Sistemas de Información

Práctica 4: Diseño, desarrollo e instalación de un sistema de información web con persistencia de datos

Jose Secadura Del Olmo

Chloé Bolle-Reddat

Sergio Saura Oliva

Facultad de Ingeniería y Arquitectura. Campus de Zaragoza.

2025

Índice

Índice 2
Introducción 3
Diseños realizados 4
Diseño frontend 4
Diseño backend (API) 5
Realización de pruebas 7
Pruebas del sistema 7
Limitaciones y trabajo futuro en las pruebas 8
Metodologías 9
Recursos y herramientas 9
Distribución del trabajo y horas 10
Dificultades encontradas 11
Bibliografía 12

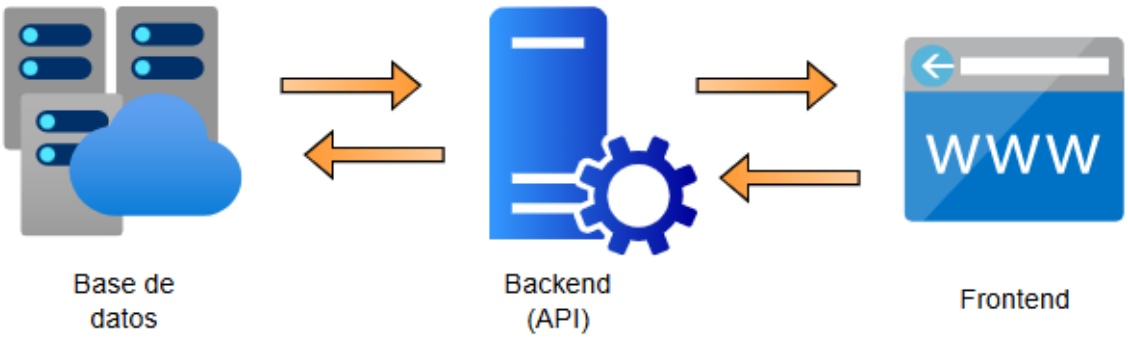
Introducción

En esta práctica se completo la mayoría del funcionamiento de la aplicación web de películas llamada Fylt. Se consiguió un sistema con persistencia de datos, se pudo integrar la capa vista, la capa de persistencia y obtener los datos de la API TMDB y poder comunicar la base de datos ya integrada con estas capas.

Cada capa está separada en repositorio de GitHub diferente, es decir, la implementación de la capa web se encuentra en un repositorio diferente al de la capa de persistencia

Para el diseño de la capa web se ha utilizado las herramientas de React/Typescript, para el desarrollo de la capa de persistencia se han utilizado las librerías y el entorno de .NET y la base de datos como anteriormente se comentó se utiliza PostgreSQL que se encuentra en la nube.

El diseño de la aplicación sería el siguiente:



Diseños realizados

Diseño frontend

Para el diseño de la capa web se ha utilizado el lenguaje de React con Typescript, se decidió este lenguaje debido a que algunos de los integrantes del equipo habían desarrollado anteriormente alguna aplicación con estas herramientas.

Se han utilizado librerías y herramientas extras para la realización del diseño. Se ha utilizado Node.js para algunos funcionamientos básicos del sistema y los diseños de Tailwind CSS que han servido para ajustar los diseños y los colores de la aplicación.

La estructura que se implementó en la aplicación es la siguiente:

