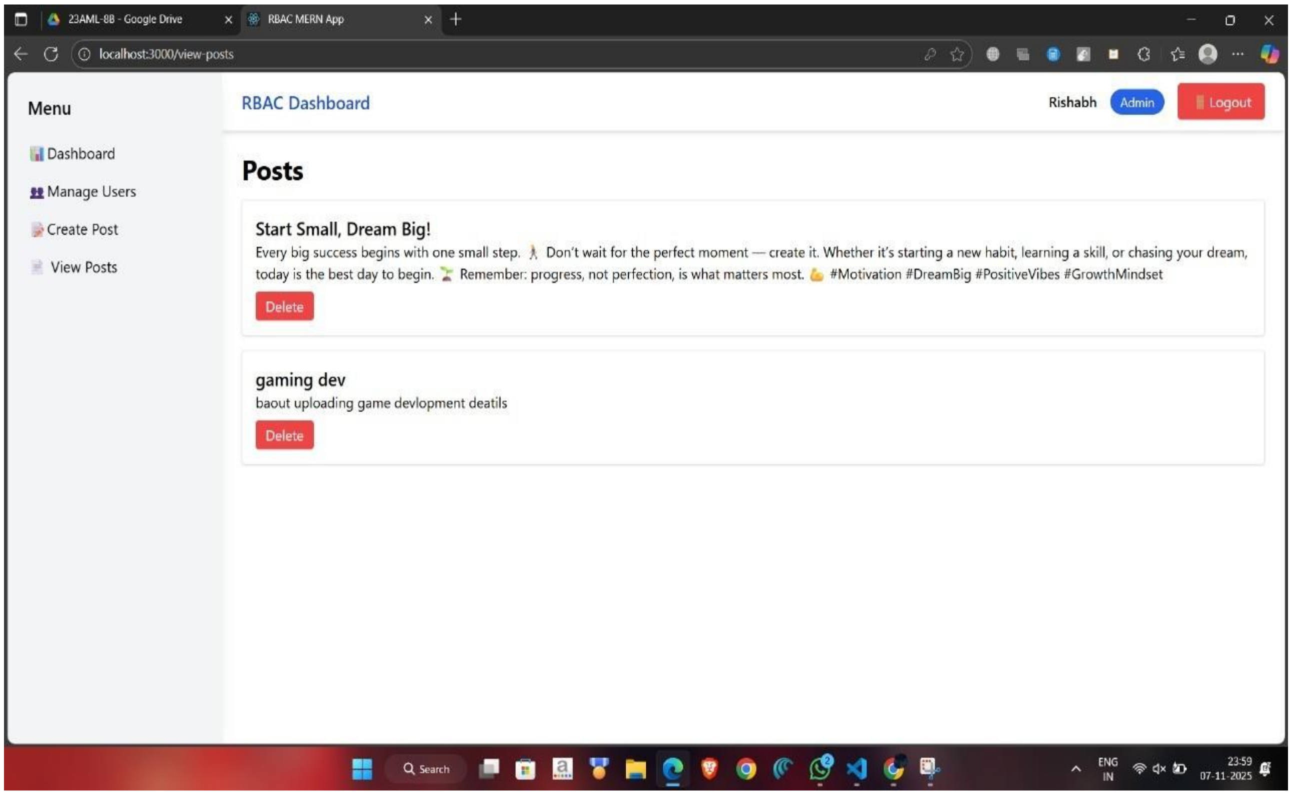
Name: Editor User
Email: editor@example.com
Role: Editor
Editor

Name: Rishabh
Email: rishabhraikwar2004@gmail.com
Role: Admin
Admin

Name: 334weds
Email: view@example.com
Role: Viewer
Viewer

Name: Adarsh Kmuar
Email: adarsh@example.com
Role: Admin
Admin

3. Create Post Successfully



Posts

Start Small, Dream Big!

Every big success begins with one small step. 🚶 Don't wait for the perfect moment — create it. Whether it's starting a new habit, learning a skill, or chasing your dream, today is the best day to begin. 🌱 Remember: progress, not perfection, is what matters most. 💪 #Motivation #DreamBig #PositiveVibes #GrowthMindset

