**To-Do List Test**

**Part 1: API Development Objective:**

Develop a simple **To-Do List API** using **.NET Core** .

The API should support CRUD operations with authentication and modular architecture.

**Requirements:**

• Implement a **To-Do List Module** within a .Net Core application.

• Define an **entity** (TodoItem) with the following properties:

public class TodoItem : Done

{

public string Title { get; set; }

public string Description { get; set; }

public bool IsCompleted { get; set; }

}

• Implement the **Application Service** (ITodoAppService) with the following endpoints:

* GET /api/todo → Retrieve all to-do items
* GET /api/todo/{id} → Retrieve a to-do item by ID
* POST /api/todo → Create a new to-do item
* PUT /api/todo/{id} → Update an existing to-do item
* DELETE /api/todo/{id} → Delete a to-do item

• Use **Entity Framework Core (Code First Approach)** with **SQL Server** as the

database.

• Implement **Repository Pattern** using .Net Framework.

• Add **basic validation** using FluentValidation.

**Part 2: Performance Optimization & Best Practices Objective:**

Optimize the API for **performance, maintainability, and scalability**.

**Tasks:**

1. Use **Asynchronous Programming (async/await)** for database queries to improve responsiveness.

2. Implement **Global Exception Handling** for cleaner error management.

3. Apply **Dependency Injection** for cleaner and more modular service

implementation.

**Bonus (Optional, but Recommended)**

• **Implement Authentication** using .Net Core’s built-in Identity System (JWT-based

authentication).

• **Add Swagger Documentation** for API endpoints.

**Evaluation Criteria:**

Clean and structured code following best practices. Proper use of .Net Core **Framework** features (modularization, dependency injection, validation, etc.). Efficient database access using **Entity Framework Core (Code First)**.

Performance optimization (async/await). Clear commit history and well-documented repository.