- **public**: Carpeta para activos estáticos (imágenes, íconos, fuentes) que se recogen directamente y se pueden usar en cualquier clase.
- **src**: Código fuente principal de la aplicación que contiene los siguientes paquetes:
    - **app**: Rutas y layouts de Next.js. Aquí se encuentra el layout.tsx (layout raíz), page.tsx que es la página principal (home) sin que estuviera la sesión iniciada como predeterminado y subcarpetas para el resto de las ventanas:
        - community , movie, profile, search: Resto de ventanas del usuario estándar.
        - login, register: Ventanas de iniciar sesión o registro.
        - admin: Ventanas del administrador que se componen de las siguientes subcarpetas: api-key, encuestas, películas y un fichero page que indica cual es la principal de estas.
    - **components**: Componentes de React que se utilizan en diferentes partes de la aplicación. Incluye componentes más específicos (movie-card, actor-card, headers) y carpeta ui (user interface) que contiene los componentes principales más básicos y que más se suelen usan (botones, inputs, etc.).
    - **lib**: Lógica y utilidades compartidas. Contiene auth-context.tsx (autenticación), utils.ts (funciones auxiliares) y carpeta api/ con cliente HTTP y servicios para auth, movies, reviews, surveys y usuarios.
    - **hooks**: Hooks personalizado, en este caso solo se usa el use-toast.ts que sirve para la creación, actualización, cierre y eliminación de las notificaciones.
    - **styles**: Archivos CSS globales como los colores por ejemplo.
- **.next**: Carpeta generada por Next.js con build, caché y archivos compilados (no editar manualmente).
- **node_modules**: Dependencias del proyecto instaladas.
- **Archivos raíz**: next.config.ts para la configuración de Next.js y package.json para configurar las dependencias de React, Tailwind CSS, Next.js, de iconos y de algunos componentes.

Diseño backend (API)

Para el diseño de la capa de persistencia se ha utilizado el entorno de .NET, se decidió utilizar este ya que aporta muy buenas herramientas para la creación de la API y es un entorno bastante completo y con herramientas que facilitan trabajar con GitHub o con Azure.

Se utilizan algunas librerías externas como BCript para la encriptación de contraseñas, por ejemplo. También se utilizan muchas librerías internas del entorno.

La estructura que se implemento para la API se divide principalmente en tres capas:

- Application: Capa superior, aquí esta lo que se va a mostrar de la API.
- Domain: Capa intermedia, se encuentra toda la lógica del programa.
- Infrastructure: Capa inferior, implementaciones para persistencia de datos y adaptadores externos para la base de datos.

Con esto tenemos el entorno bastante ordenado y permite que cualquier error que ocurra se detecte más rápido. También hay otra carpeta esta que se llamada test que guarda el programa para realizar los test de la aplicación.

A continuación se va a explicar lo que contiene más específicamente cada capa:

Capa Application:

Como bien se ha dicho antes aquí se mostrará lo que se ve de la API, contiene la siguiente estructura:

- **Properties/ launchSettings.json**: Este fichero es el de configuración de arranque cuando se inicia la API, aquí se especifica el puerto local en el que se inicia o donde se va a desplegar.
- **ApiModels/ ApiResponseBase**: Fichero encargado de gestionar las respuestas de la API, es decir si las pruebas que se realizaron fueron exitosos, fallidas, no encontradas…

1. **Controllers**: Esta carpeta contiene los controladores (endpoints) de las diferentes entidades de datos que se utilizan en la aplicación. Se hacen las llamadas para las operaciones CRUD para la base de datos comprobando si la petición fue exitosa o no y se controlan los errores de los servicios y se realizan la operación que se solicite en la API.

- **appsetings.json**: Fichero JSON que guarde la cadena de conexión de la base de datos para conectarse a esta y la cadena raíz para realizar solicitudes a la API de TMDB.
- **Program.cs**: Crea el host, habilita los cors, inicia los servicios y arranca la aplicación.

Capa Domain:

En esta capa se definen los servicios, los mappers y los vo's. Se gestiona la obtención de los datos de la API TMDB y lo que va a hacer las funciones CRUD cuando se comunique con la base de datos.

Esta capa tiene la siguiente estructura:

- **Clients/ TMDBClient**: Este paquete contiene el servicio del cliente TMDB, aquí se obtiene la lista de películas que se solicita a la API, para ello anteriormente en el endpoint de películas se obtiene una cadena en la que se junta la raíz de la url junto al token de la API de TMDB y se hace un bucle para obtener ir obteniendo varias páginas que tiene TMDB. Cada página contiene 20 películas, como la base de datos que esta subida en la nube tiene un espacio limitado, se ha decidido implementar 500 películas por cada vez que se importe, también se comprueba si las que se están importando ya están en la base de datos.
- **Mappers**: Esta carpeta contiene todas las clases que mapean el contenido de cada tabla, es decir, aquí se encuentran los mapeaos tanto de lo que se envía a la tabla como de lo que se recoge.
- **Services**: Aquí se encuentran los servicios del sistema, se realizan las operaciones CRUD para las tablas de la base de datos.
- **Utils**: Carpeta para las utilidades, en este caso se encuentra un filtro de comentarios para el administrador que se encarga de filtrar reseñas de películas que contengan palabras que no están permitidas y que el administrador pueda revisarlas.
- **VOs**: Paquete donde se encuentran los VOs para las tablas.
- **ServiceExtension.cs**: Este fichero es el encargado de registrar los servicios para la API y también se encarga de registrar la base de datos obteniendo la cadena de conexión para la base de datos.

