# Role-Based Access Control System using MERN Stack

**A PROJECT REPORT**

***Submitted by***

***Aman kumar (23bcs11038)***

***Sahil pal(_)***

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

COMPUTER SCIENCE ENGINEERING


**Chandigarh University**

JULY 2025 – NOVEMBER 2025

**BONAFIDE CERTIFICATE**

A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Technology in Computer Science and Engineering

Submitted By:
Shail Pal
Aman Kumar (23BCS11038)

Under the Supervision of:
Mr. Ojash Sir

Department of Computer Science and Engineering
Chandigarh University
CSE – 3rd Year

BONAFIDE CERTIFICATE
Certified that this project report titled “ROLE-BASED ACCESS CONTROL SYSTEM USING MERN STACK”
is the bonafide work of “Shail Pal and Aman Kumar (23BCS11038)” who carried out the project work under my supervision.

SIGNATURE
Mr. Ojash Sir
SUPERVISOR
Department of Computer Science and Engineering

SIGNATURE
Dr. Gagandeep Singh
HEAD OF THE DEPARTMENT
CSE 3rd Year

Submitted for the project viva-voce examination held on _____

INTERNAL EXAMINER EXTERNAL EXAMINER

## TABLE OF CONTENTS

## ABSTRACT

With the growing dependence on cloud and web applications, access control mechanisms have become essential to protect sensitive data and operations.
This project, “Role-Based Access Control System using MERN Stack”, presents a secure and scalable architecture to manage permissions dynamically based on user roles.
It uses MongoDB, Express.js, React.js, and Node.js (MERN) for a full-stack implementation, with JWT (JSON Web Tokens) providing secure authentication and authorization.
The system supports multiple roles—Admin, Editor, and Viewer—each governing what actions are permitted across the system.
It also demonstrates real-time UI restriction, route guarding, and ownership validation, ensuring that no unauthorized user can access protected resources.
This project serves as a foundational model for enterprise-level access management in full-stack applications.


## CHAPTER 1 – INTRODUCTION

### 1.1 Identification of Need

In the modern digital era, web applications handle massive volumes of user data.
With this growth comes the challenge of ensuring that only authorized users have access to specific data and functionalities.
Traditional authentication systems allow any authenticated user to access resources, which can lead to data breaches and unauthorized modifications.
To mitigate these risks, Role-Based Access Control (RBAC) systems have emerged as a standard security model that grants permissions based on user roles rather than individual identities.

This project fulfills the need for a fine-grained RBAC solution implemented using the MERN Stack.
The system provides a framework that not only authenticates users securely but also enforces access rules dynamically across both backend APIs and frontend interfaces.

### 1.2 Identification of Problem

Existing access control systems are often rigid and fail to support evolving role structures in modern applications.
They may also lack UI-level control, allowing restricted actions to appear clickable to unauthorized users.
Additionally, many implementations ignore data ownership—meaning users can sometimes modify content they don’t own.

Our project addresses these issues through:

Role-based API authorization

Ownership checks at the database query level

Dynamic front-end restrictions based on permissions

The solution ensures that the principle of least privilege is maintained throughout the application.

### 1.3 Identification of Tasks

The major tasks involved in this project include:

Designing and storing a role-permission matrix in configuration files.

Implementing JWT-based authentication and secure token handling.

Building Express.js middleware to verify user permissions dynamically.

Integrating MongoDB query-level ownership checks for Editors.

Developing a React frontend that enables/disables features based on roles.

Creating an Admin dashboard for user and role management.

### 1.4 Timeline

| Week | Task Description |
|---|---|
| Week 1 | Project setup, database configuration, user schema design |
| Week 2 | Implementation of JWT authentication and user registration |
| Week 3 | Role and permission matrix setup with route protection |
| Week 4 | Frontend development (React + Tailwind UI) |
| Week 5 | Integration testing and deployment using Docker |

### 1.5 Organization of Report

Chapter 1 provides an introduction, the need for RBAC, and tasks completed.

Chapter 2 reviews existing RBAC approaches and background theory.

Chapter 3 describes the design and implementation of the system.

Chapter 4 analyzes results and presents validation outcomes, followed by a conclusion and future scope.

