# Backend Developer Intern Assignment: Secure RESTful API

Node.js, Express, MongoDB Atlas, RBAC

October 2025 Submission

## Assignment Overview

This project delivers a secure, scalable RESTful API built on `Node.js`, `Express`, and `MongoDB Atlas`, implementing robust user authentication via JWT and comprehensive **Role-Based Access Control (RBAC)** for managing a **Notes** entity.

| | |
|---|---|
| **us: Backend** | Secure JWT authentication, Role-based access (`USER` vs `ADMIN`), and CRUD APIs for Notes. |
| **ic Frontend** | A simple UI (planned/to be integrated) for registration, login, and CRUD demonstration. |

## 1 Core Features Implemented

- ✓ User authentication (`register`, `login`, `logout`, `refresh-token`) using HTTP-only cookies.
- ✓ Role-Based Access Control (`USER` vs `ADMIN`) enforced via custom middleware.
- ✓ CRUD APIs for the **Notes** entity.
- ✓ Secure JWT token handling (separate Access and Refresh tokens).
- ✓ Input sanitization and validation (Mongoose, `isValidObjectId`, trim/case conversion).
- ✓ Centralized error handling and API response standardization.

## 2 Setup and Installation

The database is hosted on **MongoDB Atlas** for persistent access, meeting the scalability and live deployment requirements.

### 2.1 Prerequisites

- Node.js (v18+)
- MongoDB Atlas Account

### 2.2 Installation Guide

1. **Clone the repository:** `git clone [YOUR_REPO_URL]`
   `cd Backend`
2. **Install dependencies:** `npm install`
3. **Configure Environment:** Create a file named `.env` in the root directory.

### 2.3 Environment Variables (`.env`)

The following variables must be configured:

```
PORT=8000
FRONTEND_URL=http://localhost:3000
MONGODB_URI="mongodb+srv://user:password@cluster0.abcde.mongodb.net/notedb"
ACCESS_TOKEN_SECRET=your_access_token_secret
REFRESH_TOKEN_SECRET=your_refresh_token_secret
ACCESS_TOKEN_EXPIRY=1d
REFRESH_TOKEN_EXPIRY=10d
```

### 2.4 Running the Server

Start the development server:

```
npm run dev
```

The server will be available at `http://localhost:8000/api/v1`.

## 3 Security and Authorization Model

The system enforces strict access control using a middleware chain on all protected routes.

### 3.1 Middleware Chain

1. **`verifyJWT`**: Authenticates the user based on the `accessToken` cookie, retrieves the full user object (including `role`), and attaches it to `req.user`.
2. **`isAdmin`**: Runs after `verifyJWT`. Checks if `req.user.role` is exactly `'ADMIN'`. If not, throws `403 Forbidden`.
3. **`isNoteOwnerOrAdmin`**: Runs after `verifyJWT`. Checks if the authenticated user is the `owner` of the note specified in `req.params.noteId` OR if their role is `'ADMIN'`. If neither, throws `403 Forbidden`.

### 3.2 Database Schema Design (`User` & `Note`)

- **User Model:** Includes fields for `username`, `email`, `fullName`, `password` (hashed using Mongoose pre-save hook), and `refreshToken`. The `role` field is defined as an `enum: ['USER', 'ADMIN']` with a default of `'USER'`.
- **Note Model:** Includes `title`, `content`, and an `owner` field referencing the `User` model.

## 4 API Documentation

`Base URL: http://localhost:8000/api/v1`

### 4.1 User & Authentication Routes (`/users`)

Table 1: User Authentication Endpoints

| Method | Endpoint | Description | Auth | Body (Fields) |
|---|---|---|---|---|
| `POST` | `/register` | Create new user. | None | `username, email, fullName, password` |
| `POST` | `/login` | Logs user in. Sets JWT cookies. | None | `email, password` |
| `POST` | `/refresh-token` | Renews `accessToken` via `refreshToken`. | None | None (reads cookie) |
| `POST` | `/logout` | Clears token cookies. | ✓ | None |
| `GET` | `/current-user` | Get logged-in user profile. | ✓ | None |

Continued on next page

Table 1 – continued from previous page

| Method | Endpoint | Description | Auth | Body (Fields) |
|---|---|---|---|---|
| PATCH | /update-details | Update profile details. | ✓ | `username?`, `email?`, `fullName?` |
| PATCH | /change-password | Update user password. | ✓ | `oldPassword`, `newPassword` |

### 4.2 Notes Entity CRUD Routes (`/notes`)

Table 2: Notes CRUD Endpoints

| Method | Endpoint | Description | Auth/Role | Body (Fields) |
|---|---|---|---|---|
| POST | / | Create a new note. | ✓ | `title, content` |
| GET | / | Get all notes owned by user. | ✓ | None |
| GET | /:noteId | Get single note. | ✓ | None |
| PATCH | /:noteId | Update note. | ✓ (Owner/Admin) | `title?`, `content?` |
| DELETE | /:noteId | Delete note. | ✓ (Owner/Admin) | None |

### 4.3 Admin Routes (`/admin`)

Table 3: Admin-Exclusive Endpoints

| Method | Endpoint | Description | Auth/Role | Caution |
|---|---|---|---|---|
| PATCH | /role/:userId | Update user role. | ✓ (ADMIN) | Cannot change self. |
| DELETE | /account/:userId | Delete user and cascade delete all associated notes. | ✓ (ADMIN) | Cannot delete self. |
| DELETE | /notes/all | Deletes ALL notes globally. | ✓ (ADMIN) | Extreme caution advised. |

## 5 Scalability and Deployment Readiness

The project design provides strong foundational components for scalability:

- **Database:** Utilizing **MongoDB Atlas** inherently provides a cloud-hosted, highly available, and easily sharded database infrastructure, ready for massive scaling.
- **Modular Design:** The separation into dedicated routers (`user`, `note`, `admin`) and controllers is the first step toward a **microservices architecture**, allowing independent scaling of core functionalities.

- **Resource Optimization:** The use of short-lived Access Tokens minimizes the need for frequent database lookups, reducing load and improving API response times.
- **Asynchronous Handlers:** The `asyncHandler` wrapper ensures non-blocking I/O for all controller logic, maximizing Node.js's ability to handle concurrent requests efficiently.