Project Title: AuthShield: A Secure Role-Based Access Control System

Role-Based Access Control (RBAC) System for VRV Security Backend Developer Assignment

1. Introduction

This project implements a secure **Role-Based Access Control (RBAC)** system for authentication and authorization, aligning with VRV Security's commitment to building robust cybersecurity solutions. RBAC is a critical aspect of security, ensuring that access to resources is granted only based on user roles, minimizing risks associated with unauthorized access.

The system includes features such as user registration, login, secure token-based session management, and role-based access restrictions. By employing secure practices, this project demonstrates a comprehensive understanding of authentication, authorization, and RBAC, which are vital to building secure and scalable backend systems.

2. Objectives

The objectives of the project are as follows:

1. **Authentication**: Build a secure authentication mechanism to validate users and grant access through JWT tokens.
2. **Authorization**: Implement mechanisms to ensure that users can only access resources or perform actions allowed by their roles.
3. **Role-Based Access Control**: Design a system where roles (e.g., Admin, HOD, and Student) dictate permissions, ensuring fine-grained control over access.
4. **Secure Implementation**: Follow best practices like password hashing, token expiration, and structured API responses.
5. **User Roles & Hierarchy**:
    - **Admin**: Manage HODs and oversee the system.
    - **HOD (Head of Department)**: Manage students within their department.
    - **Student**: Access personal profile information.

3. Implementation

Technologies Used

- **Backend Framework**: Node.js with Express.js.
- **Database**: MongoDB Atlas for data storage and Mongoose for schema modeling.
- **Authentication**: JSON Web Tokens (JWT) for secure session management.

- **Encryption**: Passwords are hashed using the bcryptjs library.

3.1 Key Features

1. **User Registration**:
    - Allows users to register with role-based attributes such as username, email, password, role, and department.
    - Validates unique emails to prevent duplication.
2. **User Login**:
    - Validates user credentials and generates a secure JWT token upon successful login.
3. **Role-Based Access Control (RBAC)**:
    - Uses middleware to restrict access to specific endpoints based on user roles.
    - Example: Only an Admin can view HOD details and there respective department students, while HODs can only view students from their department.
4. **Secure Password Storage**:
    - Passwords are hashed using bcrypt to ensure secure storage in the database.
5. **Token-Based Authentication**:
    - JWT tokens are issued upon login and verified during subsequent API requests for secure session handling.

4. Project Structure

```
project/
├── controllers/
|   ├── authController.js      # Handles registration, login, and logout logic.
|   ├── userController.js      # Manages user-related data.
├── middleware/
|   ├── authMiddleware.js      # Verifies JWT tokens for secure API access.
|   ├── roleMiddleware.js      # Enforces role-based access restrictions.
├── models/
|   ├── user.js                # User schema with roles and permissions.
├── routes/
|   ├── authRoutes.js          # Routes for authentication (register/login/logout).
|   ├── userRoutes.js          # Routes for user profile and role-based endpoints.
├── config/
```

```
|   ├── db.js              # MongoDB connection configuration.
├── .env                  # Environment variables for sensitive data.
├── index.js              # Main entry point of the application.
├── package.json          # Node.js dependencies and metadata.
```

5. API Documentation

Authentication APIs

Endpoint	HTTP Method	Description	Access
/api/auth/register	POST	Registers a new user. Requires username, email, password, role, and department.	Public
/api/auth/login	POST	Logs in a user with email and password. Returns a JWT token.	Public
/api/auth/logout	POST	Logs out a user.	Authenticated Users

User APIs

Endpoint	HTTP Method	Description	Access
/api/users/profile	GET	Retrieves the logged-in user's profile data.	Authenticated Users
/api/users/hod/students	GET	Retrieves students for HODs based on department.	HOD
/api/users/admin	GET	Retrieves HOD details and respective department students	Admin

Middleware Functions

1. authMiddleware:
    - Ensures a valid JWT token is included in the Authorization header for secure API access.
2. roleMiddleware:
    - Restricts access to specific roles, e.g., only Admins can view HOD details.

