70 MERN Stack Projects — In-Depth Guide (CRUD + JWT + APIs)

Common Architecture for ALL Projects
• Frontend: React (Admin/User dashboards)
• Backend: Node.js + Express
• Database: MongoDB (Mongoose schemas)
• Auth: JWT (Register, Login, Refresh token, Protected routes, Role based access)
• Common APIs:
  POST /api/auth/register
  POST /api/auth/login
  GET /api/auth/me (protected)
• Security: Password hashing, JWT middleware, role guard
• Features: Search, filter, pagination, validations, error handling

1. Student Management System

Purpose: Admin manages students/courses. Students login and view profile and enrolled courses. JWT protects all routes. Role middleware allows only admin to modify data.

Main Modules / Collections:

1. Users (Admin/Student)
2. Students
3. Courses
4. Batches
5. Enrollment

Sample REST APIs (CRUD):

1. POST /api/students
2. GET /api/students
3. PUT /api/students/:id
4. DELETE /api/students/:id
5. POST /api/courses
6. POST /api/enroll

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

2. Employee Management System

Purpose: Admin/HR manages employees. Employees can view their profile. JWT auth with role-based access.

Main Modules / Collections:

1. Users (Admin/HR)
2. Employees
3. Departments
4. Roles

Sample REST APIs (CRUD):

1. POST /api/employees
2. GET /api/employees
3. PUT /api/employees/:id
4. DELETE /api/employees/:id

5. POST /api/departments

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

3. Library Management System

Purpose: Admin manages books. Users can issue/return books. JWT secures transactions.

Main Modules / Collections:

1. Users
2. Books
3. Authors
4. Categories
5. IssueHistory

Sample REST APIs (CRUD):

1. POST /api/books
2. GET /api/books
3. PUT /api/books/:id
4. DELETE /api/books/:id
5. POST /api/issues
6. POST /api/returns

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

4. Hospital Management System

Purpose: Role-based system where doctors see appointments, admin manages everything. JWT for security.

Main Modules / Collections:

1. Users (Admin/Doctor/Reception)
2. Patients
3. Doctors
4. Appointments
5. Reports

Sample REST APIs (CRUD):

1. POST /api/patients
2. GET /api/patients
3. POST /api/appointments
4. PUT /api/appointments/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

5. E-Commerce Platform

Purpose: Users browse and order products. Admin manages products. JWT protects user accounts and orders.

Main Modules / Collections:

1. Users
2. Products
3. Categories
4. Cart
5. Orders

Sample REST APIs (CRUD):

1. POST /api/products
2. GET /api/products
3. POST /api/cart
4. POST /api/orders

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

6. Online Course Management System

Purpose: This project implements Online Course Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Online Course

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/online

2 GET /api/online

3 PUT /api/online/:id

4 DELETE /api/online/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

7. Job Portal System

Purpose: This project implements Job Portal System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Job Portal

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/job

2 GET /api/job

3 PUT /api/job/:id

4 DELETE /api/job/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

8. Online Examination System

Purpose: This project implements Online Examination System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Online Examination

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/online

2 GET /api/online

3 PUT /api/online/:id

4 DELETE /api/online/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

9. Attendance Management System

Purpose: This project implements Attendance Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Attendance
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/attendance
2. GET /api/attendance
3. PUT /api/attendance/:id
4. DELETE /api/attendance/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

10. Task Management System

Purpose: This project implements Task Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Task
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/task
2. GET /api/task
3. PUT /api/task/:id
4. DELETE /api/task/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

11. Project Management Tool

Purpose: This project implements Project Management Tool with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Project Tool
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/project
2. GET /api/project
3. PUT /api/project/:id
4. DELETE /api/project/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

12. Bug Tracking System

Purpose: This project implements Bug Tracking System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Bug Tracking

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/bug

2 GET /api/bug

3 PUT /api/bug/:id

4 DELETE /api/bug/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

13. CRM System

Purpose: This project implements CRM System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. CRM
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/crm
2. GET /api/crm
3. PUT /api/crm/:id
4. DELETE /api/crm/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

14. Inventory Management System

Purpose: This project implements Inventory Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Inventory
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/inventory
2. GET /api/inventory
3. PUT /api/inventory/:id
4. DELETE /api/inventory/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

15. Billing System

Purpose: This project implements Billing System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Billing

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/billing

2 GET /api/billing

3 PUT /api/billing/:id

4 DELETE /api/billing/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

16. Expense Tracker

Purpose: This project implements Expense Tracker with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Expense Tracker
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/expense
2. GET /api/expense
3. PUT /api/expense/:id
4. DELETE /api/expense/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

17. Banking System

Purpose: This project implements Banking System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Banking
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/banking
2. GET /api/banking
3. PUT /api/banking/:id
4. DELETE /api/banking/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

18. Loan Management System

Purpose: This project implements Loan Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Loan