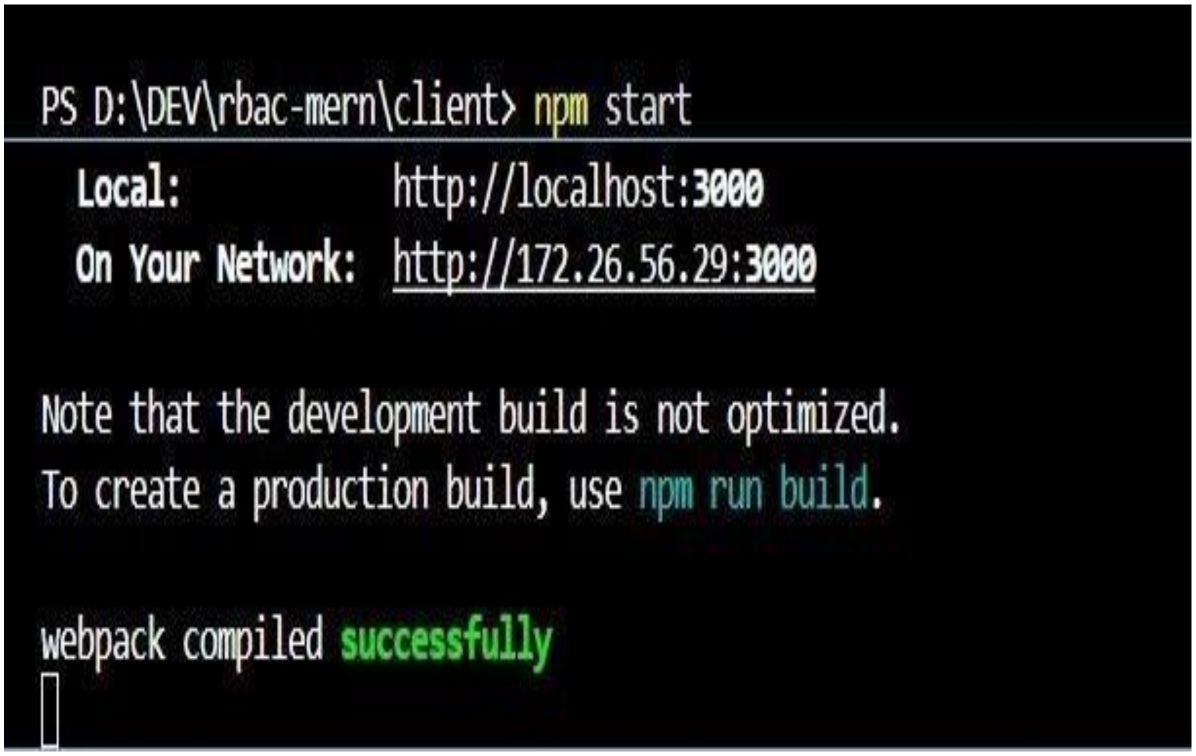
Delete

gaming dev

baout uploading game devlopment deatils

Delete

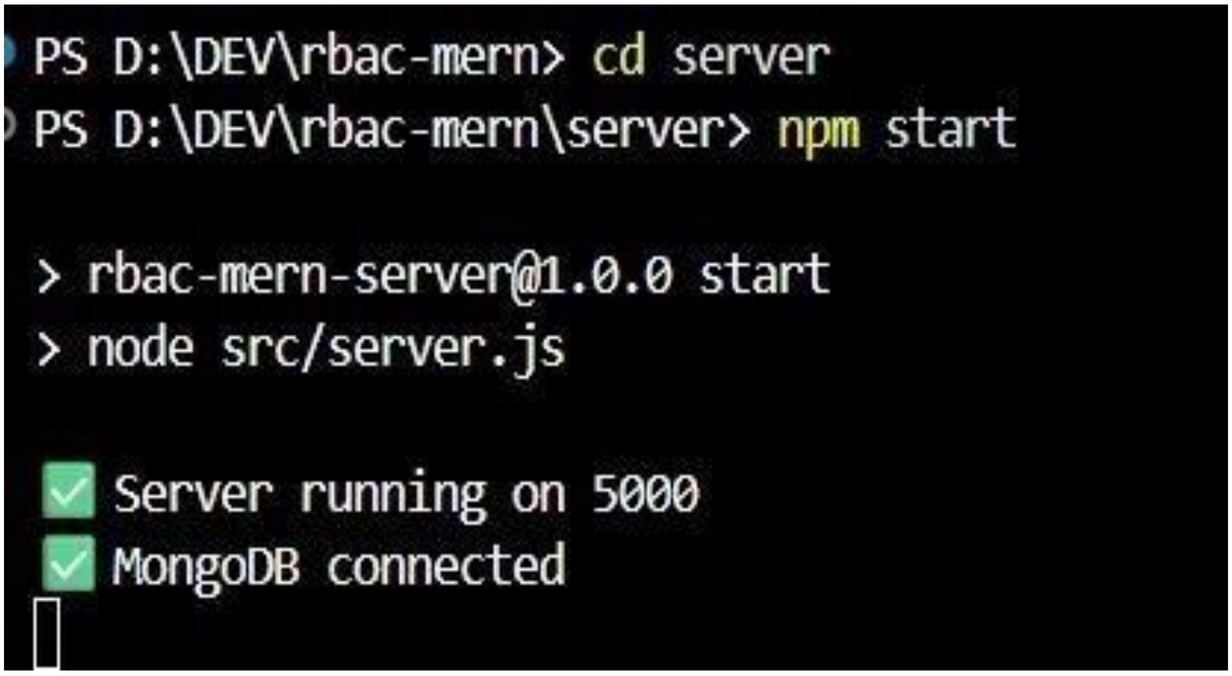
4. Backend Output Screen



```
PS D:\DEV\rbac-mern\client> npm start
  Local:            http://localhost:3000
  On Your Network:  http://172.26.56.29:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

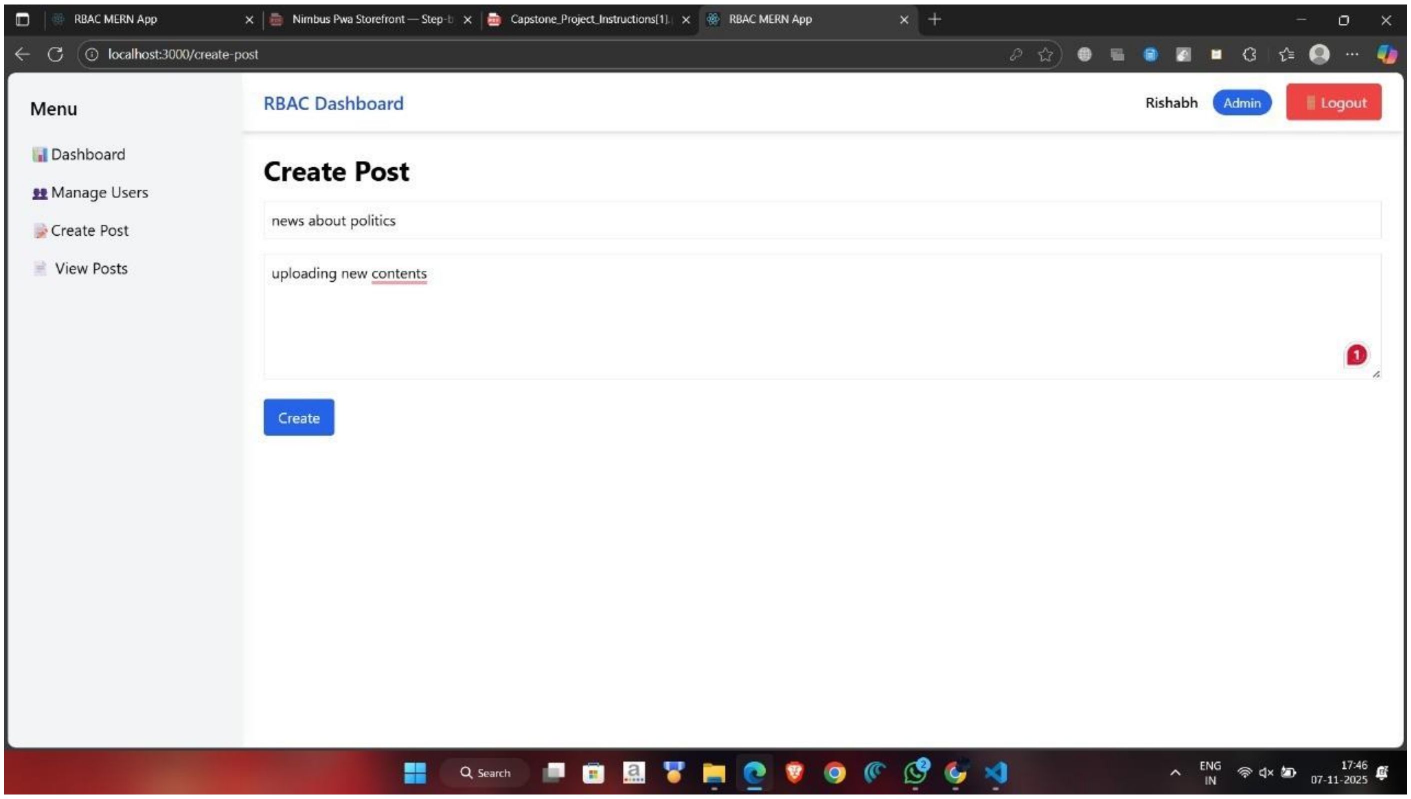


