Práctica de Desarrollo de Software
Actividad Evaluativa #4 – 20%

Construcción de Endpoints CRUD para la entidad Movie

1. **Objetivo de la práctica:**

   El objetivo de esta actividad es que el estudiante construya completamente el flujo backend para una nueva entidad llamada **Movie**, replicando la arquitectura, los patrones y estilo implementado previamente para la entidad **Category**.

   Al finalizar, el estudiante debe haber implementado correctamente:

   - Modelo de Movie.
   - Repositorio y contrato correspondiente.
   - Servicio con sus métodos asincrónicos y contrato correspondiente.
   - Controlador con endpoints CRUD.
   - Validaciones, manejo de errores y responses coherentes.

1.5 Prerrequisito

Para poder desarrollar esta práctica, el estudiante **debe estar al día con todo el flujo de la entidad Category**, incluyendo su modelo, servicio, repositorio, DTOs y endpoints.

2. **Descripción General**

   En clase se construyó la entidad **Category** y sus endpoints:

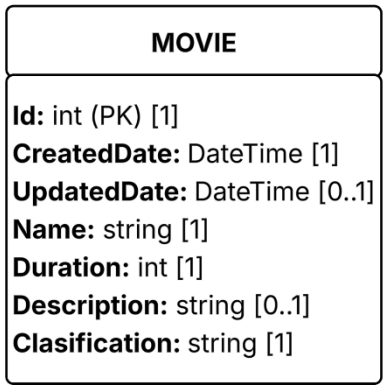
   - GetCategoriesAsync
   - GetCategoryAsync
   - CreateCategoryAsync
   - DeleteCategoryAsync
   - UpdateCategoryAsync

   Siguiendo este mismo patrón, el estudiante deberá crear *desde cero* la entidad **Movie**.

---

3. **Requerimientos de la Entidad Movie:**

   La entidad debe contener al menos los siguientes campos con sus respectivos tipos de datos:



[1] = Not Nulleable

[0..1] = Nulleable

Puedes agregar campos adicionales si lo consideras necesario.

---

4. Actividad Paso a Paso

4.1 Crear la entidad Movie (capa DAL/Models)

Implementa la clase con sus propiedades. Luego creas la migración para crear la nueva tabla en la BD.

4.2 Crear la interfaz del repositorio

Crea la interfaz IMovieRepository que contenga:

- GetMoviesAsync();
- GetMovieAsync(int id);
- CreateMovieAsync(Movie movie);
- UpdateMovieAsync(Movie movie);
- DeleteMovieAsync(int id);

4.3 Implementar el repositorio

En la carpeta correspondiente, implementa las operaciones usando el contexto de base de datos.

4.4 Crear servicio MovieService

Crea la clase de servicio que encapsule la lógica del dominio. Debe exponer los mismos métodos en versión asincrónica que el repositorio.

4.5 Crear los DTOs necesarios (DAL/Dtos/Movie)

Deben existir al menos 2 Dtos, créalos con base a las necesidades de la API.

4.6 Crear el controlador MovieController

Implementa los siguientes endpoints REST:

- GET /api/movies
- GET /api/movies/{id}
- POST /api/movies
- PUT /api/movies/{id}
- DELETE /api/movies/{id}

4.7 Validaciones mínimas

- El “Name” no puede ser vacío ni puede contener más de 100 caracteres.
- El “Duration” no puede ser vacío
- El “Clasification” no puede ser vacío ni puede contener más de 10 caracteres.
- Si se intenta actualizar o eliminar una película inexistente, retornar 404.

---

5. Porcentajes de Calificación

Calificación total: 5.0 equivalente al 100%

Distribución general:

- **API Category hecha en las clases:** 40%
- **API Movie:** 60%

Distribución del 60% de la API Movie:

- **Creación de la entidad/modelo Movie en la BD con sus respectivos DTOs:** 50%
- **Construcción de los endpoints (GetMovieAsync, GetMoviesAsync, CreateMovieAsync, UpdateMovieAsync, DeleteMovieAsync):** 50%

Nota: Todo el código y todos los endpoints debe funcionar en su totalidad, hablo tanto de los 5 endpoints de category como los nuevos 5 endpoints que crearás de Movie. Por cada endpoint que **no funcione correctamente desde Swagger**, se rebajará **0.5 puntos**.

6. Entrega.

El estudiante deberá relacionar en el archivo de Excel la URL del repositorio y su UserName de GitHub.

---

7. Notas Finales

- Mantén el mismo estilo y patrones usados en Category.
- Usa programación asincrónica en todos los métodos.
- Los nombres de los métodos deben ser coherentes y consistentes.
- Si necesitas refactorizar algo para mantener consistencia, puedes hacerlo.