Task API Capitole

Descripción

Task API Capitole es una aplicación Blazor y una API RESTful para gestionar tareas. Permite crear, leer, actualizar y eliminar tareas. Tambien permite filtrarlas o añadirlas a favoritos.

Características

- Crear una nueva tarea.
- Obtener todas las tareas.
- Obtener una tarea específica por ID.
- Actualizar una tarea existente.
- Eliminar una tarea.

Tecnologías

- .NET 8
- ASP.NET Core
- Entity Framework Core
- MongoDB
- xUnit (para pruebas)
- Moq (para pruebas)

Configuración del Proyecto

Requisitos Previos

- .NET 8 SDK
- MongoDB

Instalación

1. Clona el repositorio:

   ```
   git clone https://github.com/tu-usuario/TaskAPICapitole.git
   ```

2. Ve al directorio del proyecto:

   ```
   cd TaskAPICapitole
   ```

3. Restaura los paquetes NuGet:

   ```
   dotnet restore
   ```

Configuración de la Base de Datos

Asegúrate de tener MongoDB en ejecución y configura la cadena de conexión en `appsettings.json`:

```
{
  "MongoDbSettings": {
    "ConnectionString": "mongodb://localhost:27017",
    "DatabaseName": "CapitoleTestDb"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Estructura Hexagonal en Task API Capitole

Introducción

La estructura hexagonal, también conocida como arquitectura de puertos y adaptadores, es un patrón de diseño de software que promueve la separación de preocupaciones y la flexibilidad en el diseño de aplicaciones. En esta arquitectura, el núcleo de la aplicación (la lógica de negocio) está aislado de los detalles de implementación, como la base de datos, la interfaz de usuario o las API externas.

Beneficios de la Estructura Hexagonal

- **Separación de Preocupaciones**: La lógica de negocio se mantiene independiente de los detalles de infraestructura, lo que facilita el mantenimiento y la evolución de la aplicación.
- **Flexibilidad**: Permite cambiar fácilmente componentes externos (por ejemplo, cambiar la base de datos de MongoDB a SQL Server) sin afectar la lógica de negocio.
- **Pruebas**: Facilita las pruebas unitarias y funcionales, ya que los componentes de la lógica de negocio pueden ser probados de forma aislada.
- **Escalabilidad**: Mejora la capacidad de la aplicación para escalar, ya que los componentes pueden ser desarrollados y desplegados de manera independiente.

Estructura del Proyecto

La estructura del proyecto Task API Capitole se organiza de la siguiente manera:

1. Domain (Dominio)

Este es el núcleo de la aplicación. Contiene las entidades de negocio y las interfaces que definen el contrato que debe seguir la lógica de negocio. Aquí no hay dependencias hacia otros proyectos o bibliotecas externas.

- **Entities**: Contiene las entidades de dominio, como `TaskItem`.
- **Interfaces**: Define las interfaces de los servicios, como `ITaskService`.

2. Application (Aplicación)

Este proyecto contiene la lógica de negocio y las implementaciones de los casos de uso. Aquí se implementan los servicios definidos en el dominio.

- **Funcionalities**: Contiene la implementación de los servicios, como `TaskService`.

3. Infrastructure (Infraestructura)

Este proyecto contiene las implementaciones de los detalles técnicos, como los repositorios y las configuraciones de acceso a datos.

- **Repositories**: Implementa los repositorios para el acceso a la base de datos, como `TaskItemRepository`.
- **Settings**: Contiene configuraciones específicas de la infraestructura, como `MongoDbSettings`.

4. TaskAPICapitole (Capa de Entrada/Salida)

Este proyecto contiene los controladores de la API que manejan las solicitudes HTTP y las envían a los servicios de la aplicación.

- **Controllers**: Define los controladores de la API, como `TaskItemsController`.

5. TaskApiTests (Pruebas)

Este proyecto contiene las pruebas unitarias y funcionales para los servicios y los controladores.

- **Services**: Contiene las pruebas unitarias para los servicios, como `TaskServiceTests`.
- **Controllers**: Contiene las pruebas funcionales para los controladores, como `TaskItemsControllerTests`.

Implementación en la Aplicación

Controlador

El controlador maneja las solicitudes HTTP y utiliza los servicios de la aplicación para procesar estas solicitudes. Ejemplo de `TaskItemsController`:

```
[Route("api/[controller]")]
[ApiController]
public class TaskController : ControllerBase
{

    private readonly ITaskService _taskService;

    public TaskController(ITaskService taskService)
    {
        _taskService = taskService;
    }

    [HttpGet]
    public async Task<ActionResult<List<TaskItem>>> Get()
    {
        List<TaskItem> tasks = await _taskService.GetAllTasks();
        if (tasks == null)
        {
            return NotFound();
        }
        return Ok(tasks);
    }

    [HttpPost]
    public async Task<ActionResult> Post([FromBody] TaskItem task)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        await _taskService.AddTask(task);
        return CreatedAtAction(nameof(Get), new { id = task.Id }, task);
    }

    [HttpPut("{id}")]
    public async Task<ActionResult> Put(string id, [FromBody] TaskItem task)
    {
        if (id != task.Id)
        {
            return BadRequest();
        }
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        await _taskService.UpdateTask(task);
        return NoContent();
    }

    [HttpDelete("{id}")]
    public async Task<ActionResult> Delete(string id)
    {
        await _taskService.DeleteTask(id);
        return NoContent();
    }
}
```