```
PS D:\DEV\rbac-mern> cd server
PS D:\DEV\rbac-mern\server> npm start

> rbac-mern-server@1.0.0 start
> node src/server.js

✅ Server running on 5000
✅ MongoDB connected
```

5. Posts Display



Create Post

news about politics

uploading new contents

LIMITATION & FUTURE SCOPE

The **AuthFlow – MERN Role-Based Access Control (RBAC)** project successfully demonstrates the implementation of fine-grained access management using modern web technologies. However, as with any developing software system, there are certain constraints and opportunities for enhancement identified during development and testing.

Limitations

- **Limited Role Variety**
  The current implementation includes only three predefined roles — *Admin*, *Editor*, and *Viewer*. While effective for demonstration purposes, it limits flexibility for organizations that require customized roles or more granular permission levels.
- **Manual Role and Permission Management**
  Role and permission configurations are currently handled manually through backend settings or the Admin dashboard. The system lacks a self-service interface to dynamically create, update, or delete roles and permissions in real time.
- **No Multi-Tenant Architecture**
  The platform functions as a single-tenant application, where all users share one database instance. Enabling multi-tenancy would allow multiple organizations to use the same system with isolated data and configurations.
- **Absence of Advanced Authentication Mechanisms**
  While JWT-based authentication ensures secure access, additional layers such as Two-Factor Authentication (2FA), OTP-based login, or biometric verification could further strengthen security.
- **No Real-Time Notification System**
  The system currently lacks in-app or email-based notifications to alert users about key activities such as role changes, access denials, or post updates.