## CHAPTER 2 – LITERATURE REVIEW / BACKGROUND STUDY

### 2.1 Historical Context

Access control is one of the oldest topics in computer security.
Earlier systems relied on Discretionary Access Control (DAC) and Mandatory Access Control (MAC), which lacked flexibility.
In 1996, NIST proposed the Role-Based Access Control (RBAC) model that defined access rights through predefined roles rather than individual users.

## 2.2 Existing Solutions

Firebase Authentication: Simplifies user login but lacks customizable permission layers.

AWS IAM: Provides enterprise-grade access management but is complex for small applications.

Keycloak: Offers complete user and role management but requires heavy configuration.

Our MERN-based RBAC solution provides the same level of control within a lightweight, developer-friendly structure suitable for startups and academic projects.

## 2.3 Bibliometric Analysis

Research in RBAC highlights its adaptability to various domains such as banking, education, healthcare, and enterprise IT.
It significantly reduces the administrative workload by managing users in groups (roles) rather than individually.

## 2.4 Review Summary

RBAC offers several advantages:

Centralized access control

Enhanced security and traceability

Simplified administration

Improved scalability

However, traditional RBAC systems can become rigid. The goal of this project is to make RBAC more adaptive and UI-aware using MERN.

## 2.5 Problem Definition

Design and implement a full-stack RBAC system that provides:

Authentication (Login/Register)

Authorization (Role-based API control)

Ownership-based data access

Real-time UI restrictions

## 2.6 Objectives

Build a modular backend API using Node.js & Express.

Store users, roles, and posts in MongoDB.

Secure authentication using JWT & bcrypt.

Enforce access through middleware and route protection.

Display UI dynamically based on user roles.

Provide Admin tools for management.

## CHAPTER 3 – DESIGN FLOW / PROCESS

### 3.1 System Architecture

The system is divided into three layers:

Frontend (React): Handles UI rendering, route guarding, and dynamic components.

Backend (Node.js/Express): Manages APIs, JWT verification, and role-based access.

Database (MongoDB): Stores users, roles, and content with ownership references.

Architecture Flow
User logs in → Backend generates JWT token containing user role.

Token stored in cookies/localStorage.

Each API request validates the token and checks permissions.

Frontend adjusts available controls based on user role.

### 3.2 Methodology

Backend Implementation
Role definitions are stored in config/roles.js.

Middleware authorize(role) checks permissions before executing routes.

Ownership enforcement uses MongoDB filters ({ authorId: req.user.id }).

Frontend Implementation
React routes guarded using <PrivateRoute> components.

Role-based components (like “Edit” button) are shown conditionally.

Tooltips display reason when a button is disabled (e.g., “Access Restricted”).

### 3.3 Technologies Used

| Technology | Purpose |
| --- | --- |
| MongoDB | Stores users, posts, and roles |
| Express.js | Handles RESTful API routing |
| React.js | Builds responsive UI |

| | |
|---|---|
| Node.js | Backend runtime environment |
| JWT | Authentication & role encoding |
| bcrypt.js | Password hashing |
| Tailwind CSS | Styling framework |
| Docker | Deployment & environment setup |

### 3.4 Database Schema

User Model:

userId

username

password (hashed)

role (Admin / Editor / Viewer)

Post Model:

postId

title

content

authorId

timestamps

## CHAPTER 4 – RESULTS ANALYSIS AND VALIDATION

### 4.1 Observations

The RBAC system was successfully implemented and tested.
Each role performed actions limited to its assigned capabilities.

| Role | Permissions |
|---|---|
| Admin | Full CRUD + User Management |
| Editor | CRUD on Own Posts Only |
| Viewer | Read-Only Access |

### 4.2 Validation

The system was validated through:

Unit tests on middleware functions.

Integration testing with multiple roles.