Capa Infrastructure:

Es la capa encargada exclusivamente de la persistencia y de los adaptadores hacia recursos externos y traducir las llamadas del dominio. Aquí se indica la estructura de la base de datos y de cada una de las tablas.

Contiene la siguiente estructura:

- **Context**: Es el contexto de la base de datos:
    - **Configuration**: Define la configuración de cada tabla de la base de datos, es decir, se indican las columnas, la clave principal, relaciones…
    - **FyItContext.cs**: Este fichero se encarga en definir el contexto de la base de datos para poder enviar o recoger datos de esta. Se encarga de definir el esquema que hay que mirar y las tablas del contexto.
- **DAOs**: Este paquete contiene los ficheros DAO de cada una de la tabla de la base de datos. Se define como es cada tabla de la base de datos.

Realización de pruebas

Durante la cuarta sesión se realizaron las actividades relacionadas con la verificación del correcto funcionamiento del sistema, su preparación para la implantación y la elaboración de la documentación técnica correspondiente. Estas tareas aseguran que el sistema desarrollado cumple con los requisitos establecidos, es estable y se encuentra preparado para su uso y mantenimiento futuros.

Pruebas del sistema

En esta fase se implementaron pruebas de integración utilizando NUnit, Moq, Entity Framework Core y el paquete Microsoft.AspNetCore.Mvc.Testing. El propósito de estas pruebas fue comprobar el funcionamiento correcto de la conexión a nuestra base de datos alojada en Neon.
Las pruebas desarrolladas verifican los siguientes aspectos:
La conectividad con la base de datos.
El correcto funcionamiento del endpoint (get) encargado de obtener el listado de películas.
La inicialización adecuada de la API en un entorno de pruebas controlado.

Prueba de conexión con la base de datos Neon

Esta prueba verifica que la cadena de conexión configurada permite establecer comunicación con la base de datos externa para poder conectarnos sin errores.
Resultado: correcto.
Tiempo de ejecución: 2.5 segundos.

Prueba del endpoint "/api/Peliculas"

Se comprueba que el controlador de películas responde con un código HTTP 200 OK al realizar una solicitud GET. Lo cual nos indica que realizando la llamada podremos obtener la película. No se han desarrollado más test de este tipo debido a que mediante /swagger se han probado todos los endpoints realizados para su correcto funcionamiento, pero si está bien revisarlos de esta forma.
Resultado: correcto.
Tiempo de ejecución: 4.0 segundos.

Resumen de resultados

- CanConnectToNeonDatabase: correcto (2.5 s).
- GetAllPeliculas_ReturnsOk: correcto (4.0 s).

Todas las pruebas han finalizado satisfactoriamente, lo que confirma que el sistema responde de manera adecuada en un entorno de pruebas y que los servicios esenciales se encuentran operativos.

Limitaciones y trabajo futuro en las pruebas

Debido al alcance de la práctica y a la estructura del proyecto en esta fase, el número de pruebas implementadas es reducido. Se han priorizado las pruebas esenciales, pero será necesario implementar nuevas en un futuro.

El sistema queda preparado para ampliar de forma significativa su cobertura de pruebas. Entre las mejoras previstas se encuentran:

- Incorporación de pruebas unitarias específicas para los servicios.
- Ampliación de pruebas de integración cubriendo otros módulos (listas, usuarios, rankings, comentarios, encuestas, etc.).
- Implementación de pruebas de extremo a extremo (E2E) utilizando herramientas como Playwright.
- Pruebas de carga y rendimiento, especialmente sobre los endpoints relacionados con la importación de datos desde TMDB ya que dependemos de una base de datos externa y su conexión.

Aunque en esta fase inicial el número de pruebas es limitado, ya se ha construido la infraestructura necesaria para facilitar la ampliación futura de todos los tipos de test.