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/loan

2 GET /api/loan

3 PUT /api/loan/:id

4 DELETE /api/loan/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

19. Insurance Management System

Purpose: This project implements Insurance Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Insurance
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/insurance
2. GET /api/insurance
3. PUT /api/insurance/:id
4. DELETE /api/insurance/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

20. Online Voting System

Purpose: This project implements Online Voting System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Online Voting
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/online
2. GET /api/online
3. PUT /api/online/:id
4. DELETE /api/online/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

21. Complaint Management System

Purpose: This project implements Complaint Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Complaint
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/complaint
2 GET /api/complaint
3 PUT /api/complaint/:id
4 DELETE /api/complaint/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

22. Event Management System

Purpose: This project implements Event Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Event
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/event
2. GET /api/event
3. PUT /api/event/:id
4. DELETE /api/event/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

23. College Management System

Purpose: This project implements College Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. College
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/college
2. GET /api/college
3. PUT /api/college/:id
4. DELETE /api/college/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

24. Hostel Management System

Purpose: This project implements Hostel Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Hostel
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/hostel
2 GET /api/hostel
3 PUT /api/hostel/:id
4 DELETE /api/hostel/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

25. Vehicle Service System

Purpose: This project implements Vehicle Service System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Vehicle Service
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/vehicle
2. GET /api/vehicle
3. PUT /api/vehicle/:id
4. DELETE /api/vehicle/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

26. Car Rental System

Purpose: This project implements Car Rental System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Car Rental
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/car
2. GET /api/car
3. PUT /api/car/:id
4. DELETE /api/car/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

27. Hotel Booking System

Purpose: This project implements Hotel Booking System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Hotel Booking

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/hotel

2 GET /api/hotel

3 PUT /api/hotel/:id

4 DELETE /api/hotel/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

28. Travel Management System

Purpose: This project implements Travel Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Travel
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/travel
2. GET /api/travel
3. PUT /api/travel/:id
4. DELETE /api/travel/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

29. Restaurant Management System

Purpose: This project implements Restaurant Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Restaurant
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/restaurant
2. GET /api/restaurant
3. PUT /api/restaurant/:id
4. DELETE /api/restaurant/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

30. Food Ordering System

Purpose: This project implements Food Ordering System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Food Ordering
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/food
2. GET /api/food
3. PUT /api/food/:id
4. DELETE /api/food/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

31. Online Grocery Store

Purpose: This project implements Online Grocery Store with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Online Grocery Store
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/online
2 GET /api/online
3 PUT /api/online/:id
4 DELETE /api/online/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

32. Product Review System

Purpose: This project implements Product Review System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Product Review
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/product
2 GET /api/product
3 PUT /api/product/:id
4 DELETE /api/product/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

33. Courier Management System

Purpose: This project implements Courier Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Courier

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/courier

2 GET /api/courier

3 PUT /api/courier/:id

4 DELETE /api/courier/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

34. Vendor Management System

Purpose: This project implements Vendor Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Vendor

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/vendor

2 GET /api/vendor

3 PUT /api/vendor/:id

4 DELETE /api/vendor/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

35. HR Management System

Purpose: This project implements HR Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 HR

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/hr

2 GET /api/hr

3 PUT /api/hr/:id

4 DELETE /api/hr/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

36. Payroll Management System

Purpose: This project implements Payroll Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Payroll
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/payroll
2. GET /api/payroll
3. PUT /api/payroll/:id
4. DELETE /api/payroll/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

37. Leave Management System

Purpose: This project implements Leave Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Leave
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/leave
2. GET /api/leave
3. PUT /api/leave/:id
4. DELETE /api/leave/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

38. Interview Scheduling System

Purpose: This project implements Interview Scheduling System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Interview Scheduling
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/interview
2. GET /api/interview
3. PUT /api/interview/:id
4. DELETE /api/interview/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

39. Portfolio Management System

Purpose: This project implements Portfolio Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Portfolio

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/portfolio

2 GET /api/portfolio

3 PUT /api/portfolio/:id

4 DELETE /api/portfolio/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

40. Freelancer Platform

Purpose: This project implements Freelancer Platform with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Freelancer
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/freelancer
2. GET /api/freelancer
3. PUT /api/freelancer/:id
4. DELETE /api/freelancer/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

41. Blog System

Purpose: This project implements Blog System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Blog
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/blog
2. GET /api/blog
3. PUT /api/blog/:id
4. DELETE /api/blog/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

42. CMS System

Purpose: This project implements CMS System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. CMS
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/cms
2. GET /api/cms
3. PUT /api/cms/:id
4. DELETE /api/cms/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

43. News Publishing System

Purpose: This project implements News Publishing System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 News Publishing
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/news
2 GET /api/news
3 PUT /api/news/:id
4 DELETE /api/news/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

44. Forum System

Purpose: This project implements Forum System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Forum
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/forum
2 GET /api/forum
3 PUT /api/forum/:id
4 DELETE /api/forum/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

