Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Matemáticas e informática

Trabajo Fin de Grado

Desarrollo de una Aplicación Web para la Gestión de Videojuegos.

Autor: Jinxiang Ye
Tutor(a): Antonio Latorre

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Matemáticas e informática

Título: Desarrollo de una Aplicación Web para la Gestión de Videojuegos.

Junio 2025

Autor: Jinxiang Ye
Tutor: Antonio Latorre
Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

La adquisición de videojuegos online está experimentando un creciente desarrollo en España. Mientras que ya existen aplicaciones web que resuelven las necesidades generales de los usuarios, la mayoría carece de herramientas en la gestión de datos personalizados, como información de coleccionismo.

El propósito de este proyecto es el desarrollo de una aplicación web orientado a la gestión de bases de datos de videojuegos, implementando las funcionalidades básicas que disponen las aplicaciones actuales, además de poder personificar los juegos que guardas en tu propia colección.

Se describe el proceso de creación de dicha página, desde la selección de tecnologías hasta la implementación de las llamadas externas, justificando las decisiones que se han tomado en el proceso.

Los sistema final incluye las siguientes funcionalidades: una amplia navegación por el catálogo de videojuegos existentes, la posibilidad de almacenar en una colección personal, registro de metadatos personalizados de dichos videojuegos, tales como notas personales, partidas personificadas, datos de colección y una módulo de administración para gestionar los videojuegos en la base de datos.

El diseño de esta aplicación pretende ser el comienzo de un proyecto a mayor escala cuyo objetivo final es convertirse en una plataforma digital de intercambio de juegos físicos.

Abstract

Online video game purchasing is growing in Spain. While several web applications already cover the general user needs, most of them lack features for managing personalized information such as collection-target information.

The aim of this project is to develop an web application for managing a video-game database, providing standard features offered by the current solutions and, furthermore, lets users personalize the games stored in their collection.

This thesis describes the development process, from the technology selection to the implementation of external API calls, justifying the decisions made.

Finally the system offers the following functionalities: browsing through an extensive video-game catalog, the posibility of storing titles in a personal collection, storage of custom metadata, such as personal notes, playthroughs, collector details, and an administration module for managing the games in the database.

The proposed design pretendes to be a starting point for a larger-scale projecto whose ultimate goal is to become a digital platform for trading physical games.

Tabla de contenidos

Capítulo 1

Introducción

El mercado del videojuego en España ha registrado una facturación de 2.408 Millones de euros [1], lo que indica un crecimiento del 3 % en comparación con el año anterior. De toda esta cantidad se destacan los 1.558 Millones de euros [1] que corresponden a las ventas de videojuegos online mientras que las ventas físicas retrocedieron un 18,11 % [1], con únicamente 800 Millones de euros [1]. Esta tendencia muestra como cada vez más usuarios se mueven al contexto digital a la hora de adquirir videojuegos.

Debido a la creciente popularidad en el ámbito digital, han surgido páginas que permiten administrar videojuegos online como son IGDB, RAWG, Backloggd. Aportando a los usuarios un lugar donde guardar, almacenar y explorar videojuegos online. Sin embargo, esta digitalización nos muestra la necesidad de traer ese mercado físico, especialmente para los coleccionistas y aficionados a los juegos tradicionales. En este sentido, el proyecto busca proporcionar una plataforma que no solo facilite la navegación de videojuegos online sino que además contiene funcionalidades para administrar y gestionar información de los videojuegos físicos. Esta solución pretende ser el comienzo de una plataforma que da visibilidad a la comunidad alrededor de los videojuegos físicos.

Desde el punto de vista personal, tengo una moivación muy concreta para realizar este proyecto: el deseo de aplicar mis conocimietos sobre el diseño web, adquiridos en mi estancia en las prácticas, en un ambito de especial interes para mí como lo es el de los videojuegos. Aunque anteriormente mis tareas en la empresa consistían principalmente en el desarrollo del front-end, esta experiencia como de desarrollador full-stack representa una oportudad muy valiosa para ampliar mis conocimientos en el ambito de back y bases de datos. Este proyecto supondrá un gran reto tanto académico como personal.

1.1. Objetivos generales

Los objetivos generales que tendremos que completar en este proyecto son los siguientes:

- Identificación y especificación de las herramientas software que se vayan a utilizar para el desarrollo del sistema (en el caso particular Django, Python y MariaDB).
- Diseño de las tres partes fundamentales de la aplicación: base de datos, interfaz visual y especificaciones funcionales.
- Implementación de la aplicación siguiendo rigurosamente los requisitos establecidos en la fase de diseño.
- Validación mediante un plan de pruebas, en cuanto a seguridad, situaciones extremas y corrección de errores.

1.2. Objetivos específicos

Los objetivos generales anteriores permitirán llevar a cabo los objetivos específicos relacionados con las funcionalidades que proporciona la página, estos son:

- Desarrollar una plataforma que permita navegar por un catálogo de videojuegos con opciones de filtrado avanzado.
- Implementar funcionalidades básicas **CRUD** [2] (Create, Read, Update y Delete) para la gestión de videojuegos, con un manejo amistoso.
- Permitir guardar videojuegos, tanto en tu colección personal, como en estanterías personalizadas para mejor acceso a ellos.
- Permitir proporcionar metadatos adicionales sobre los videojuegos guardados, tales como notas personales sobre dicho producto, partidas realizadas.