- **Basic Logging and Monitoring Capabilities**
  The existing logging focuses on basic authentication and access control events. A more advanced observability setup with dashboards using tools like Grafana, ELK Stack, or Prometheus could enhance monitoring and debugging.

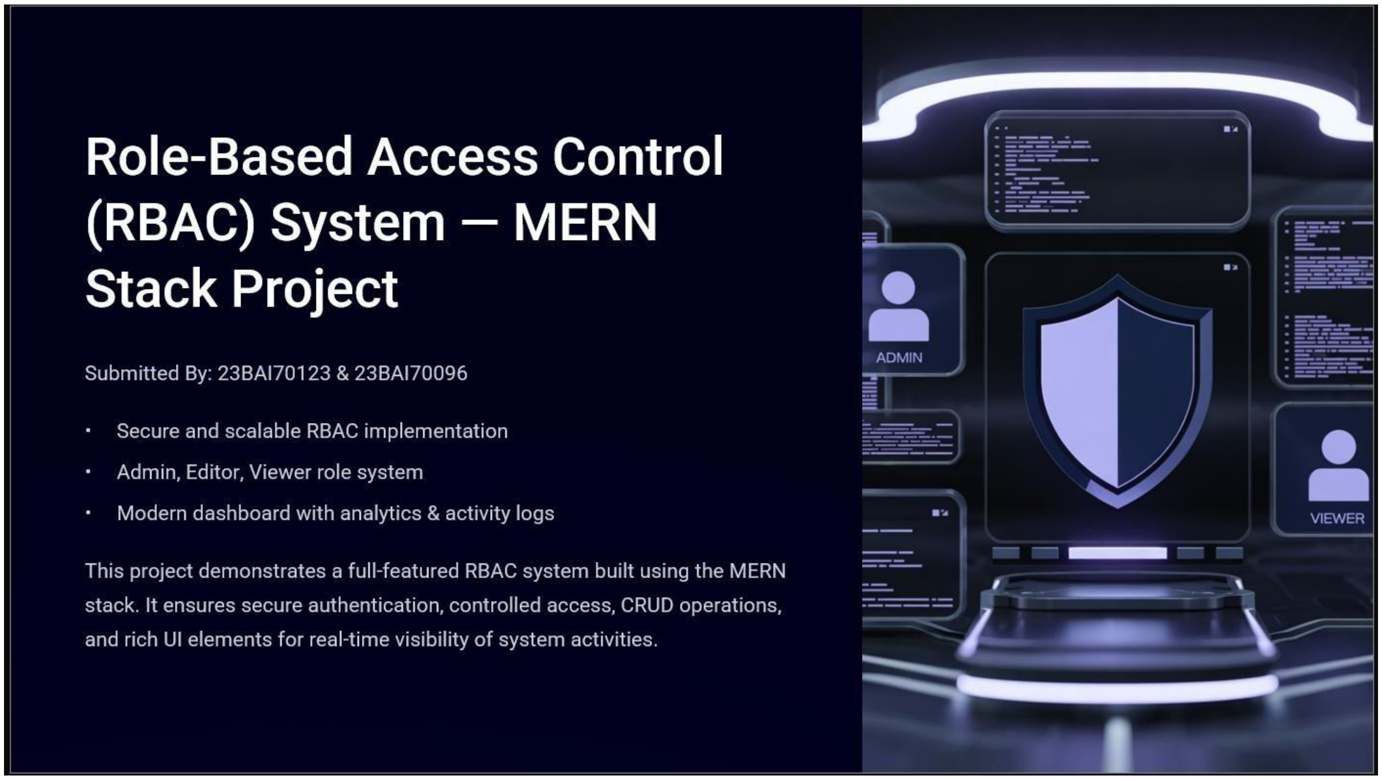
Future Scope

1. **Responsive and Modern UI/UX Design**
   Upgrade the frontend using modern frameworks such as **Tailwind CSS**, **Material UI**, or **Bootstrap 5**. This will ensure a visually appealing, responsive, and accessible interface across all device types and screen sizes.
2. **Mobile Application Development**
   Build a cross-platform mobile app using **React Native** or **Flutter** to provide seamless access for users on Android and iOS devices, ensuring convenience and mobility.
3. **Cloud Deployment and CI/CD Integration**
   Migrate the system to cloud platforms such as **AWS**, **Azure**, or **Google Cloud**, and implement **CI/CD pipelines** for automated builds, testing, and deployment. This will streamline maintenance and support continuous improvement.
4. **AI-Powered Access Insights**
   Incorporate machine learning models to detect abnormal user behavior, flag suspicious login patterns, and generate predictive insights for proactive security management.
5. **Role-Based Dashboard Personalization**
   Develop customized dashboards for each user role (Admin, Editor, Viewer) to display relevant data, actions, and analytics, improving efficiency and user experience.
6. **Multi-Tenancy Support**
   Expand the architecture to support **multi-organization environments**, where each tenant has isolated data, configurations, and role hierarchies — ideal for enterprise-level scalability.
7. **Integration with Third-Party Authentication Providers**
