COLLEGE CODE: 8203

COLLEGE NAME: A.V.C. College of Engineering

DEPARTMENT: Information Technology

STUDENT NM-ID: 2CF9A0E1E49C80C86124EED021004FAD

ROLL NO: 23IT114

DATE: 08/09/2025

Completed the project named as Phase 1

TECHNOLOGY PROJECT NAME: *USER AUTHENTICATION SYSTEM*

SUBMITTED BY,

VEDIKA.D

NAME: Vedika.D

MOBILE NO: 9486786011

Problem Statement:

Modern web applications handle sensitive data such as personal details, financial information, and private communications. However, insecure authentication mechanisms make them vulnerable to unauthorized access, session hijacking, brute-force attacks, and data breaches.

Traditional session-based authentication often struggles with scalability in distributed systems and lacks flexibility for role-based access control (RBAC). Without a secure, token-based approach, applications cannot reliably protect routes, verify user identity across multiple services, or prevent misuse of credentials.

Therefore, there is a strong need to design a robust User Authentication System that ensures:

- Secure login/logout functionality using hashed credentials (bcrypt).
- JWT-based token management to enable stateless authentication.
- Session handling for persistent user experience.
- Route protection to restrict access to authorized users.
- Role-based access control (admin, user, moderator, etc.) for fine-grained authorization.

This system will provide a scalable, secure, and efficient authentication solution suitable for modern applications requiring both data protection and user-specific access privileges.

Users & Stakeholders:

1. End Users

These are the people who will directly use the authentication system:

- **Regular Users**:
    - Sign up, log in, and log out.
    - Access personal dashboards or features based on their role.
    - Store sessions securely without needing to log in repeatedly.

- **Admin Users**:
    - Manage system users (create, update, delete accounts).
    - Assign roles (e.g., user, moderator, admin).
    - Monitor system logs and security alerts.
- **Moderators (if applicable)**:
    - Limited admin-like permissions (e.g., approve content, restrict users).
    - Cannot manage system settings or other admins.

2. Stakeholders

These individuals or groups benefit from or are affected by the authentication system:

- **Application Developers**
    - Responsible for designing and implementing the system using Node.js, Express, MongoDB, bcrypt, and JWT.
    - Ensure secure coding practices and compliance with security standards.
- **System Administrators / IT Security Team**
    - Maintain the server, databases, and tokens.
    - Monitor logs for suspicious activity and prevent breaches.
- **Business Owners / Product Managers**
    - Ensure the authentication system aligns with business goals (e.g., user trust, data protection, compliance with privacy laws like GDPR).
    - Care about scalability, performance, and customer satisfaction.
- **End Customers / Clients of the Application**
    - Indirect beneficiaries who rely on secure authentication for data protection.

  - Their trust in the system depends on the reliability of login/logout and role-based protection.

User Stories:

1. Regular User

- *As a user, I want to sign up with my email and password, so that I can create a secure account.*
- *As a user, I want my password to be stored securely, so that no one can access it even if the database is compromised.*
- *As a user, I want to log in and receive a session/JWT token, so that I can access protected features without re-entering credentials repeatedly.*
- *As a user, I want to log out, so that no one else can use my account on this device.*
- *As a user, I want error messages when I enter wrong credentials, so that I know what went wrong.*

---

2. Admin User

- *As an admin, I want to manage user accounts (create, update, delete), so that I can maintain the system effectively.*
- *As an admin, I want to assign roles (user, moderator, admin), so that users have appropriate permissions.*
- *As an admin, I want to view security logs, so that I can monitor suspicious login attempts.*
- *As an admin, I want to protect admin-only routes, so that unauthorized users cannot access system settings.*

---

3. Moderator (Optional Role)

- *As a moderator, I want limited administrative privileges, so that I can manage users/content without full system control.*
- *As a moderator, I want to access only specific protected routes, so that I do not interfere with admin operations.*

---

4. Application Developer

- *As a developer, I want to integrate JWT-based authentication, so that APIs are secure and stateless.*
- *As a developer, I want middleware to validate tokens, so that unauthorized requests are blocked automatically.*
- *As a developer, I want bcrypt for hashing passwords, so that sensitive credentials are never stored in plain text.*

---

5. Business Owner / Product Manager

- *As a business owner, I want a reliable authentication system, so that users trust the application.*
- *As a business owner, I want role-based access control, so that I can support different user levels (free, premium, admin).*
- *As a business owner, I want compliance with security standards, so that the system avoids legal issues and builds credibility.*

---

6. System Administrator / Security Team

- *As a system administrator, I want to monitor login attempts and failed logins, so that I can detect brute-force attacks.*
- *As a system administrator, I want to invalidate tokens and sessions, so that compromised accounts can be secured.*
- *As a system administrator, I want audit logs of authentication events, so that I can ensure accountability and traceability.*

MVP Features – User Authentication System:

1. User Account Management

- **User Signup (Registration)**
    - Create an account with username/email & password.