1.3. Siguientes capítulos

El resto del trabajo queda dividido en:

- Estado del arte, donde exploraremos el mercado, explicaremos los conceptos claves y donde se justificará las tecnologías utilizadas.
- Desarrollo, compuesto por análisis, diseño e implementación. En los que se detallan el proceso de creación de la página.
- Análisis de impacto.
- Resultado, conclusiones y trabajo futuro.

Capítulo 2

Estado del arte y preliminares

2.1. Estado del arte

En el desarrollo de aplicaciones orientadas a la gestión de videojuegos online, destacan las siguientes plataformas por su funcionalidades especializadas:

- **HLTB** [3] (How long to beat), destaca en almacenar y gestionar los tiempos medios necesarios para completar los videojuegos según datos aportados por la comunidad.
- **IGDB** [4] (The Internet game database), plataforma adquirida por Twitch, es una extensa base de datos de videojuegos, especializado en la completitud de los metadatos, además de permitir a sus usuarios guardar, crear y modificar datos concretos de los videojuegos.

Mientras que cumplen satisfactoriamente con sus respectivas áreas, estos presentan aspectos mejorables especialmente en el ámbito relacionado con datos más personales como notas, datos de coleccionismo, etc. El objetivo será combinar las ventajas de ambos y añadir las opciones personalizables que se menciona anteriormente.

2.2. Preliminares

A continuación se presentan unos conceptos claves necesarios para entender la estructura y las tecnologías que se utilizan en este trabajo.

- **Modelo Vista Controlador** [5] (MVC), se trata de una arquitectura que separa una aplicación en tres capas independientes:
  - Modelo: Es la capa que gestiona los datos, es decir la que tiene acceso a la base de datos.
  - Vista: Es la capa dedicada a la gestión de la interfaz que se le muestra al cliente.

- Controlador: Capa dedicada a la lógica de negocios, la parte funcional que recoge y manipula los datos que recibe del Modelo para mostrarlos en la vista.

Este patrón separa las tres partes fundamentales de la programación web, que son: base de datos(Modelo), front(Vista) y back(Controlador). Entre los frameworks/ aplicaciones que facilitan el desarrollo de esta arquitectura, destacan Laravel [6] (Que usa como lenguaje de programación PHP) y Django [7] (Que usa como lenguaje de programación Python).

- **Mariadb**, se trata de un sistema de gestión de bases de datos relacional, al igual que MySQL [8]. Entre sus ventajas se pueden destacar la simplicidad, un rendimiento adecuado para proyectos pequeños y medianos. Debido a su simplicidad y eficacia para aplicaciones con un tamaño no excesivo, es una opción muy llamativo que no exige excesivo conocimiento técnico.

- **Métodos CRUD** [2]: Se trata del conjunto de operaciones básicas sobre un objeto. Dichas operaciones consisten en la *creación* (Create), *lectura* (Read), *actualización* (Delete), *borrado* (Delete) sobre el objeto Game en la base de datos.

- **API** [9] (Application Programming Interface) es el conjunto de reglas que permite a dos sistemas intercambiar información. En este proyecto se hablarán de dos tipos de APIs:

  1. API externa, en este caso IGDB, se trata de peticiones que se realizan hacia una plataforma externa mediante peticiones HTTP de tipo REST.

  2. API interna, creadas por en el propio framework de Django, que usualmente se suelen llamar por los ficheros javascript.

- **Fetch** [10], interfaz de javascript que se utiliza para realizar llamadas sobre una api. Es especialmente útil para actualizar las páginas sin necesidad de actualizar la misma. Estos pueden tener método (Get, Post) y un cuerpo donde se le pasan los parámetros. Tras ello recogen la respuesta y actualizan la información de la página sin recargarlo.

- **Template**, se trata de ficheros de texto que separan los datos de la aplicación de la forma en los que se presentan. En la mayoría de casos se tratan de ficheros con extensión HTML. Por ejemplo, en el caso de Django:

  ```
  <h1> {{ valor }} </h1>
  ```

- **Wireframes** [11]: es un esquema de baja fidelidad que representa la estructura básica de una página sin colores ni detalles gráficos. Su función principal es facilitar la estructuración de la información.

2.3. Decisiones tomadas

2.3.1. Framework

Una de las técnicas más usadas en la programación actual es la de utilizar un framework de front-end(React, Angular) y otro de back-end(Node.js, Express JS) completamente independientes, lo que facilita el trabajo paralelo de ambas partes en equipos de trabajo. Pero la decisión tomada para utilizar un solo framework que funciona tanto para implementar front como back tiene como respuesta los siguientes puntos:

- Experiencia previa con Laravel, que al igual que Django ofrece la posibilidad de trabajar en front y back, facilitando su interconexión y reduciendo así la curva de aprendizaje.
- Integración más eficiente de front-end y back-end eliminando la necesidad de gestionar dos frameworks independientes.

Por otra parte, la elección de Django ante Laravel es debido a la mayor popularidad del lenguaje Python respecto al de PHP.

2.3.2. Base de datos

En la arquitectura global de la plataforma, la base de datos constituye uno de los pilares fundamentales del proyecto. Entre las opciones que presenta el mercado, se ha optado por una SGBD(sistemas de gestión de base de datos) relacionales de código abierto: MariaDB y PostgreSQL.