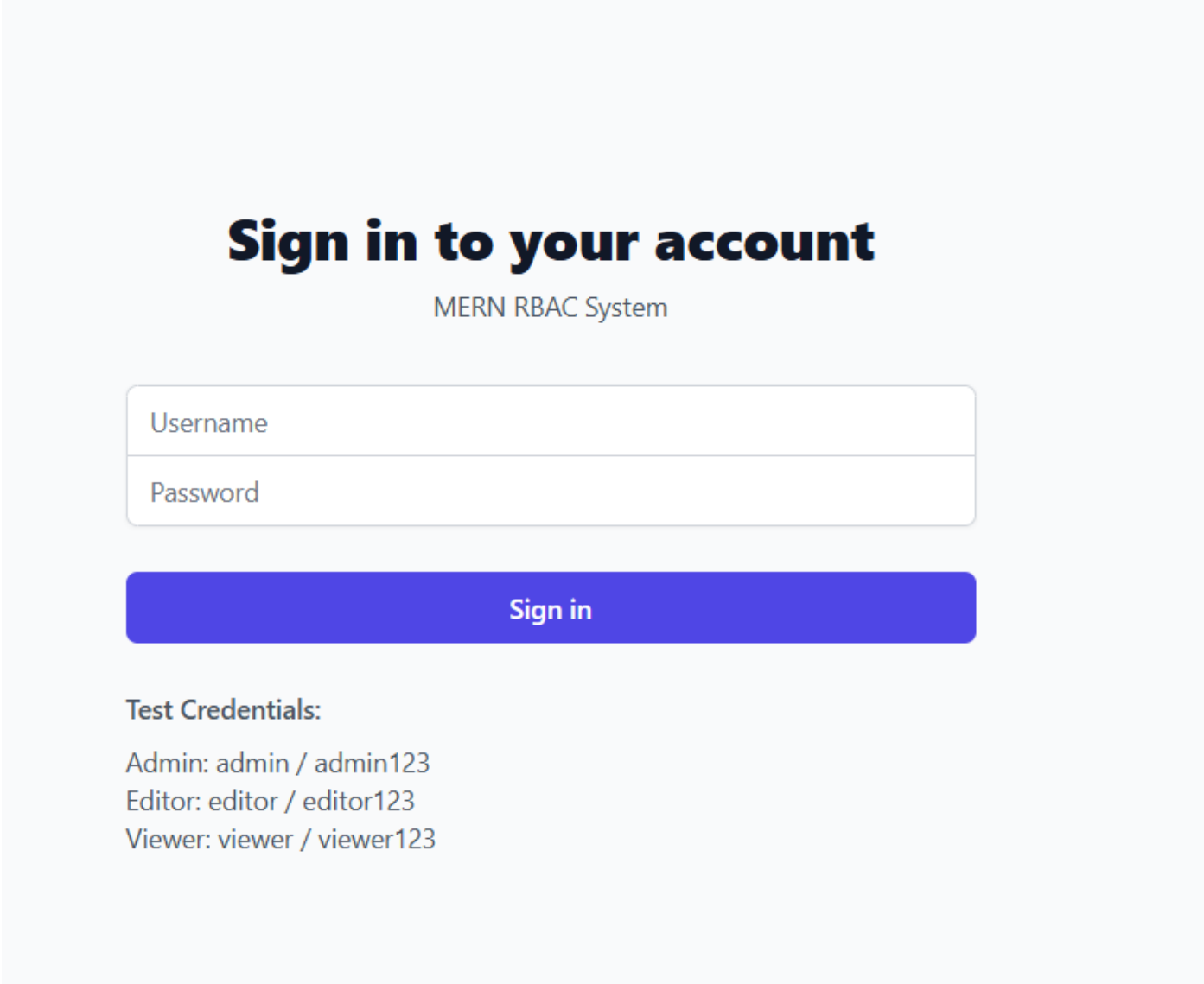
UI testing to ensure hidden controls for unauthorized roles.