    - Password hashing with **bcrypt** before storing in MongoDB.
- **User Login**
    - Validate credentials against database.
    - On success, generate **JWT token** for authentication.
- **User Logout**
    - Invalidate token/session.
    - Clear stored JWT from client (cookies/localStorage).

2. Authentication & Security

- **Password Hashing (bcrypt)** to secure stored credentials.
- **JWT Token Generation & Verification** for stateless authentication.
- **Session Handling** (optional hybrid approach with cookies).
- **Middleware for Protected Routes** to check token validity before granting access.

3. Authorization (RBAC – Role-Based Access Control)

- **Basic Roles:**
    - *User*: Access personal data & standard features.
    - *Admin*: Manage users, roles, and sensitive routes.
    - *Moderator (Optional)*: Limited admin rights.
- **Route Protection Based on Roles** (e.g., /admin accessible only by admin).

4. Error Handling & Validation

- Proper error messages for:
    - Invalid credentials.
    - Expired or invalid tokens.

    - Unauthorized role access.
- Input validation (strong password policy, valid email format).

5. Logging & Monitoring

- Log authentication attempts (success & failure).
- Track failed logins for security analysis.

6. Developer/Integration Features

- **API Endpoints** for login, signup, logout, token validation.
- **Scalable Design** (stateless JWT ensures easy integration with microservices).

Wireframes/API Endpoint List:

A. Login Page

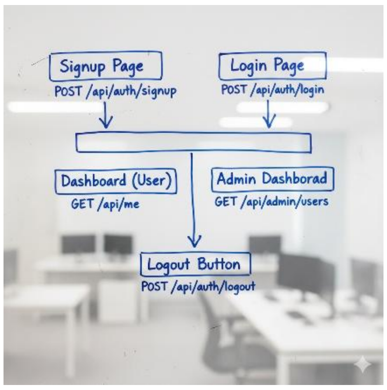
- **Fields:** Email/Username, Password
- **Buttons:** Login, Forgot Password, Signup
- **Features:**
    - Show/hide password
    - Error messages for invalid credentials

B. Signup Page

- **Fields:** Name, Email, Password, Confirm Password, Role (if admin adds user)
- **Buttons:** Signup, Login
- **Features:**
    - Password strength indicator

    - Validation messages

C. Dashboard (Post-login)



- **Sections:**
    - Welcome message
    - Role-based content:
        - Regular user: Personal info, profile settings
        - Admin: User management panel, role assignment.
        - **Buttons/Actions:** Logout, Edit Profile

Method	Endpoint	Description	Access
POST	/api/signup	Register new user (bcrypt hashed)	Public
POST	/api/login	Authenticate user & issue JWT	Public
GET	/api/logout	Invalidate session/JWT	Authenticated
GET	/api/profile	Fetch logged-in user profile	Authenticated
GET	/api/admin	Admin-only route (check role middleware)	Admin only

Acceptance Criteria:

1. Signup / Registration

- Users must be able to register with a valid email and password.
- Passwords must be hashed using bcrypt before storing in the database.
- Duplicate email registration should be prevented.
- On successful registration, a confirmation message is returned.

2. Login / Authentication

- Users must be able to log in with valid credentials.
- On successful login, a JWT token is issued.
- Invalid credentials should return a clear error message (401 Unauthorized).

- Logged-in users must have a session or token stored securely (cookies/localStorage).

3. Logout

- Users must be able to log out, invalidating the current session or token.
- Accessing protected routes after logout must return 401 Unauthorized.

4. Protected Routes

- All protected endpoints must require a valid JWT token.
- Unauthorized access attempts must be rejected with 401 Unauthorized.
- Access to admin-specific routes must require the admin role.

5. Role-Based Access Control (RBAC)

- Users with roles other than admin must be restricted from admin-only actions.
- Admin users must be able to create, update, or delete users.
- Role changes must take effect immediately for all affected users.

6. Password Security

- Passwords must meet minimum strength criteria (e.g., minimum 8 characters, at least one uppercase, one lowercase, one number).
- Passwords must be stored securely using bcrypt hashing.
- Password reset or recovery functionality (optional) must be secure and token-based.

7. Error Handling & Validation

- Input validation errors must return meaningful messages to the client.
- Server errors should return appropriate HTTP status codes (500, 400, 404) with messages.
- All responses must follow consistent JSON structure.

8. Performance & Scalability

- The system must handle multiple concurrent login sessions without crashing.

- JWT-based stateless authentication must allow horizontal scaling of the application.

9. Compliance & Security

- Tokens should have an expiration time (e.g., 1 hour) and use secure signing keys.
- Sensitive data must never be exposed in API responses.
- System should be resilient to common attacks like brute-force, SQL injection, or XSS.