Factor	**MariaDB – Ventajas / Desventajas**	**PostgreSQL – Ventajas / Desventajas**
Ámbito de aplicación	**+** Ideal para prototipos y aplicaciones con baja concurrencia. **–** Menos eficiente ante escrituras masivas y consultas muy complejas.	**+** Orientado a entornos empresariales con escrituras frecuentes y lógica SQL avanzada. **–** Sobredimensionado para proyectos centrados casi exclusivamente en lectura.
Curva de aprendizaje	**+** Configuración sencilla. **–** Capacidades avanzadas limitadas.	**+** Motor muy completo y extensible. **–** Requiere experiencia previa, es decir mayor complejidad inicial.

Cuadro 2.1: Comparativa entre MariaDB y PostgreSQL [12]

Otro punto en el que nos debemos basar para la elección es el método con el que se poblará la base de datos. Para este proyecto se elige el **poblado incremental**

bajo demanda porque nos enfrentamos a un catálogo con una cantidad enorme de elementos que una mayoría nunca se consumen.

Tras el análisis comparativo y la definición del método de colección datos, se puede concluir que **MariaDB** es la opción que encaja mejor en este proyecto, con el flujo de carga incremental y la rápida lectura de datos.

Capítulo 3

Desarrollo

3.1. Análisis

3.1.1. Objetivos funcionales

El sistema debe ofrecer las funcionalidades listadas a continuación.

1. **CRUD de videojuegos** Debe permitir la creación, edición y eliminación de videojuegos.
2. **Búsqueda de juegos**. Debe permitir la búsqueda por nombre y el filtrado de juegos tanto por plataforma, como por género.
3. **Visualización de detalles** Debe permitir ver detalles del juego como *Fecha de lanzamiento*, *Plataformas* y más metadatos del juego.
4. **Gestión de la colección personal** Debe permitir añadir o eliminar títulos de *la colección*. Al igual que los filtrados para buscar juegos coleccionados.
5. **Metadatos personalizados** Debe permitir añadir a los juegos guardados datos personalizados como *precio de compra*, *datos de coleccionista*, etc.
6. **Estanterías virtuales** Debe permitir crear y eliminar estanterías (`Shelf`) y agrupar juegos en ellas. Un juego puede estar en ninguna, una o varias estanterías.

3.1.2. Modelo de ciclo de vida y justificación

El proyecto adopta el **modelo en cascada** por:

1. Tener requisitos bien definidos y estables desde el principio.
2. Ser desarrollador único contra un cliente que actúa como üsuario-propietario".
3. Facilidad para sincronizar entregables con los hitos académicos.

Fases identificadas:

1. **Especificación de requisitos (6h):**

- Definir funcionalidades principales y secundarias que tendrá la página.
- Definición de tecnologías a usar para el desarrollo de la aplicación.

2. **Análisis (15h):**
   - Estudio e investigación de las tecnologías identificadas, framework, base de datos, APIs.
   - Definir la arquitectura del sistema.
3. **Diseño (55h):**
   - Modelado de la base de datos.
   - Creación de wireframes y diseño UI/UX.
   - Diseño de la API y estructura del backend.
4. **Codificación (165h):**
   - Desarrollo del backend.
   - Implementación del frontend.
   - Integración con la API externa.
   - Implementación de vistas y filtrado de datos.
5. **Validación y pruebas (28h):**
   - Definir casos de prueba.
   - Realizar pruebas funcionales y de rendimiento.
   - Corrección de errores.
6. **Documentación y presentación (28h):**
   - Redacción de la memoria del TFG.
   - Preparación de la presentación.

3.1.3. Diagrama de gantt

La Figura 3.1 muestra el diagrama de Gantt con las fases identificadas.