   Enable **Single Sign-On (SSO)** using platforms like **Google, Microsoft, or Auth0**, allowing users to log in securely through their existing credentials.
8. **Comprehensive Audit Trail Interface**
   Create a dedicated **Audit Trail Dashboard** that allows administrators to view, filter, and export system activities — aiding transparency and compliance reporting.
9. **Data Encryption and Compliance Enhancements**
   Strengthen data protection with **end-to-end encryption** for sensitive information and ensure compliance with global data privacy standards such as **GDPR** or **HIPAA**.

GITHUB URL

https://github.com/rishabhcseaiml/rbac

PRESENTATION SCREENSHOTS



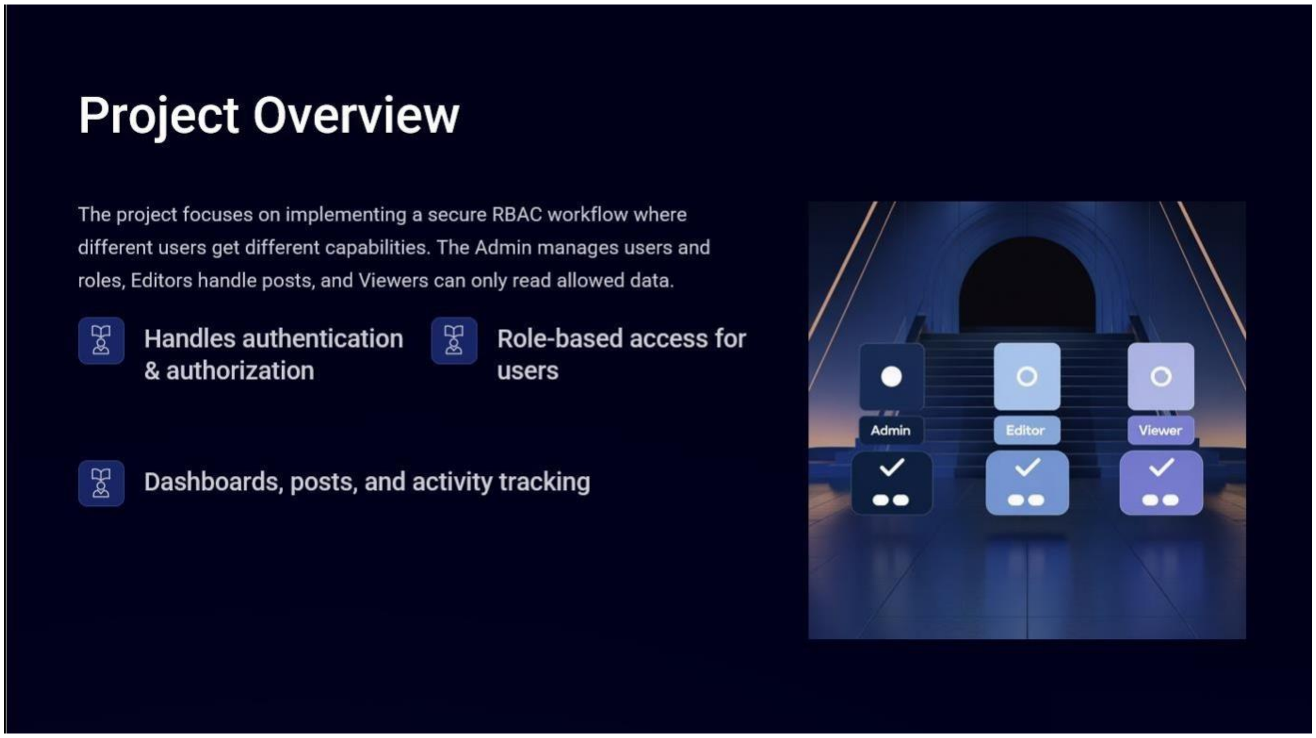
Role-Based Access Control (RBAC) System — MERN Stack Project

Submitted By: 23BAI70123 & 23BAI70096

- Secure and scalable RBAC implementation
- Admin, Editor, Viewer role system
- Modern dashboard with analytics & activity logs

This project demonstrates a full-featured RBAC system built using the MERN stack. It ensures secure authentication, controlled access, CRUD operations, and rich UI elements for real-time visibility of system activities.

ADMIN
VIEWER

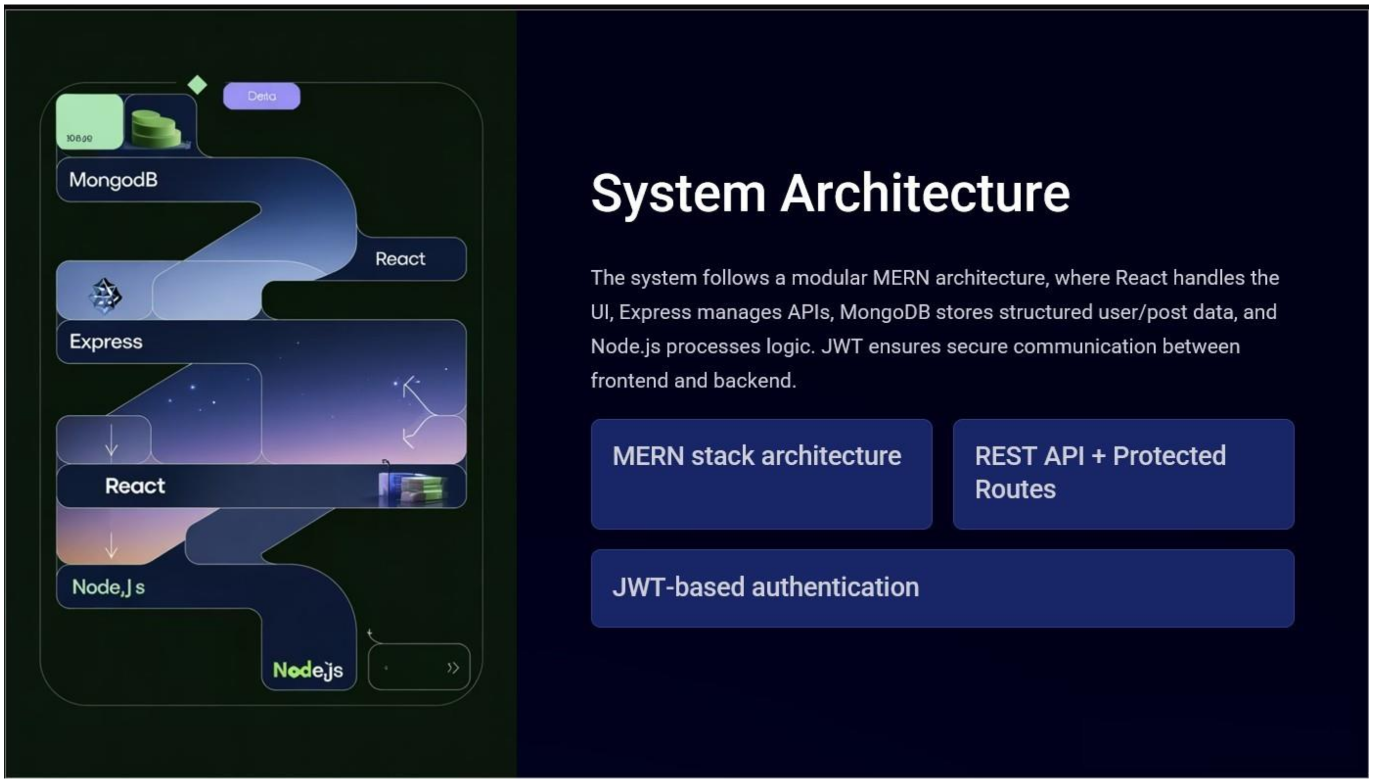


Project Overview

The project focuses on implementing a secure RBAC workflow where different users get different capabilities. The Admin manages users and roles, Editors handle posts, and Viewers can only read allowed data.

- Handles authentication & authorization
- Role-based access for users
- Dashboards, posts, and activity tracking

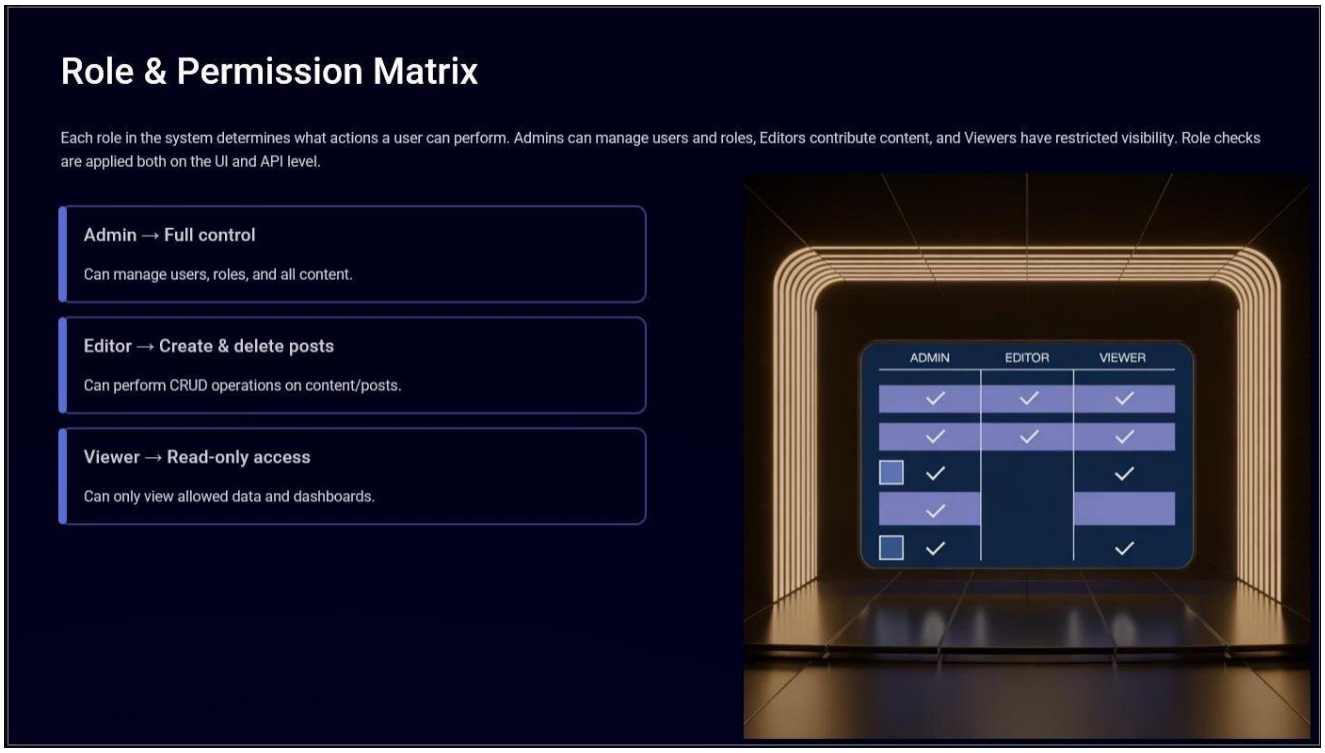
Admin
Editor
Viewer



System Architecture

The system follows a modular MERN architecture, where React handles the UI, Express manages APIs, MongoDB stores structured user/post data, and Node.js processes logic. JWT ensures secure communication between frontend and backend.