3. Documentación generada

En esta sesión se elaboraron los documentos necesarios para garantizar el uso, mantenimiento y comprensión del sistema.

Manual de instalación

Incluye la información necesaria para desplegar el sistema:

- Requisitos previos.
- Instalación del backend (.NET 9).
- Configuración de la base de datos Neon.
- Gestión de claves API, especialmente las utilizadas para TMDB y YouTube.
- Actualmente, la app únicamente funciona en local, por lo que para lanzarla únicamente hay que lanzar el back y el front correctamente. Para que se sincronicen y se permita el CRUD ya se modificó el archivo Program.cs del back.

Metodologías

Recursos y herramientas

Los recursos que se han utilizado para la realización de la capa web (frontend) son las herramientas de React con Typescript usando Visual Studio Code y el codespace que ofrece GitHub. Se utilizaron librerías de TailwindCSS, el framework de Next.js y algunos componentes que se han encontrado en diferentes web o repositorios.

Para trabajar en local se utilizó el motor de Node.js que es un entorno en tiempo de ejecución multiplataforma para poder ejecutar el programa.

Los recursos que se utilizaron para realizar la API que interconecta la base de datos y el frontend (el backend), se utilizó el entorno de .NET con Microsoft Visual Studio utilizando diferentes librerías principalmente internas del lenguaje.

Para la conexión de PostgreSQL se instalaron los siguientes paquetes:

- Microsoft.EntityFrameworkCore.Design (9.0.10)
- Microsoft.EntityFrameworkCore.Relational (9.0.10)
- Npgsal (9.0.4)
- Npgsql.EntityFrameworkCore.PostgreSQL (9.0.4)
- NpgsqlJson.NET (9.0.4)

Anteriormente el código del frontend y del backend estaban unidos con el mismo repositorio que guarda toda la documentación de este proyecto. Para controlar mejor las versiones del proyecto y obtener una mayor calidad en el código, se decidió crear dos repositorios nuevos que son independientes donde uno guarda el proyecto de la API y el otro el proyecto del frontend.

Se usa un control de versiones de GitHub donde cada miembro del equipo tiene su propia rama para que ninguno pueda tocar código que otro está modificando, así se reducen problemas de conflictos de ficheros. Cuando se comprueba todo y se considera que esta bien, se hace una pull request y se añade a la rama principal (main).

Para la realización de los test se creó una carpeta aparte llamada Test en el backend y se utilizó la herramienta NUnit para realizarlos.

Para la base de datos se utilizó Neon para alojar en la nube la base de datos de la aplicación, en esta se guarda todo lo que se recoge de la API de TMDB y otro tipo de información necesaria. Para facilitar el trabajo y que sea más fácil visualizar la base de datos, se conectó con Azure Data Studio que permite una mejor visualización y es mucho mejor para cuando se quieren probar las query en la base de datos.

Distribución del trabajo y horas

El líder del equipo ha sido Sergio Saura que se encargó de distribuir el trabajo de manera que todos los miembros tocaran tanto la parte del backend como la del frontend.

Se utilizó Microsoft To Do para asignar las diferentes tareas a cada miembro del grupo, en la imagen “Ilustración 1: To Do” se muestran las diferentes tareas que fueron asignadas a cada miembro.