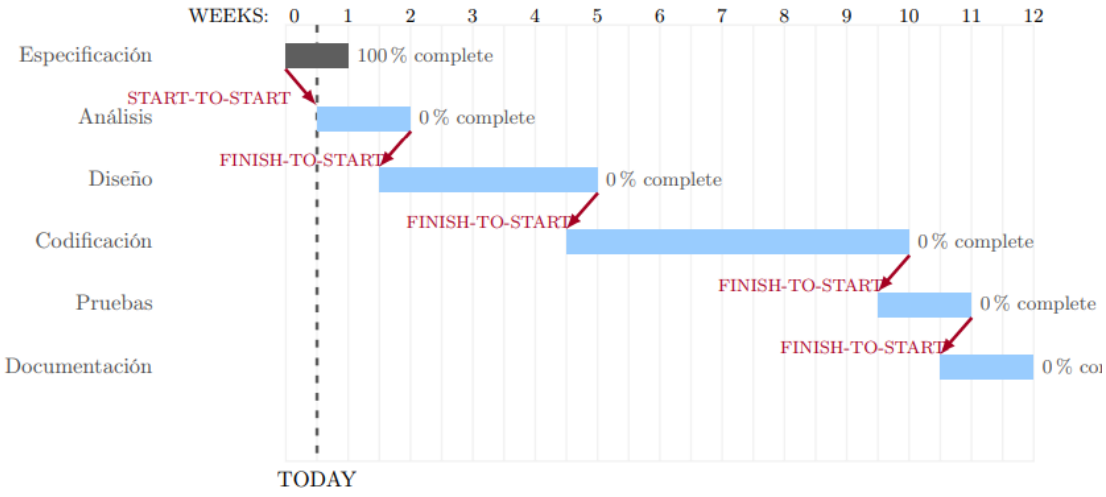


Figura 3.1: Diagrama de Gantt Inicial

3.1.4. Supuestos y limitaciones

- **API IGDB**. Límite: 4 peticiones por segundo.
- **Concurrencia**. Sistema orientado a un único usuario. La creación de más usuarios queda fuera del alcance de este proyecto.
- **Usuario/administrador**. Por simplicidad, se supone que el usuario que usará el sistema es al mismo tiempo el administrador, teniendo acceso de edición y borrado sobre la base de datos.
- **Datos multimedia**. Las imágenes se guardan como enlaces y no como imágenes, luego se requiere de acceso a internet para una experiencia agradable.
- **Lenguaje único**. Interfaz sólo disponible en castellano.

3.2. Diseño

3.2.1. Diseño del Modelo lógico

La Figura 3.2 ilustra el modelo lógico del sistema. Se compone principalmente de:

- **Game**: entidad principal que contiene todo la información de cada videojuego (título, fecha de lanzamiento, portada, etc.) Tiene una referencia a sí misma pues si se trata de un *dlc* o un *mod* estos señalarán al juego padre relacionado. Los datos obligatorios para la Entidad `Game` son:
  - `title`: Título del videojuego.
  - `region` – Región donde el juego está disponible.

- `description` – Descripción del juego.
- `release_date` – Fecha oficial de lanzamiento.
- `added_date` – Fecha en la que el registro en la base de datos.

- **Entidades de referencia**: tablas como `Platform`, `Genre`, `Developer`, `Publisher`, `Franchise` y `Label` se utilizan para normalizar los atributos de `Game`. Todas ellas tienen el atributo de `id` junto al de `name`.
- **Datos personalificables**: tablas como `Collection`, `Price`, `Playthough`. Esta información adicional puede ser añadida a un juego una vez esta esté almacenada en la base de datos.
- **Tablas N:M**: las tablas intermedias representan las relaciones N:M entre las entidades. Para ilustrar, `GamePlatform`, puede haber distintas plataformas en las que está disponible un videojuego, al igual que puede haber muchos videojuegos que están disponibles una misma plataforma.