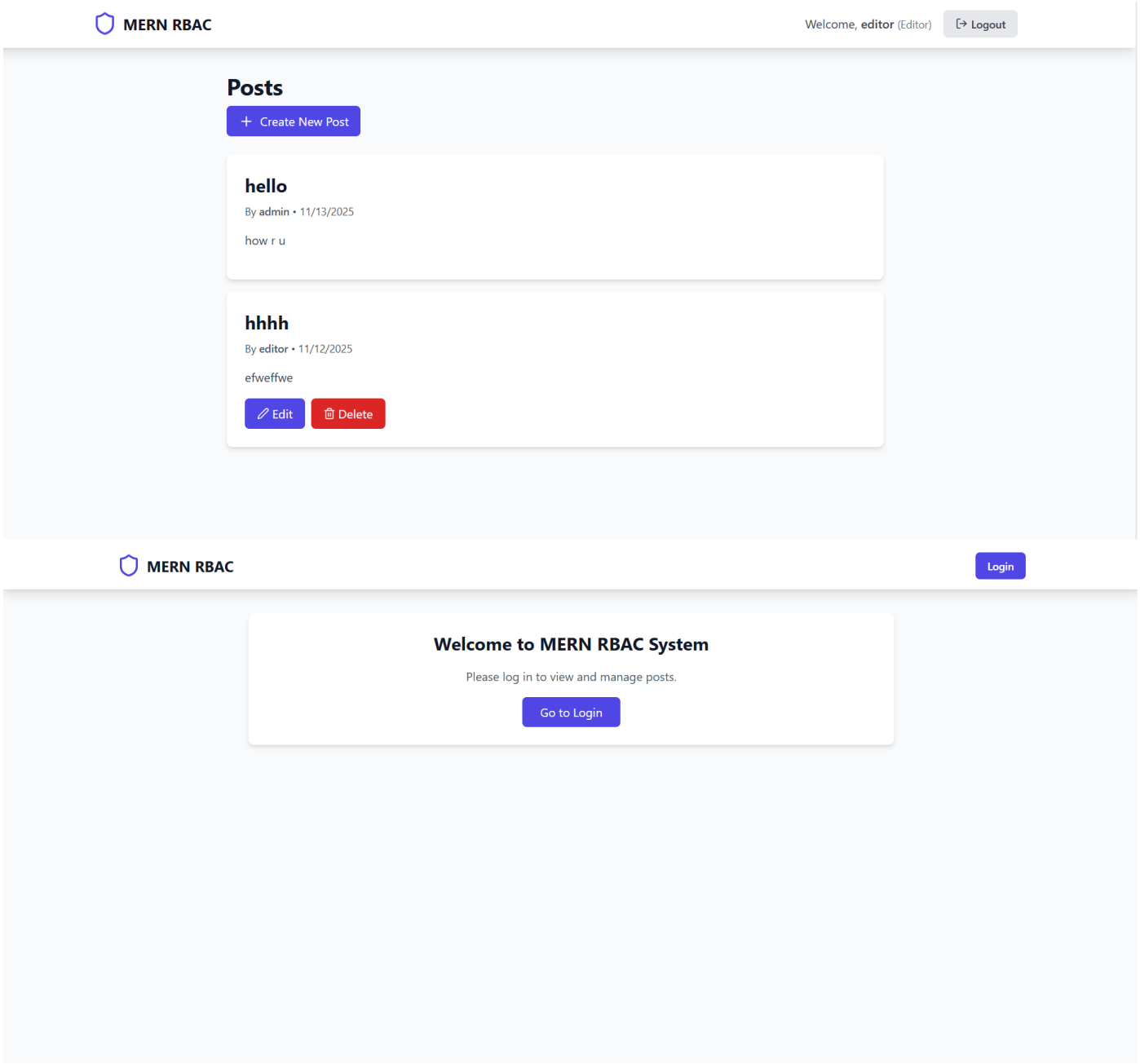
Error codes:

401 Unauthorized: Invalid/expired JWT.

403 Forbidden: Role lacks permission for requested action.

4.3 Sample Screenshots






## CONCLUSION AND FUTURE WORK

The Role-Based Access Control System using MERN Stack successfully demonstrates secure, scalable, and role-based permission handling in full-stack applications.
It integrates JWT authentication, middleware-based authorization, and UI-level control for a comprehensive security model.

Future Enhancements
Two-Factor Authentication (2FA)

Audit Logs for Admins

Email Verification System

Graph-based Permission Visualization

Integration with Cloud IAM tools

## REFERENCES

Sandhu, R.S. et al. (1996). Role-Based Access Control Models. IEEE Computer.

MongoDB Documentation – Access Control Implementation.

Node.js Security Practices – OWASP Foundation.

React Documentation – Conditional Rendering.

JWT.io – JSON Web Token Specifications.

Tailwind CSS Docs – Component Design for Dashboards.