45. Community Management System

Purpose: This project implements Community Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Community

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/community

2 GET /api/community

3 PUT /api/community/:id

4 DELETE /api/community/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

46. Real Estate System

Purpose: This project implements Real Estate System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Real Estate

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/real

2 GET /api/real

3 PUT /api/real/:id

4 DELETE /api/real/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

47. Property Listing System

Purpose: This project implements Property Listing System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Property Listing

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/property

2 GET /api/property

3 PUT /api/property/:id

4 DELETE /api/property/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

48. Tenant Management System

Purpose: This project implements Tenant Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Tenant

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/tenant

2 GET /api/tenant

3 PUT /api/tenant/:id

4 DELETE /api/tenant/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

49. Society Management System

Purpose: This project implements Society Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Society

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/society

2 GET /api/society

3 PUT /api/society/:id

4 DELETE /api/society/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

50. Maintenance Request System

Purpose: This project implements Maintenance Request System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Maintenance Request

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/maintenance

2 GET /api/maintenance

3 PUT /api/maintenance/:id

4 DELETE /api/maintenance/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

51. Help Desk System

Purpose: This project implements Help Desk System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Help Desk
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/help
2. GET /api/help
3. PUT /api/help/:id
4. DELETE /api/help/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

52. IT Asset Management

Purpose: This project implements IT Asset Management with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. IT Asset
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/it
2. GET /api/it
3. PUT /api/it/:id
4. DELETE /api/it/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

53. Subscription Management System

Purpose: This project implements Subscription Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Subscription
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/subscription
2. GET /api/subscription
3. PUT /api/subscription/:id
4. DELETE /api/subscription/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

54. Online Quiz System

Purpose: This project implements Online Quiz System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Online Quiz
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/online
2 GET /api/online
3 PUT /api/online/:id
4 DELETE /api/online/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

55. Order Tracking System

Purpose: This project implements Order Tracking System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Order Tracking
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/order
2. GET /api/order
3. PUT /api/order/:id
4. DELETE /api/order/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

56. Warehouse Management System

Purpose: This project implements Warehouse Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Warehouse
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/warehouse
2. GET /api/warehouse
3. PUT /api/warehouse/:id
4. DELETE /api/warehouse/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

57. Purchase Order Management System

Purpose: This project implements Purchase Order Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Purchase Order
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/purchase
2. GET /api/purchase
3. PUT /api/purchase/:id
4. DELETE /api/purchase/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

58. Delivery Management System

Purpose: This project implements Delivery Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Delivery
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/delivery
2. GET /api/delivery
3. PUT /api/delivery/:id
4. DELETE /api/delivery/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

59. Supplier Management System

Purpose: This project implements Supplier Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Supplier
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/supplier
2. GET /api/supplier
3. PUT /api/supplier/:id
4. DELETE /api/supplier/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

60. Personal Finance Manager

Purpose: This project implements Personal Finance Manager with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)
2 Personal Finance Manager
3 Reports
4 Settings

Sample REST APIs (CRUD):

1 POST /api/personal
2 GET /api/personal
3 PUT /api/personal/:id
4 DELETE /api/personal/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

61. Tour Package Management System

Purpose: This project implements Tour Package Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Tour Package
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/tour
2. GET /api/tour
3. PUT /api/tour/:id
4. DELETE /api/tour/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

62. Room Reservation System

Purpose: This project implements Room Reservation System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Room Reservation
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/room
2. GET /api/room
3. PUT /api/room/:id
4. DELETE /api/room/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

63. Bike Rental System

Purpose: This project implements Bike Rental System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1 Users (Admin/User)

2 Bike Rental

3 Reports

4 Settings

Sample REST APIs (CRUD):

1 POST /api/bike

2 GET /api/bike

3 PUT /api/bike/:id

4 DELETE /api/bike/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

64. Recruitment Management System

Purpose: This project implements Recruitment Management System with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. Recruitment
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/recruitment
2. GET /api/recruitment
3. PUT /api/recruitment/:id
4. DELETE /api/recruitment/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB
2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized

65. E-Learning Platform

Purpose: This project implements E-Learning Platform with full CRUD. Admin manages master data. Users perform operations. JWT secures all routes and role middleware restricts admin-only operations.

Main Modules / Collections:

1. Users (Admin/User)
2. E-Learning
3. Reports
4. Settings

Sample REST APIs (CRUD):

1. POST /api/e-learning
2. GET /api/e-learning
3. PUT /api/e-learning/:id
4. DELETE /api/e-learning/:id

JWT Authentication Flow:
1. User registers → password hashed → saved in DB

2. User logs in → server generates JWT token
3. Token stored in frontend (localStorage/cookie)
4. Token sent in Authorization header for every request
5. Backend middleware verifies token and role
6. If valid → allow access, else → 401 Unauthorized