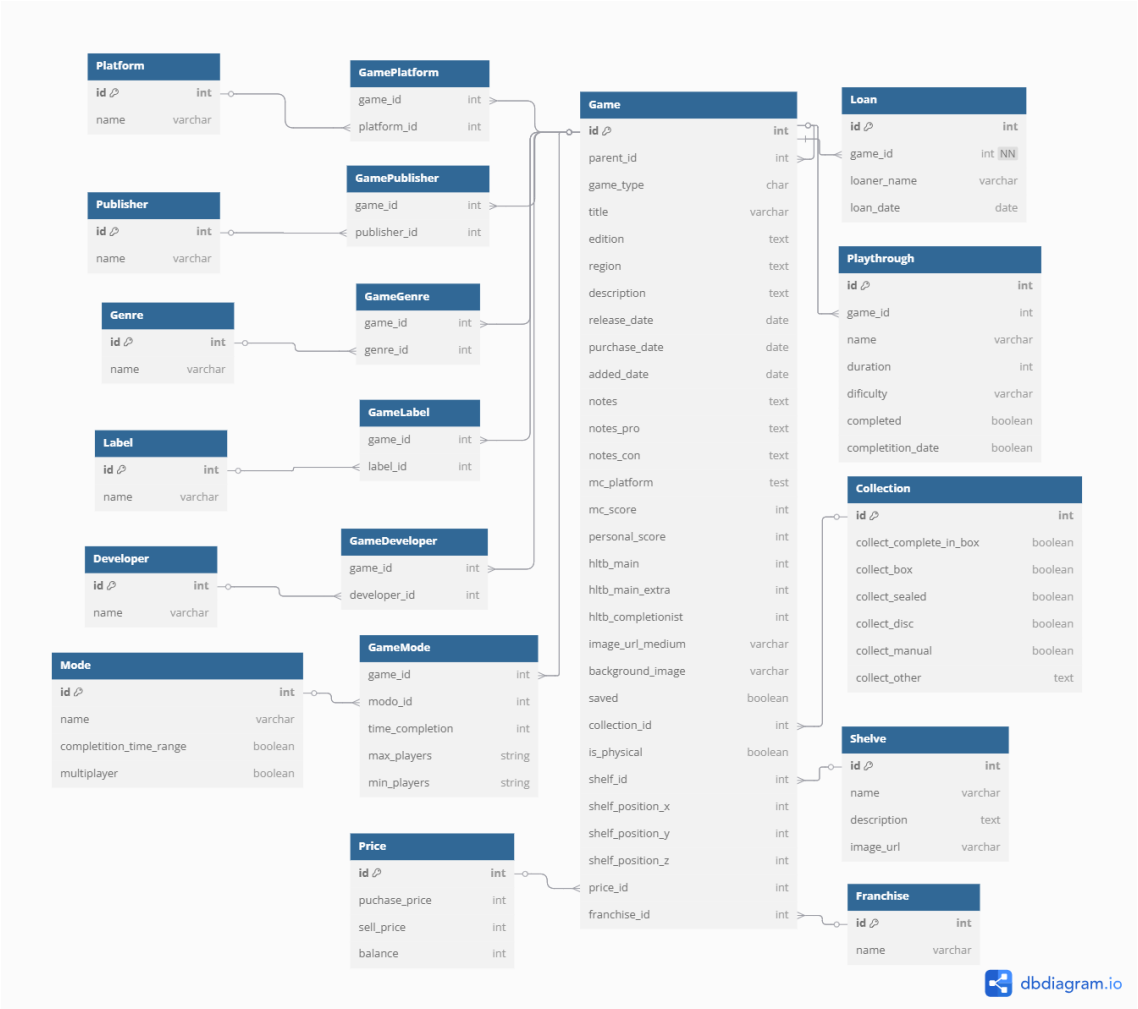


Figura 3.2: Diagrama entidad-relación

3.2.2. Arquitectura de la aplicación

La arquitectura de Django dividida en 3 capas gestiona una petición HTTP de la siguiente forma: (Figura 3.3):

1. Al recibir una petición de llamada, busca en el fichero urls.py si existe la llamada que se pide, en caso de no estar definida lanza un error 404 de página no encontrada (Figura 3.4).

2. En caso de que lo haya encontrado, urls.py llamará a la función asociada en el fichero view.py.

3. Dicha función accederá mediante los modelos a la base de datos ya sea para leer como escribir información

4. Por último devolverá la información obtenida junto con una pantalla de extensión .html como respuesta.

Luego la estrutura de Django queda como:

Presentación Templates HTML con Bootstrap, css y javascript.

Aplicación `views.py` y `utilidades.py` que encapsulan la lógica de negocio y las llamadas externas.

Dominio Modelos Django en `models.py` conteniendo las entidades de la base de datos: `Game`, `Platform`, `Shelf`, etc.

Infraestructura MariaDB, enrutamiento(`urls.py`), plantillas(`/games/plantillas/.html`) y ficheros estaticos como son imagenes, estilado y js (`/static`).

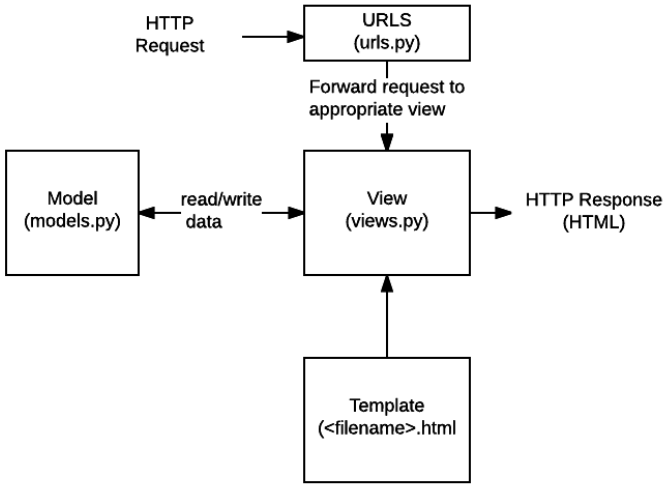


Figura 3.3: Arquitectura lógica por capas. Fuente: *Django — Introducción*, Mozilla Contributors, MDN Web Docs (CC BY-SA 2.5).

Figura 3.4: Error al no estar definido la url

3.2.3. Diseño de la interfaz de usuario

Para el diseño de las wireframes se ha usado la herramienta de Figma. A continuación se muestran el diseño de las componentes y pantallas iniciales:

Header con buscador

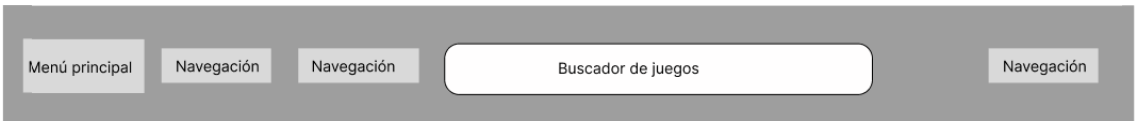


Figura 3.5: Diseño header con buscador

Carta de Juego

Figura 3.6: Diseño de las cartas de juego

Filtro

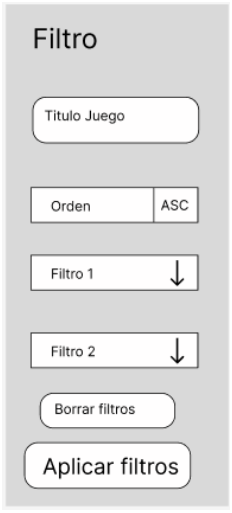


Figura 3.7: Diseño del filtro de juegos

Pantalla de búsqueda

Figura 3.8: Diseño de pantalla con juegos y filtro

Menú principal

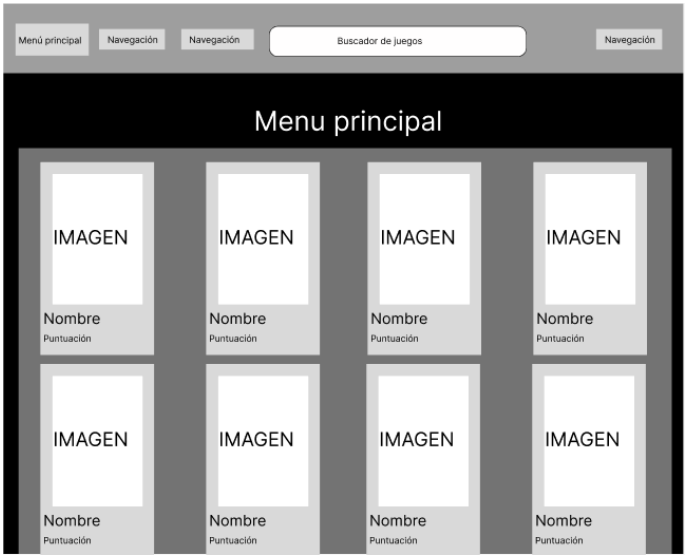


Figura 3.9: Diseño del menú principal

Creación de juegos

Figura 3.10: Formulario de creación o edición de juego

3.3. Implementación

En esta sección se describirá el proceso de construcción de la aplicación, desde la inicialización y configuración inicial, hasta las funciones principales de la aplicación y la creación de las pantallas. La estructura que sigue esta sección es la siguiente:

1. **Requisitos previos**: software y herramientas mínimas.
2. **Inicialización del proyecto**: creación del entorno de trabajo.

3. **Configuración personalizada**: ajustes del fichero `settings.py` y `urls.py`
4. **Modelos principales**: migración de la base de datos con las entidades y relaciones.
5. **Llamadas externas**: integración API de IGDB y recogida de datos de HLTB.
6. **Lógica de negocio**: vistas y servicios de `views.py`
7. **Front-end, Templates y Formularios**: Estructura de los ficheros html, carga de estilos y renderizado de formularios.

3.3.1. Requisitos previos

Antes de empezar con la creación del proyecto en Django, se tiene que tener instalado las siguientes herramientas:

- **Python:** Versión 3+.
- **IDE:** En este caso Visual Studio Code.
- **Gestor de paquetes:** Es recomendable disponer de un gestor de paquetes como puede ser `pip`.

3.3.2. Inicialización del proyecto

Para crear un proyecto en Django, primero es necesario la instalación de sus bibliotecas, esto se consigue ejecutando en la línea de comandos la siguiente línea:

```
$ pip install Django
```

Una vez instalado correctamente, se pasa a la creación del proyecto. Esto se consigue con las siguientes líneas de comando:

```
$ django-admin startproject Proyecto
$ cd Proyecto
$ python manage.py startapp games
```

Una vez finalizado esto, tendremos una estructura de ficheros parecido a lo siguiente:

Estructura resultante

```
Proyecto/
  Proyecto/
     __init__.py
     settings.py      # ajustes del proyecto
     urls.py          # enrutamiento
     wsgi.py
     asgi.py
  games/               # app principal
     models.py         # donde se registran los modelos
```

```
views.py          # lógica de negocios
apps.py
tests/
...
```

3.3.3. Configuración personalizada

A continuación pasamos a ver las modificaciones a los ajustes y personalizaciones que se han hecho en el proyecto.

settings.py

- Incluir la app `games`:

```
INSTALLED_APPS = [
    ...,
    'games',
]
```

- Base de datos: En este caso, se ha usado Mariadb desplegado localmente.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "videogames",
        "USER": "[Tu usuario]",
        "PASSWORD": "[Tu contrasena]",
        "HOST": "127.0.0.1",
        "PORT": "3306",
        "OPTIONS": {"charset": "utf8mb4"},
    }
}
```

Projecto/urls.py

Para que Django reconozca las rutas definidas en la app `games`, es necesario incluir su fichero de rutas en el enrutamiento de la raiz. Para ello, se crea el fichero `games/urls.py` con un contenido similar a la siguiente:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.start, name="menu"),
    ]
```

Tras ello, se añade al enrutamiento base del proyecto `Projecto/urls.py`:

```
from django.urls import path, include
urlpatterns = [
```

```
    path("", include("games.urls")),
]
```

En adelante, cuando se referencia a añadir rutas, siempre se tendrá como referencia `games/urls.py`.

3.3.4. Modelos principales

Para construir la base de datos en Django, se usará el fichero `models.py`, en donde se definirán los modelos de entidad-relación para posteriormente migrar la base de datos con ello. La estructura de este fichero se basa en el diseño que se ha hecho de la base de datos, anteriormente mencionado. A continuación, como ejemplo, se muestra el modelo de la Entidad `Playthrough`:

```
class Playthrough(models.Model):
    name = models.CharField(max_length=255, blank=True)  # Optional
    game = models.ForeignKey(
        'Game',
        on_delete=models.CASCADE,
        related_name='playthroughs'
    )
    duration = models.IntegerField()  # in minutes
    difficulty = models.CharField(max_length=255, blank=True)
# Optional
    completed = models.BooleanField(default=False)
    completition_date = models.DateField(blank=True, null=True)

    def __str__(self):
        return f"Playthrough: {self.name}, {self.duration}, {self.difficul
```

Una vez finalizado, se procede a migrar los cambios con los siguientes comandos:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

3.3.5. Llamadas externas y lógica de negocio

La plataforma consume la API pública de IGDB para capturar metadatos de videojuegos, una parte esencial para la búsqueda y creación de juegos en la base de datos. A continuación se va a ver como se implementa esta servicio.