- MERN stack architecture
- REST API + Protected Routes
- JWT-based authentication



Role & Permission Matrix

Each role in the system determines what actions a user can perform. Admins can manage users and roles, Editors contribute content, and Viewers have restricted visibility. Role checks are applied both on the UI and API level.

Admin → Full control

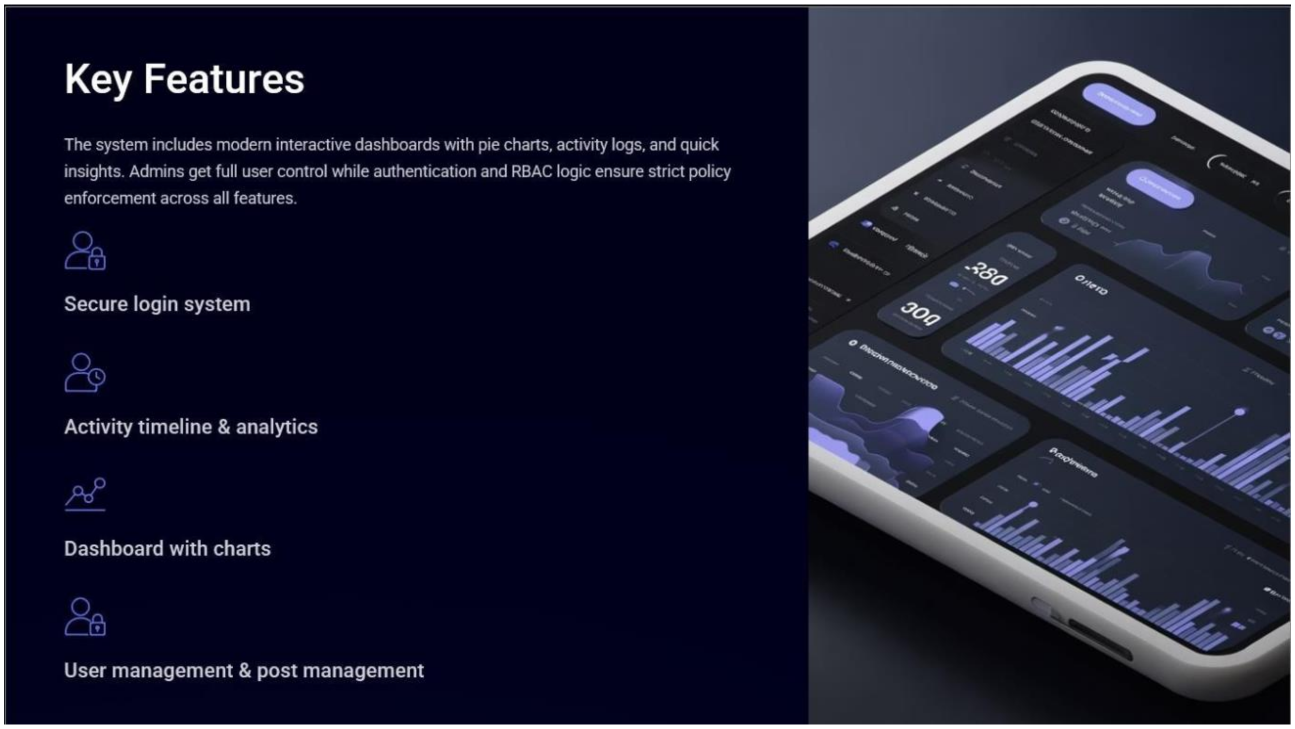
Can manage users, roles, and all content.

Editor → Create & delete posts

Can perform CRUD operations on content/posts.

Viewer → Read-only access

Can only view allowed data and dashboards.



Key Features

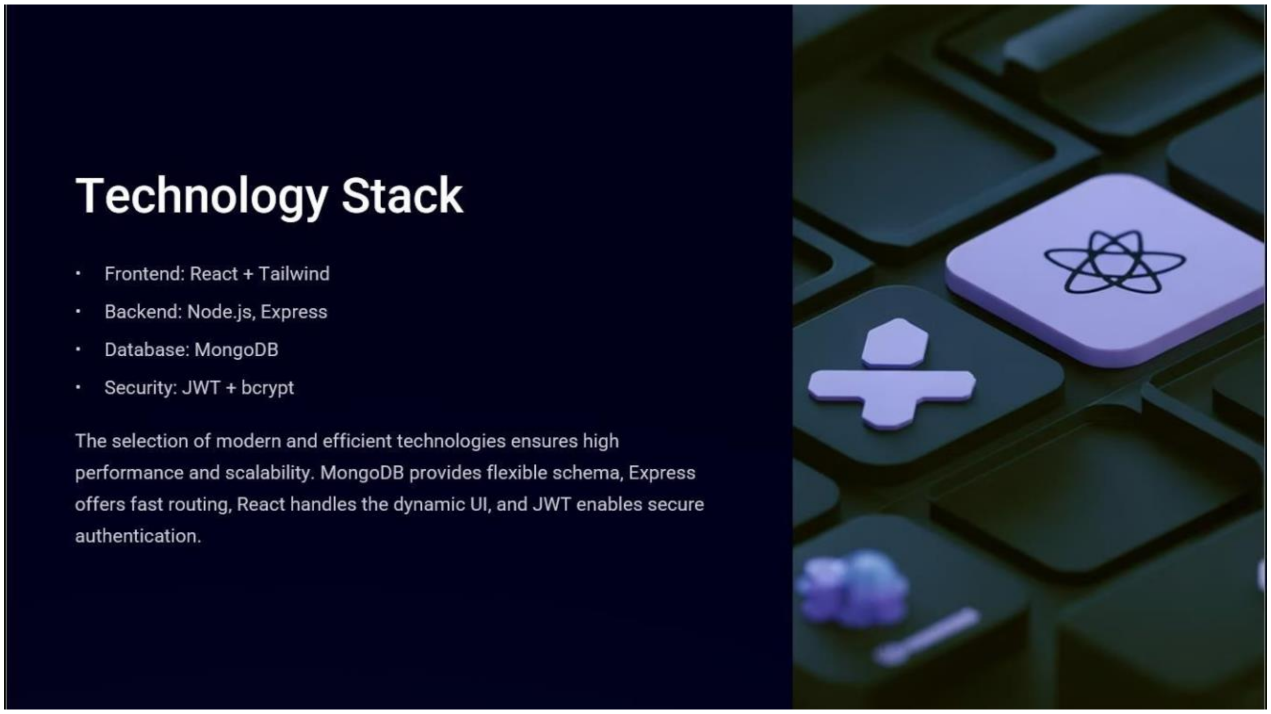
The system includes modern interactive dashboards with pie charts, activity logs, and quick insights. Admins get full user control while authentication and RBAC logic ensure strict policy enforcement across all features.

Secure login system

Activity timeline & analytics

Dashboard with charts

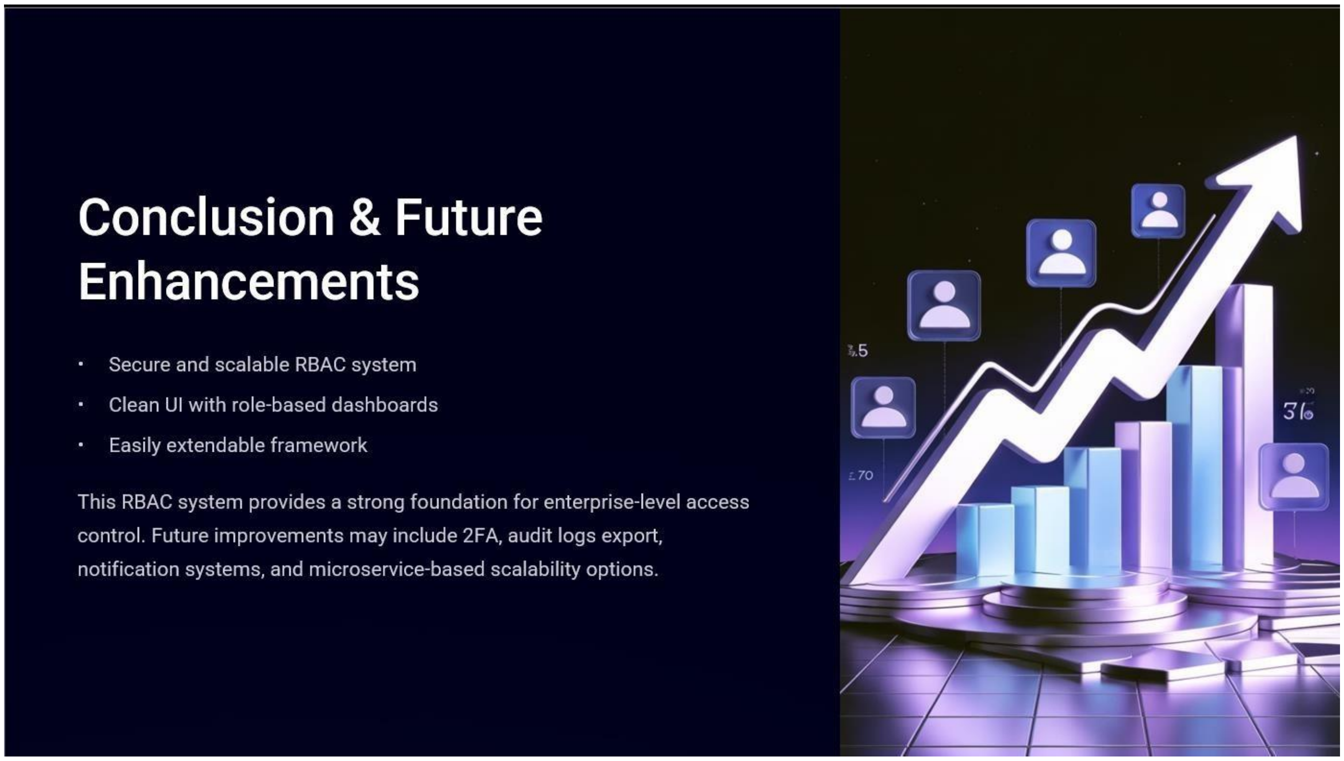
User management & post management



Technology Stack

- Frontend: React + Tailwind
- Backend: Node.js, Express
- Database: MongoDB
- Security: JWT + bcrypt

The selection of modern and efficient technologies ensures high performance and scalability. MongoDB provides flexible schema, Express offers fast routing, React handles the dynamic UI, and JWT enables secure authentication.



Conclusion & Future Enhancements

- Secure and scalable RBAC system
- Clean UI with role-based dashboards
- Easily extendable framework

This RBAC system provides a strong foundation for enterprise-level access control. Future improvements may include 2FA, audit logs export, notification systems, and microservice-based scalability options.

PROJECT CODE

FRONTEND CODE

1. Dashboard.jsx

