Coding Assessment: Designing a Task Management and Notification System

---

Overview

You are tasked with designing and implementing a task management system that tracks tasks, assigns them to users, and manages notifications for updates and deadlines. The system should emphasize clean architecture, scalability, and performance while providing a user-friendly interface for task tracking and notifications.

Objective

The goal is to evaluate your ability to design and implement a robust system for managing tasks and user assignments. Your solution should demonstrate strong design principles, efficient data handling, and maintainable code.

Note: This is an **open-book exam**, and you can use online resources. However, you are fully responsible for every line of code in your project. Be prepared to explain your design choices and code during the review.

Technology Stack:

- Preferred: **ASP.NET MVC** project.
- Alternative: **.NET backend** with any other frontend framework.

Project Requirements

1. **Task Management**
   - Design a system to manage tasks with the following attributes:
     - **Title**
     - **Description**
     - **Priority** (Low, Medium, High)
     - **Status** (To-Do, In Progress, Completed)
     - **Due Date**
     - **Assignee** (users assigned to the task, one task can be assigned to multiple users).
   - Tasks should be:
     - **Created**: Allow users to create new tasks.
     - **Updated**: Modify task attributes (e.g., status or priority).

        - **Deleted**: Remove tasks no longer needed.

2. **User Management**
    - Implement a mechanism for managing users with the following attributes:
        - **Name**
        - **Email**
        - **Role** (e.g., Admin, Manager, User).
    - Each user can have multiple tasks assigned to them.
3. **Notifications**
    - Send notifications to users for:
        - Task updates (e.g., status change).
        - Upcoming deadlines (e.g., tasks due within 24 hours).
    - Notifications should include details like:
        - Task Title
        - Updated Field (if applicable)
        - Due Date (if applicable).
4. **Reporting and Filtering**
    - Provide functionality to filter tasks by:
        - Assignee
        - Priority
        - Status
    - Implement reporting features such as:
        - Total tasks by status (e.g., number of tasks in progress or completed).
        - Overdue tasks (tasks past their due date).
5. **Design Patterns and Architecture**
    - Use appropriate design patterns (e.g., **Repository** for data access, **Observer** for notifications).
    - Structure your project with a **layered architecture**.

6. **Performance and Error Handling**
    - Optimize the system for querying tasks and users, especially when filtering by attributes like status or priority.
    - Consider optimizations such as:
        - Indexing strategies
        - Efficient data retrieval methods.
    - Ensure proper error handling for invalid operations (e.g., assigning a task to a non-existent user).
7. **Extended Scenarios (Optional)**
    - Discuss or implement real-time updates for tasks using WebSockets or SignalR.
    - Describe or implement a recurring task mechanism (e.g., weekly or monthly recurring tasks).

Task Breakdown

Task 1: Database/Entity Design

- Outline the database structure, focusing on relationships between:
    - Tasks
    - Users
    - Notifications.
- Ensure efficient handling of task assignments and notification delivery.
- Prepare a **database/entity relationship diagram** and include it in a Git repository.

Task 2: Architecture & Core Implementation

- Implement the core system architecture, focusing on:
    - Task creation, updating, and assignment.
    - Notification generation and delivery.
- Showcase your understanding of:
    - Dependency injection
    - Abstraction
    - Separation of concerns.

- Push the completed implementation to the Git repository.

Task 3: Code Walkthrough & Final Discussion

Location: Brain Station 23, Mirpur Office.

- Conduct a walkthrough of your code, explaining:
  - Your design decisions.
  - Handling of edge cases (e.g., overdue tasks, notification failures).
  - Justification for chosen patterns and architecture.
- Discuss potential extensions, such as:
  - Real-time updates or integrations with external systems (e.g., calendar tools).