# Backend Developer Assignment

## Objective

Develop a comprehensive RESTful API for a task management system using Node.js. The API should include user authentication, role-based access control, task management, and integration with a third-party notification service. Document the API using the OpenAPI Specification (OAS) to ensure clarity and ease of understanding for other developers.

## Core Requirements

### 1. User Registration

- **Endpoint**: Allow users to sign up by providing a username, email, and password.
- **Validation**:
  - Ensure a valid email format.
  - Enforce strong password criteria.
- **Optional**: Send a confirmation email upon successful registration.

### 2. User Login

- **Endpoint**: Allow registered users to log in using their credentials (username/email and password).
- **Functionality**:
  - Validate user credentials.
  - Issue a JWT token upon successful login.
- **Security**: Implement rate limiting to prevent brute-force attacks.

### 3. User Logout

- **Endpoint**: Log out the authenticated user.
- **Functionality**: Invalidate the JWT token to ensure the user is logged out securely.

### 4. Get User Profile

- **Endpoint**: Retrieve the profile information of the authenticated user.
- **Fields**: Include fields such as username, email, roles, and any other relevant user information.
- **Security**: Ensure the endpoint is protected and accessible only to authenticated users.

## 5. Role-Based Access Control (RBAC)

- **Implementation**: Define different roles with varying access levels to endpoints:
  - **Admin**: Full access to all endpoints, including user management and task assignment.
  - **Manager**: Access to manage tasks and view user profiles within their team.
  - **User**: Access to manage their own tasks and view their own profile.
- **Security**: Enforce role-based restrictions at the endpoint level.

## 6. Task Management

- **CRUD Operations**:
  - **Create Task**: Endpoint to create a new task with fields such as title, description, due date, priority, and status.
  - **Read Task**: Endpoint to retrieve a list of tasks, with optional filtering and sorting parameters.
  - **Update Task**: Endpoint to update task details.
  - **Delete Task**: Endpoint to delete a task.
- **Security**: Ensure tasks are associated with users and enforce access control.

## 7. Task Assignment

- **Functionality**:
  - Assign tasks to users.
  - Allow managers to assign tasks to users within their team.
- **Endpoints**:
  - View assigned tasks.
  - Update task assignments.

# Advanced Features (Bonus Points)

## 1. Real-Time Updates

- **Implementation**: Use WebSockets (e.g., Socket.io) to implement real-time updates for task changes.
- **Functionality**: Notify users of task changes in real-time.

## 2. Analytics

- **Endpoints**: Provide basic analytics endpoints to track the number of tasks completed, pending, and overdue.
- **Functionality**: Retrieve task completion statistics by user and team.

## 3. Caching

- **Implementation**: Use Redis to implement caching for frequently accessed endpoints.
- **Consistency**: Ensure cache invalidation strategies are in place for data consistency.

## 4. Rate Limiting

- **Implementation**: Apply rate limiting to protect the API from abuse.
- **Configuration**: Configure rate limits based on user roles and endpoint sensitivity.

## 5. Search and Filtering

- **Functionality**: Implement search and filtering for tasks based on various criteria (e.g., status, priority, due date).
- **Performance**: Ensure efficient querying and indexing for optimal performance.

# Additional Notes

## Libraries and Middleware

- **Flexibility**: Feel free to use any additional libraries or middleware that you find suitable for the implementation.
- **Best Practices**: Ensure the chosen libraries are well-maintained and commonly used in the industry.

## Scalability and Performance

- **Design**: Consider scalability and performance aspects in your design decisions.
- **Optimization**: Implement strategies to handle a large number of concurrent requests efficiently.

# Submission Guidelines

## 1. Repository and Implementation

- **Fork**: Fork the provided repository to your own GitHub account.
- **Development**: Implement the solution in your forked repository, ensuring all core requirements are met.
- **Commits**: Use meaningful commit messages and maintain a clean commit history.
- **Code Quality**: Ensure the code is well-structured and modular for maintainability.

## 2. Documentation

- **Setup**: Provide detailed documentation on how to set up and run the application locally.
- **Instructions**: Include instructions for installing dependencies, configuring environment variables, and starting the server.
- **API Overview**: Provide an overview of the API endpoints, including their purpose and usage examples.
- **Assumptions**: Document any assumptions or design decisions made during the implementation.

## 3. Functional Requirements

- **Compliance**: Ensure the API meets all functional requirements outlined in the assignment.
- **Error Handling**: Implement proper error handling and validation for all endpoints.
- **RBAC**: Verify that role-based access control is correctly enforced for each endpoint.

## 4. API Documentation

- **OpenAPI**: Document the API using OpenAPI Specification (OAS) version 3.0.
- **Interactive Docs**: Provide a link to the Swagger UI or ReDoc page where the API documentation can be interactively explored.
- **Details**: Ensure the documentation includes details such as endpoint paths, request methods, request and response schemas, query parameters, and authentication requirements.

## 5. Deployment (Optional but Recommended)

- **Deployment**: Deploy the API to a cloud provider (e.g., Heroku, AWS, GCP) for demonstration purposes.
- **URL**: Provide the URL for the deployed API in the documentation.
- **Security**: Ensure the deployment is secure and accessible.