Backend Software Engineering Coding Exercise

Exercise Overview

In this exercise, you will design and implement a simple invoicing system for a company. The system will manage companies, users, and invoices. You will implement a RESTful API that allows companies to send and retrieve invoices. The API will also require authorization for invoice submission.

Requirements

1. **Entities**:
    - **Company**:
        - `id`: unique identifier (string)
        - `name`: name of the company (string)
        - `users`: list of users (each user has a unique identifier. All have the same permissions)
    - **Invoice**:
        - `invoice_id`: unique identifier (alphanumeric string)
        - `date_issued`: date when the invoice was issued (ISO 8601 format)
        - `net_amount`: net amount of the invoice (float)
        - `vat_amount`: VAT amount (float)
        - `total_amount`: total amount (float)
        - `description`: brief description of the invoice (string)
        - `company_id`: the ID of the company that issued the invoice,
        - `counter_party_company_id`: the ID of the company that receives the invoice
2. **API Endpoints**:
    - `POST /invoice`: Create a new invoice. Requires authorization.
    - `GET /invoice/sent`: Retrieve a list of invoices sent by the authenticated company. Supports filtering by `counter_party_company`, `date_issued`, and `invoice_id`.
    - `GET /invoice/received`: Retrieve a list of invoices received by the authenticated company. Supports filtering by `counter_party_company`, `date_issued`, and `invoice_id`.

3. **Authorization**:
    - Implement a basic token-based authentication mechanism. For the purposes of this exercise, you can use hardcoded tokens for demonstration.

Implementation Guidelines

- Choose the .NET framework for your implementation (e.g., ASP.NET Core).
- Use any data structure you are comfortable with to store companies and invoices.
- Write unit tests for your API endpoints to ensure they work as expected.
- Use best practices for API design, including proper HTTP status codes and error handling.
- An additional bonus will be awarded for implementing a deployment solution using Docker.

Evaluation Criteria

1. **Functionality**: Does the implementation meet the requirements outlined above?
2. **Code Quality**: Is the code clean, well-structured, and easy to understand?
3. **Testing**: Are there adequate tests for the implemented API endpoints?
4. **Error Handling**: Are errors handled gracefully, with appropriate HTTP status codes?
5. **Documentation**: Is the code well-documented, with comments explaining key parts?

Submission

Please submit your code in a Git repository or in a zip file, along with a README file that includes instructions on how to run the application and any relevant details about your implementation.