```
import React, { useEffect, useState, useMemo } from "react";
import axios from "axios";
import Layout from "../components/Layout";
import { Pie } from "react-chartjs-2";
import {
  Chart as ChartJS,
  ArcElement,
  Tooltip,
  Legend,
} from "chart.js";

ChartJS.register(ArcElement, Tooltip, Legend);

export default function Dashboard({ user, onLogout })
  { const [roleCounts, setRoleCounts] = useState({
  Admin: 0,
    Editor: 0,
    Viewer: 0,
  });

  const [totalUsers, setTotalUsers] = useState(0);
  const [totalPosts, setTotalPosts] = useState(0);

  const [recentPosts, setRecentPosts] = useState([]);
  const [timeline, setTimeline] = useState([]);
  const [canShowRoleChart, setCanShowRoleChart] = useState(true);

  const timeAgo = (iso) => {
    const d = new Date(iso);
    const diff = Math.floor((Date.now() - d.getTime()) / 1000);

    const units = [
      ["y", 365 * 24 * 3600],
      ["mo", 30 * 24 * 3600],
      ["d", 24 * 3600],
      ["h", 3600],
      ["m", 60],
      ["s", 1],
    ];

    for (const [label, secs] of units)
      { const val = Math.floor(diff / secs);
      if (val > 0) return `${val}${label} ago`;
    }

    return "just now";
  };

  const loadData = async () => {
    const token = localStorage.getItem("token");
    const headers = { Authorization: "Bearer " + token };

    try {
      const postsRes = await axios.get("/posts", { headers });
      const posts = postsRes.data || [];

      posts.sort((a, b) => new Date(b.createdAt) - new Date(a.createdAt));

      setTotalPosts(posts.length);
      setRecentPosts(posts.slice(0, 5));
      setTimeline(posts.slice(0, 10)); // 🔥 timeline = recent posts only
    try {
      const usersRes = await axios.get("/users", { headers });
      const users = usersRes.data || [];

      setTotalUsers(users.length);
      setRoleCounts({
        Admin: users.filter((u) => u.role === "Admin").length,
        Editor: users.filter((u) => u.role === "Editor").length,
        Viewer: users.filter((u) => u.role === "Viewer").length,
      });

      setCanShowRoleChart(true);
    } catch {
      setCanShowRoleChart(false);
    }
  } catch (err) {
    console.log("Dashboard load error", err);
  }
};

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

const roleChartData = useMemo(
  () => ({
    labels: ["Admins", "Editors", "Viewers"],
    datasets: [
      {
        data: [roleCounts.Admin, roleCounts.Editor, roleCounts.Viewer],
        backgroundColor: ["#ef4444", "#3b82f6", "#22c55e"],
        borderWidth: 0,
      },
    ],
  }),
  [roleCounts]
);

return (
  <Layout user={user} onLogout={onLogout}>
    <h1 className="text-4xl font-bold text-blue-700 mb-
      4"> Welcome, {user.name} 👋
    </h1>
    <p className="text-gray-600 mb-6">Role: {user.role}</p>

    {/* Cards */}
    <div className="grid grid-cols-1 md:grid-cols-3 gap-6 mb-10">
      <div className="p-6 bg-blue-50 border-l-4 border-blue-600 rounded shadow">
        <h2 className="text-sm text-gray-600">Total Users</h2>
        <p className="text-3xl font-bold text-blue-700">
          {canShowRoleChart ? totalUsers : "—"}
        </p>
      </div>

      <div className="p-6 bg-green-50 border-l-4 border-green-600 rounded shadow">
        <h2 className="text-sm text-gray-600">Total Posts</h2>
        <p className="text-3xl font-bold text-green-700">{totalPosts}</p>
      </div>

      <div className="p-6 bg-purple-50 border-l-4 border-purple-600 rounded shadow">
        <h2 className="text-sm text-gray-600">Your Role</h2>
        <p className="text-3xl font-bold text-purple-700">{user.role}</p>
      </div>
    </div>

    {/* Chart + Recent Posts */}
    <div className="grid grid-cols-1 md:grid-cols-2 gap-8 mb-10">
      <div className="bg-white p-6 rounded shadow">
        <h2 className="text-xl font-bold mb-4">User Role Distribution</h2>

        {canShowRoleChart ? (
          <Pie data={roleChartData} />
        ) : (
          <p className="text-gray-500">Admin only section</p>
        )}
      </div>

      <div className="bg-white p-6 rounded shadow">
        <h2 className="text-xl font-bold mb-4">Latest 5 Posts</h2>
```

```
          {recentPosts.map((p) => (
            <div
              key={p._id}
              className="border rounded p-4 mb-3 hover:shadow duration-200"
            >
              <p className="text-xs text-gray-500">{timeAgo(p.createdAt)}</p>
              <h3 className="font-semibold">{p.title}</h3>
              <p className="text-sm text-gray-600 mt-1">
                {p.content.length > 120
                  ? p.content.substring(0, 120) + "..."
                  : p.content}
              </p>
            </div>
          ))}
        </div>
      </div>