Obtención del *access token* de IGDB

El servicio de IGDB emplea el sistema de autenticación de Twitch.[1] Para obtener un *access token* válido se siguen los pasos:

[1] `https://api-docs.igdb.com/#account-creation`

1. **Registro de la cuenta**: crear una cuenta de Twitch para obtener las claves `client_id` y `client_secret`.
2. **Solicitud del token**: enviar la petición `POST` a la dirección `https://id.twitch.tv/oauth2/token` con los parámetros `client_id`, `client_secret` y `grant_type=client_credentials`
3. **Aceptación de la respuesta**: El servidor devuelve un objeto JSON como el de la Figura 3.1. Ese campo `access_token` será valido durante aproximadamente 65 días (5587808 seg).

Listing 3.1: Respuesta de la autenticación de Twitch

```
{
  "access_token": "access12345token",
  "expires_in": 5587808,
  "token_type": "bearer"
}
```

Una vez conseguida las claves, para evitar que sean accesibles, se guardan en el fichero `.env` creado en la raíz del proyecto. Posteriormente se recogen en el fichero `settings.py` de la siguiente manera:

```
import os
from dotenv import load_dotenv
BASE_DIR = Path(__file__).resolve().parent.parent
load_dotenv(dotenv_path=BASE_DIR / '.env', override=True)

CLIENT_ID = os.getenv('Client_ID')
ACCESS_TOKEN = os.getenv('Access_Token')
```

Integración API

Para la integración, se ha decidido utilizar un fichero adicional, nombrado `utilidades.py`. En ella se define la función *fetch igdb* con parámetros:

- endpoint: el nombre del dato que se desea solicitar. (Por ejemplo: /games si se trata de juegos o /genres si se trata de generos).
- fields: los atributos de ese dato que se quieren capturar como lista.
- limit: cantidad de elementos de ese dato que se pide.
- offset: indice desde donde se empieza a hacer la busqueda.
- search: atributo a buscar del dato.
- filters: filtros que se aplican en la búsqueda.
- sort: orden en el que vienen los datos.

Esta función recoje las variables de Client-ID y ACCESS TOKEN que se ha almacenado en settings para la validación y devuelve una lista de objetos como la siguiente:

```
{
    'id': 136396,
    'cover':
        {'id': 163581,
         'image_id': 'co3i7x'
        },
    'name': 'Hollow Knight: Lifeblood',
    'rating': 79.79989874614452
}
```

Para ver el código completo de `fetch_igdb` (y funciones auxiliares), consulte el Anexo B.

Listing 3.2: Ejemplo uso fetch_igdb

```
data = fetch_igdb(
            endpoint="games",
            fields=["id", "name", "cover.image_id", "rating"],
            limit=20,
            filters=filters_string or None
        )
```

Datos de duración de HLTB

Para recoger los datos de la duración media de los videojuegos, se ha usado la librería de *howlongtobeatpy* [13]. Esta API no oficial utiliza *webscrapping* **ref_webscrapping** para recoger los datos del tiempo medio de un juego. Para ver la función completa, consulte el Anexo B.

Lógica de negocio `views.py`

El módulo de `views.py` es el controlador de las peticiones HTTP: recibe las solicitudes de los clientes, aplica la lógica de negocio y retorna una respuesta en formato `JsonResponse` o una template. Se distinguen en dos tipos de vistas:

a) **Acciones de escritura ("write")**, que realizan modificaciones en la base de datos y devuelven un JsonResponse. Se puede apreciar en el Código 3.3, que siguen una secuencia muy estructurada:

   1. Validan el método de la petición.
   2. Obtienen el objeto de la base de datos.
   3. Se realizan las operaciones atómicas y se guardan usando *update_fields['atributo específico']* para reducir consultas.
   4. Devuelven respuesta JSON para la función fetch.

b) **Acciones de lectura (ŕead")**, que realizan consultas y renderizan un template. La secuencia de estos se puede describir como:

1. Pedir datos dado un parámetro de llegada o del propio *request*.
2. Retornar como un template tal que los datos se inyecten de manera directa sobre el template correspondiente.

Esta clasificación ayuda a entender de manera más clara los posibles errores que pueden tener, a la hora de hacer pruebas.

Listing 3.3: Funciones tipo write

```
'''
Funcion para quitar juego de la colección.
'''

def take_game_from_collection(request, pk):
    if request.method != "POST":
        return JsonResponse({"detail": "Método no permitido", "ko": 1}, st

    game = get_object_or_404(Game, pk=pk)
    game.saved = False
    game.save(update_fields=['saved'])
    return JsonResponse({"detail": "Juego eliminado de la colección", "ok"
```

Listing 3.4: Funciones tipo read

```
'''
Pantalla y métodos para buscar juegos.
'''

def view_redirigir_a_busqueda(request):
    # Cargamos plataformas y géneros para los filtros
    generos = Genre.objects.values_list('id', 'name').distinct()
    plataformas = Platform.objects.values_list('id', 'name').distinct()

    return render(request, "games/pantallas/Pagina_buscador.html", {
        "platforms": plataformas,
        "genres": generos,
        "initial_query": request.GET.get("q", "").strip(),
    })
```

3.3.6. Front-end y Templates

Plantillas hereditarias

Para la implementación de las pantallas, se ha utilizado unas plantillas hereditarias, es decir ficheros con una estructura general de la aplicación tal que cualquier otro archivo lo puede incluir en su contenido, por ende "heredar"su estrutura. Para ello se define un fichero layout `New_app.html` (Ver código 3.5) que incluye un fichero de estilado, un bloque de estilado, de contenido y de código, en la cual sus herederos podrán rellenar en caso de necesitarlo. dimensión y un header donde está el buscador de títulos.

