Campus Resource Portal – Backend Project Report

Project Summary

The Campus Resource Portal is a Node.js-based backend that supports student and admin roles. It allows users to register, authenticate, and view or manage announcements/posts based on role.

Tech Stack: Node.js, Express, MongoDB, Mongoose, JWT, bcrypt
Security: JWT for Auth, RBAC for Access Control
Data: Users & Posts

Folder Structure

```
backend/
├── config/
├── models/
│   ├── User.js             # User schema (name, email, password, role)
│   └── Post.js             # Post schema (title, message, status, createdBy)
├── services/
│   ├── userService.js      # User registration/login logic
│   ├── postService.js      # Post CRUD operations
├── controllers/
│   ├── authController.js   # Handles register/login requests
│   └── postController.js   # Handles post-related requests
├── auth/
│   ├── authService.js      # Password hashing, token generation, validation
│   └── rbac.js             # protect (JWT middleware) + allowRoles()
├── routes/
│   ├── authRoutes.js       # Routes: /api/auth/register, /login
│   └── postRoutes.js       # Routes: /api/posts/
├── .env                    # MONGO_URI, JWT_SECRET, PORT
└── server.js               # Express app entry point
```

JWT + RBAC Flow (CLI Diagram)

```
+---------+        +------------+         +-------------+
| Client  |  --->  |  /login    |   --->  |   DB (User) |
+---------+        +------------+         +-------------+
   |                   |                     |
   | <--- JWT Token -----+                       |
   |                                        |
   |--- Auth Request w/ JWT --> [ protect() middleware ]
                              |
                       +--------------------+
                       | Verify JWT Token   |
                       +--------------------+
                              |
                        [ allowRoles('admin') ]
                              |
                       +--------------------+
                       | Check user.role    |
                       +--------------------+
                              |
                     Access granted or denied
```

Code Overview (Major Files)

1. config/database.js

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


const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('MongoDB connected');
  } catch (err) {
    console.error(err);
    process.exit(1);
  }
};
```

```
module.exports = connectDB;
```

---

2. models/User.js

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

const userSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  password: String,
  role: { type: String, enum: ['student', 'admin'], default: 'student' }
});

module.exports = mongoose.model('User', userSchema);
```

---

3. auth/authService.js

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

const generateToken = (user) => {
  return jwt.sign({ id: user._id, role: user.role }, process.env.JWT_SECRET, {
    expiresIn: '1d',
  });
};

const hashPassword = (password) => bcrypt.hash(password, 10);
const comparePasswords = (input, hash) => bcrypt.compare(input, hash);

module.exports = { generateToken, hashPassword, comparePasswords };
```

---

4. auth/rbac.js

```
const jwt = require('jsonwebtoken');
const protect = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) return res.status(401).json({ msg: 'No token' });


  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET);
    next();
  } catch (err) {
    res.status(401).json({ msg: 'Invalid token' });
  }
};
const allowRoles = (...roles) => (req, res, next) => {
  if (!roles.includes(req.user.role)) {
    return res.status(403).json({ msg: 'Access denied' });
  }
  next();
};

module.exports = { protect, allowRoles };
```

---

5. routes/postRoutes.js

```
const express = require('express');
const router = express.Router();
const { protect, allowRoles } = require('../auth/rbac');
const {
  createPost,
  getAllPosts,
  updatePost,
  deletePost,
} = require('../controllers/postController');

router.post('/', protect, allowRoles('admin'), createPost);
router.get('/', protect, getAllPosts);
router.put('/:id', protect, allowRoles('admin'), updatePost);
router.delete('/:id', protect, allowRoles('admin'), deletePost);

module.exports = router;
```

API Collection

Endpoint	Method	Access	Description
/api/auth/register	POST	Public	Register as a student
/api/auth/login	POST	Public	Get JWT for access
/api/posts	GET	Student/Admin	View all posts
/api/posts	POST	Admin only	Create a post
/api/posts/:id	PUT	Admin only	Edit a post
/api/posts/:id	DELETE	Admin only	Delete a post

Sample Request Bodies

Register (Student)

```
POST /api/auth/register

{
  "name": "John Doe",
  "email": "john@student.com",
  "password": "12345678"
}
```

Login

```
POST /api/auth/login

{
  "email": "john@student.com",
  "password": "12345678"
}
```

Create Post (Admin)

```
POST /api/posts

Authorization: Bearer <JWT>

{
  "title": "Exam Notice",
  "message": "Midterms start next week",
  "status": "active"
}
```

Program Flow

Registration

1. authController.register() → userService.registerUser()
2. Password hashed
3. Role set as 'student' by default
4. Token returned

Login

1. authController.login() → userService.loginUser()
2. Password validated
3. JWT issued

Post Creation (Admin Only)

1. postController.createPost() calls service
2. Protected by protect + allowRoles('admin')

Post Viewing (All Authenticated)

1. getAllPosts() returns all posts

Summary

Layer	Responsibility
Models	Mongoose schemas for DB
Services	Business logic
Controllers	Request/response orchestration
Routes	API endpoints + route-level middleware
Auth	JWT issuance, password security, RBAC

Would you like the **PDF version**, **Postman collection**, or **React frontend** next?

Core Features

1. User Registration

- Allows both **students** and **admins** to register.
- Automatically assigns default role as student unless explicitly set as admin.

2. User Login

- Authenticates users with **email and password**.
- Returns a **JWT token** upon successful login for secure session handling.

3. Role-Based Access Control (RBAC)

- Middleware-driven control:
    - protect middleware ensures the user is authenticated.
    - allowRoles('admin') restricts actions based on role.
- Ensures only admins can manage posts, students can only view.

Post Management (Admin Only)

4. Create Post

- Admin can create resource/facility-related posts with:
    - title
    - message
    - status (Active or Closed)
    - Automatically records createdBy field.

5. Update Post

- Admin can **edit** the content and status of posts.

6. Delete Post

- Admin can **delete** any post by ID.

Post Viewing (Student & Admin)

7. View All Posts

- All users (students and admins) can view all available posts.

8. View Single Post

- Any user can fetch a specific post by its ID.

Security Features

9. JWT-Based Authentication

- Every route (except /register, /login) is protected via JWT.
- Tokens include user ID and role.

10. Password Hashing

- Passwords are securely hashed using **bcrypt** before storing.

11. Authorization Middleware

- Prevents students from accessing or modifying admin-level routes.

Structural Features

12. Modular Codebase

- Separated into clean folders:
    - models, routes, services, controllers, auth, config.

13. Environment Config

- Secure .env file usage:
    - PORT, MONGO_URI, JWT_SECRET.

14. CORS Enabled

- Allows requests from other origins like React frontend.

Developer-Friendly Features

15. API Ready for Frontend

- RESTful API design.
- JSON structured responses.
- Status codes (200, 201, 400, 401, 403, 500) for clarity.

16. Easy Testing with Postman

- Token-based testing.
- CRUD operations covered for posts.
- Authentication workflows fully testable.

Optional Extensible Features (Possible Enhancements)

Feature Idea	Description
Admin Seeder	Add script to create initial admin user via CLI.
Timestamps	Use Mongoose timestamps for post logs.
Search & Filter	Add keyword search or status filter to posts.
Email Notifications	Send email on post creation/update (optional).
Logs	Add request logging (e.g., morgan) for audit trails.