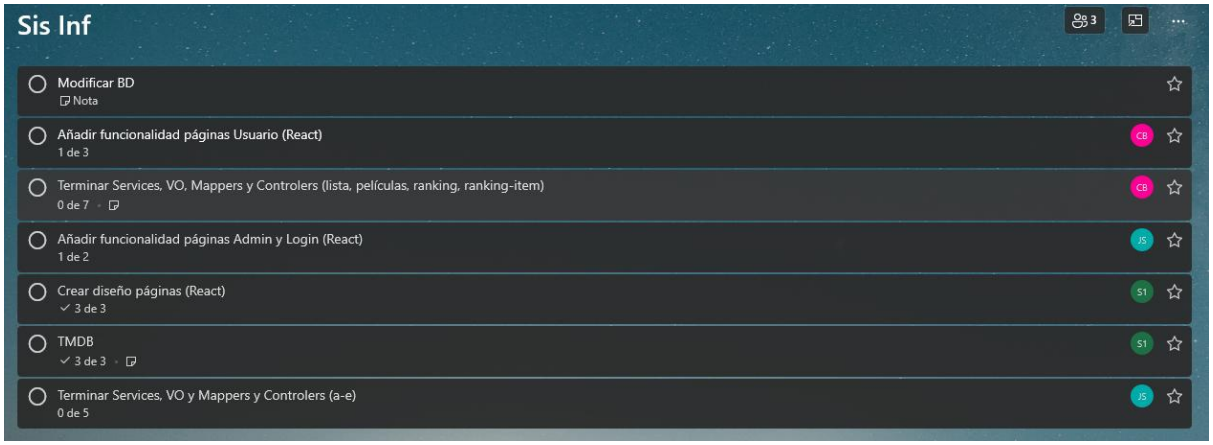


Ilustración 1: To Do

Sergio Saura se encargó la realizar la gran parte de la documentación del proyecto, se encargó de arreglar y revisar código del backend de cada rama de cada miembro, hizo una gran parte del diseño principal de la página web con algunas de sus funcionalidades básicas y se encargó de obtener varios datos de la API de TMDB y realizar un CRUD bien implementado de la tabla de películas.

Chloé se encargó de realizar el CRUD de las tablas listas, películas y ranking. Se le asignó la tarea de realizar las funcionalidades de los usuarios estándar que incluía obtener los datos en el front la cual no llegó a realizar.

José Secadura hizo el CRUD del resto de tablas que faltaban, la realización de las funcionalidades de la web, tanto de iniciar sesión y del administrador, consiguiendo conectar la base de datos con la capa web. También tuvo que realizar la tarea de Chloé haciendo absolutamente toda la funcionalidad de los usuarios.

La distribución de horas es aproximada ya que algunos miembros realizaron más tareas de las que tenía asignadas, las horas aparecen en la tabla “Tabla 1: Horas”.

Miembro	Lectura y búsqueda de documentación y recursos	Frontend	Backend	Memoria	Total
Sergio	8	16	20	4	**48**
Chloé	2	8	10	0	**20**
José	5	18	16	1	**40**

Tabla 1: Horas

Dificultades encontradas

Hubo bastantes dificultades a la hora de realizar el proyecto. Ningún miembro del equipo tenía experiencia en la interconexión frontend y backend entre React y .NET, lo cual se tuvo que buscar y probar diferentes implementaciones.

Para obtener los datos de TMDB no se empleó mucho tiempo debido a que tiene una muy buena documentación en su web. Aun así, al no tener ninguna experiencia previa con esta API sí que se empleó un tiempo para leerse la documentación y buscar las diferentes llamadas.

Para la API de YouTube hubo complicaciones para obtener los tráileres de las películas, debido a la falta de tiempo y la dificultad que supuso no se acabó de implementar.

Para realizar los test se tuvo que invertir bastante tiempo en conocer la herramienta NUnit de Visual Studio y se tuvieron que hacer varias pruebas ya que no se conseguía implementar correctamente.

Otra dificultad que impidió poder terminar todo el proyecto correctamente y que todo funcionara como se había planteado fue la carga desigual final de trabajo entre los miembros del equipo por algunos fallos u otros. Sergio y José tuvieron que asumir un mayor volumen de tareas finalmente, lo que provocó que no todas las funciones planeadas se completaran al 100%.

También se invirtió mucho tiempo corregir varias partes que se realizaron en lo que también se invirtió bastante tiempo.

Bibliografía

Aquí se mencionarán o se pondrán los enlaces de diferentes sitios que se han visitado:

Nunit:

https://docs.nunit.org/

https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-nunit

TMDB:

https://developer.themoviedb.org/reference/getting-started

https://www.reddit.com/r/webdev/comments/16vbvni/help_with_the_movie_database_api_genres/

.NET:

https://learn.microsoft.com/es-es/dotnet/

https://stackoverflow.com/questions/tagged/.net

Base de datos:

https://neon.com/

https://neon.com/docs/introduction

React/Typescript:
https://tailwindcss.com/

https://react.dev/learn/typescript

https://www.typescriptlang.org/docs/handbook/react.html

https://nodejs.org/es/learn/getting-started/introduction-to-nodejs

General:

https://chatgpt.com/
https://github.com/gothinkster/realworld