Backend Developer Test (User Management System using .NET 8)

Objective

The objective of this test is to assess your skills in developing a backend solution using **.NET 8**, focused on building a **User Management System**. You will implement a secure RESTful API that supports user registration, authentication, and user management, with JWT-based access control, webhook notifications, and logging for visibility and debugging.

Use Case: User Management System

You are building a backend service that allows clients to manage user accounts. The API should support:

- User registration and login
- Protected CRUD operations on user records
- Webhook integration when users log in
- Application-level logging for key operations and errors

Requirements

1. Storage Solution

- Use a relational database of your choice.
- Use any preferred ORM (e.g., Entity Framework Core, Dapper, NHibernate, etc.).

- Store user data with the following fields:

    - Id (GUID or integer)
    - Username
    - Email
    - Password

    - FirstName
    - LastName
    - CreatedAt
    - LastLoginAt (timestamp)

2. .NET 8 Project

- Use **ASP.NET Core 8**.
- Follow proper architecture: layered structure, dependency injection, clean code principles.

3. RESTful API Endpoints

Authentication

- POST /auth/register — Register a new user
- POST /auth/login — Authenticate a user and return a JWT
    - On successful login:
        - Update LastLoginAt
        - Trigger webhook
        - Log the login event

User Management (JWT Protected)

- GET /users — List all users
- GET /users/{id} — Get user by ID
- PUT /users/{id} — Update user
- DELETE /users/{id} — Delete user

---

4. JWT Authentication

- Secure all /users endpoints using JWT.

- Token must:
    - Include expiration
    - Be signed with a secure key

- Passwords should be treated the same as if this was a production environment

---

5. Webhook Integration

Webhook Behavior

- On each successful login:
    - Send a **POST** request to a configured webhook URL.
    - Include a list of all users who have logged in within the last 30 minutes.

Example Payload

```
{
  "event": "user_logged_in",
  "timestamp": "2025-05-29T14:12:00Z",
  "activeUsers": [
    {
      "id": "1",
      "username": "jdoe",
      "email": "jdoe@example.com",
      "lastLoginAt": "2025-05-29T14:12:00Z"
    }
  ]
}
```

Failures in webhook delivery should be logged, but retries are optional.

---

6. Logging Solution

Implement a logging solution that:

- Use any logging solution
- Logs key application events:
    - User registration
      User login (success and failure)
    - Webhook dispatch results

    - Critical errors and exceptions

- Outputs logs to:

    - **Console** by default
    - Optionally a file or external service (e.g., Serilog, Seq, or structured logging sink)

Logging Guidelines

- Use appropriate logging levels (Information, Warning, Error, etc.)
- Format logs clearly for observability

---

7. API Data Contracts Documentation

- Provide a document or README section with:

    - Request/response schemas
    - Required fields and validation rules
    - Example payloads

Guidelines

- Clean, modular architecture (e.g., services, repositories, models, DTOs)
- Validation and error handling using standard HTTP status codes
- Logging and configuration should be production-ready
- Git history should be logical and consistent

Submission

- Submit a Git repo (GitHub, GitLab, etc.) with:
    - Source code
    - README.md containing:
        - Setup and run instructions

- API overview and authentication explanation
- Webhook behavior and configuration
- Logging overview
- Data contracts documentation