    </Layout>
  );
}
```

2. App.jsx

```
import React, { useState } from "react";
import { Routes, Route } from "react-router-dom";

import LoginPage from "./pages/LoginPage";
import Dashboard from "./pages/Dashboard";
import CreatePost from "./pages/CreatePost";
import ViewPosts from "./pages/ViewPosts";
import ManageUsers from "./pages/ManageUsers";

export default function App() {
  const [user, setUser] = useState(null);

  const handleLogin = (data) =>
    { localStorage.setItem("token",
    data.token); setUser(data);
  };

  const handleLogout = () => {
    localStorage.removeItem("token");
    setUser(null);
  };

  // ✅ If user not logged in → show only login page
  if (!user) {
    return <LoginPage onLogin={handleLogin} />;
  }

  return (
    <Routes>
      <Route
        path="/dashboard"
        element={<Dashboard user={user} onLogout={handleLogout} />}
      />

      <Route
        path="/manage-users"
        element={<ManageUsers user={user} onLogout={handleLogout} />}
      />

      <Route
        path="/create-post"
        element={<CreatePost user={user} onLogout={handleLogout} />}
      />

      <Route
        path="/view-posts"
        element={<ViewPosts user={user} onLogout={handleLogout} />}
      />
      {/* ✅ default fallback — open dashboard */}
      <Route
        path="*"
        element={<Dashboard user={user} onLogout={handleLogout} />}
      />
    </Routes>
  );
}
```

BACKEND CODE

1.Seed.js

```
require("dotenv").config();
const mongoose = require("mongoose");
const User = require("../models/User");
console.log("Loaded MONGO_URI =", process.env.MONGO_URI);

(async () =>
  { try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("✅ MongoDB connected");

    await User.deleteMany({});

    const users = [
      { name: "Admin User", email: "admin@example.com", password: "admin123", role: "Admin" },
      { name: "Editor User", email: "editor@example.com", password: "editor123", role: "Editor" },
      { name: "Viewer User", email: "viewer@example.com", password: "viewer123", role: "Viewers" }
    ];

    for (let u of users) {
      const hash = await User.hashPassword(u.password);

      await new
        User({ name:
        u.name, email:
        u.email,
        passwordHash: hash,  // ✅ Correct field
        role: u.role
      }).save();
    }

    console.log("\n🌱 Seed complete!");
    process.exit(0);
  } catch (err) {
    console.error("❌ Error:",
    err); process.exit(1);
  }
})();
```

2. App.js

```
const express = require("express");


const cors = require("cors");


require("dotenv").config();




const authRoutes = require("./routes/auth");
const userRoutes = require("./routes/users");

const postRoutes = require("./routes/posts");


const app = express();


app.use(cors());

app.use(express.json());


// ✅ AUTH FIRST

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


// ✅ USERS ROUTE

app.use("/users", userRoutes);


// ✅ POSTS ROUTE

app.use("/posts", postRoutes);


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

  { res.send("✅ RBAC API

  Running");

});


module.exports = app;
```

3. Auth.js

```
const express = require("express");
const jwt = require("jsonwebtoken");
const User = require("../models/User");
const router = express.Router();

router.post("/login", async (req, res) => {
try {
const user = await User.findOne({ email: req.body.email });

if (!user || !(await user.verifyPassword(req.body.password))) {
return res.status(401).json({ message: "Invalid credentials" });
}



const token = jwt.sign(
{ id: user._id, role: user.role },
process.env.JWT_SECRET,
{ expiresIn: "1h" }
);

res.json({
token,
role: user.role,
name: user.name,
email: user.email
});



} catch (err) {
res.status(500).json({ message: "Server error" });
}
});


module.exports = router;
```

4.Model/Post.js

```
const mongoose = require("mongoose");

const postSchema = new mongoose.Schema(
  {
    title: { type: String, required: true },
    content: { type: String, required: true },

    // ✅ Author reference (User ID)
    author: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true
    }
  },
  { timestamps: true }
);

// ✅ Fix overwrite error (important)
module.exports = mongoose.models.Post || mongoose.model("Post", postSchema);
```

REFERENCES

- Suneel Kumar. 2023. *Implementing Role-Based Access Control (RBAC) in Node.js*. Medium. (“RBAC … restricts access based on user roles”) Medium
- Jayant Choudhary. 2023. *Building Role-Based Access Control (RBAC) in Node.js and Express.js*. Medium. (“Step-by-step walkthrough for integrating RBAC”) Medium
- MongoDB Documentation. “Role-based Access Control (RBAC)” article. MongoDB+2Medium+2
- DataSunrise. *RBAC in MongoDB – Knowledge Center*. (“Roles, privileges, user assignment…” examples) DataSunrise
- Rabindra Tamang. 2024. *How to implement Role-Based Access Control (RBAC) in Node.js applications*. dev.to. DEV Community
- Research Paper: Eeshan Gupta et al. 2021. *Attribute-Based Access Control for NoSQL Databases* (focus on MongoDB). National Science Foundation
- Research Paper: “Analysis of Role-Based Access Control Methods in NoSQL Databases”. ResearchGate
- Research Paper: David F. Ferraiolo, D. Richard Kuhn. 2009. *Role-Based Access Controls*. (classic foundational paper) arXiv
- GeeksforGeeks. 2025. *Configure Role-Based Access Control in MongoDB*. (“How to create user-defined roles, built-in roles, etc.”) GeeksforGeeks
- StackOverflow Q&A: “MongoDB + Node JS + Role Based Access Control (RBAC)”. stackoverflow.com

BIBLIOGRAPHY

Extensive documentation and online resources were referred to during the development of the **RBAC-MERN Project**, ensuring best practices in design, security, and implementation. The following sources were primarily consulted:

- **React Official Documentation** — for understanding hooks (*useState*, *useEffect*), component architecture, and efficient state handling.
- **Node.js & Express.js Documentation** — for backend server configuration, route handling, and middleware integration.
- **MongoDB & Mongoose Documentation** — for defining data schemas, managing collections, and executing CRUD operations.
- **React Router Documentation** — for implementing route-based access control and secure navigation.
- **JSON Web Token (JWT) Documentation** — for secure authentication workflows and token validation mechanisms.
- **Redux Toolkit Documentation** — for managing centralized application state and role-based UI rendering.
- **Axios Documentation** — for secure and structured communication between the frontend and backend via REST APIs.
- **Docker Documentation** — for containerizing the backend and database components, enabling consistent deployment across environments.
- **OWASP Security Guidelines** — for adhering to web application security standards and preventing common vulnerabilities.
- **Online Learning Resources & Communities** — Tutorials and articles from platforms such as *FreeCodeCamp*, *CSS-Tricks*, and *Medium* were studied to apply industry best practices in authentication, authorization, and UI security.

Stack Overflow was also frequently used to resolve technical issues related to JWT middleware, CORS configuration, and React integration challenges.