6. Sample API Workflow

1. **User Registration (Admin/HOD/Student)**:
    - **Request**:

POST http://localhost:5000/api/auth/register

```
{
  "username": "dilesh",
  "email": "dilesh@gmail.com",
  "password": "123456",
  "role": "Student",
  "department": "CSE"
}
```

Response:

```
{
    "message": "User registered successfully"
}
```

User Login:

- **Request**:

POST

http://localhost:5000/api/auth/login

```
{
  "email" : "admin@gmail.com",
  "password" : "admin"
}
```

Response:

{

"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3NDQ5YTkyNzlmZjZiNzlkMzU1NDc4ZSIsInJvbGUiOiJBZG1pbiIsImVtYWlsIjoiYWRtaW5AZ21haWwuY29tIiwidXNlcm5hbWUiOiJhZG1pbiIsImlhdCI6MTczMjU5MzE1MiwiZXhwIjoxNzMyNTk2NzUyfQ.s-XiUv_i656eKi6infK2YM2Y0RUKkD-qXuQJbSEWHJE"

}

Access User Profile:

- **Request** (with token):

GET /api/users/profile

Authorization: Bearer jwt-token

- **Response**:

{

"user": {

"id": "123456",

"username": "JohnDoe",

"email": "john@example.com",

"role": "Student",

"department": "CSE"

}

}

1. **View Admin Panel**:
    - **Request**:

    GET http://localhost:5000/api/users/admin

    Authorization: Bearer jwt-token

    - **Response**:


```
{
    "user": {
        "_id": "67449a9279ff6b79d355478e",
        "username": "admin",
        "email": "admin@gmail.com",
        "role": "Admin",
        "department": null,
        "__v": 0
    },
    "hods": [
        {
            "hod": {
                "_id": "67449a6f79ff6b79d355478a",
                "username": "Rekha",
                "email": "rekha@gmail.com",
                "password":
"$2a$10$M7DVKc3CGEcKuhGT5mfT5eSxhjOMs2aXmEzIbrmIcliiNUag86ZQ6",
                "role": "HOD",
                "department": "CSE",
                "__v": 0
            },
            "students": [
                {
                    "_id": "67449a4579ff6b79d3554783",
                    "username": "ram",
                    "email": "ram@gmail.com",
                    "role": "Student",
```

```
                    "department": "CSE",
                    "__v": 0
                },
                {
                    "_id": "67449f2dadaa48da8453e699",
                    "username": "sanjay",
                    "email": "sanjay@gmail.com",
                    "role": "Student",
                    "department": "CSE",
                    "__v": 0
                },
                {
                    "_id": "674547139d8d2423b31df2df",
                    "username": "dilesh",
                    "email": "dilesh@gmail.com",
                    "role": "Student",
                    "department": "CSE",
                    "__v": 0
                }
            ]
        }
    ]
}
```


View HOD Panel :

- **Request** (with token):

  GET http://localhost:5000/api/users/hod/students

  Authorization: Bearer jwt-token

- **Response**:


```
{
    "user": {
        "_id": "67449a6f79ff6b79d355478a",
        "username": "Rekha",
        "email": "rekha@gmail.com",
        "role": "HOD",
        "department": "CSE",
        "__v": 0
    },
    "students": [
        {
            "_id": "67449a4579ff6b79d3554783",
            "username": "ram",
            "email": "ram@gmail.com",
            "role": "Student",
            "department": "CSE",
            "__v": 0
        },
        {
            "_id": "67449f2dadaa48da8453e699",
            "username": "sanjay",
```

```
            "email": "sanjay@gmail.com",
            "role": "Student",
            "department": "CSE",
            "__v": 0
        },
        {
            "_id": "674547139d8d2423b31df2df",
            "username": "dilesh",
            "email": "dilesh@gmail.com",
            "role": "Student",
            "department": "CSE",
            "__v": 0
        }
    ]
}
```


7. Security Features

1. **Password Hashing**: Ensures passwords are stored securely using bcrypt.
2. **JWT Authentication**: Provides stateless, secure session management with token expiration.
3. **Role-Based Middleware**: Enforces granular access control for APIs.

8. Conclusion

This project demonstrates a practical implementation of **Role-Based Access Control (RBAC)** using Node.js, Express.js, and MongoDB. By adhering to security best practices and creating role-specific access, this system ensures secure and efficient user management. The modular architecture allows easy scalability and integration into larger systems, making it a robust solution for VRV Security's requirements.