Vista / Función	Responsabilidad principal	Tipo
`start`	Menú principal con los juegos más populares (IGDB)	Read
`view_crear_juego`	Muestra un formulario vacío para crear un juego	Read
`view_editar_juego`	Muestra un formulario con datos para editar	Read
`view_ver_juego`	Muestra pantalla de juego	Read
`view_redirigir_a_busqueda`	Renderiza buscador con filtros iniciales	Read
`search_games`	Devuelve resultados de búsqueda (JSON)	Read
`view_mostrar_coleccion`	Renderiza la colección filtrable	Read
`api_coleccion`	API JSON de la colección (con filtros)	Read
`api_coleccion_not_in_shelf`	Juegos no asignados a una estantería	Read
`admin_manage_games`	Panel admin (vista) con filtros	Read
`admin_get_games`	API JSON para el panel admin	Read
`shelf_list`	Lista de estanterías + formulario alta	Read
`shelf_detail`	Vista detallada de estantería (juegos por z)	Read
`eliminar_juego`	Borra un juego de la BD	Write
`add_igdb_game`	Añade un juego (vía IGDB) a la colección	Write
`get_or_create_game`	Obtiene o crea un `Game` desde IGDB	Utility
`add_edit_game_manual`	Alta/edición manual de juego	Write
`take_game_from_collection`	Marca el juego como no guardado	Write
`manage_playthrough`	Añade/edita partidas	Write
`manage_playthrough_delete`	Elimina una partida	Write
`manage_notes`	Guarda notas y puntuación personal	Write
`manage_price`	Actualiza o crea precio	Write
`manage_collection`	Crea/actualiza datos de coleccionista	Write
`create_shelf`	Crea una nueva estantería	Write
`delete_shelf`	Elimina estantería y limpia referencias	Write
`add_games_to_shelf`	Añade juegos a estantería	Write
`remove_game_from_shelf`	Quita juego de estantería	Write

Cuadro 3.1: Resumen de vistas y utilidades en `views.py`

Listing 3.5: Plantilla `New_app.html`

```
{% load static %}
<!DOCTYPE html>
<html lang="es" dir="ltr">
  <head>
    <meta charset="UTF-8" />
    {% block title %}
      <title>Games</title>
    {% endblock %}
    <link rel="stylesheet" href="{% static 'css/main.css' %}" />

    {% block styles %}

    {% endblock %}
  </head>

  <body class="custom-body">
    {% include 'games/includes/Header_buscador.html' %}

    {% block content %}

    {% endblock %}

    <script src="{% static 'js/main.js' %}"></script>
    {% block scripts %}

    {% endblock %}
  </body>
</html>
```

Así cualquier otra pantalla que quiera heredar de este layout, tendrá que añadir el resto del contenido en el bloque *content*, un ejemplo de esto se puede observar en el Código 3.6:

Listing 3.6: Página_principal.html

```
{% extends "games/layouts/New_app.html" %}
{% block title %}<title>Página Principal</title>{% endblock %}

{% load static %}

{% block styles %}

<link rel="stylesheet" href="{% static 'css/pagina_principal.css' %}">
{% endblock %}

{% block content %}

<subsection class="destacados-subsection">
```

```
    <h2>Juegos más destacados</h2>
    <div class="destacados-container">
        <!-- Contenido -->
    </div>
</subsection>

{% endblock %}
```

Los templates resultantes son los siguientes:

Template (`.html`)	**Descripción**
Admin_manage_page.html	Panel de gestión de los videojuegos, pudiendo editar y eliminar juegos de la base de datos.
Coleccion.html	Módulo en el que se muestran los juegos guardados en la colección.
Form_juego.html	Pantalla de creación/edición de juego
Nueva_pagina_juego.html	Pantalla principal de un juego, en caso de estar en la colección, tiene acceso a editar datos avanzados.
Pagina_buscador.html	Pantalla de búsqueda de videojuegos con filtros.
Pagina_principal.html	Menú principal donde se muestran los juegos más populares.
Shelf_detail.html	Pantalla que muestra los juegos que pertenecen a una estantería.
Shelf_list.html	Muestra la lista de estanterías que están registradas

Cuadro 3.2: Listado de pantallas implementadas en la aplicación `games`

Organización de ficheros estáticos

Del ejemplo anterior, se puede observar que se carga el archivo de *static*, esto indica a Django la ubicación donde se encuentran las imágenes, los estilados y los códigos js.

- `/static/css`: Contiene todos los estilados necesarios para las pantallas.
- `/static/img`: Carpeta conteniendo las imagenes usadas por las pantallas.
- `/static/js`: Carpeta conteniendo los ficheros js para el correcto funcionamiento de los componentes de las páginas.

Capítulo 4

Resultados y conclusiones

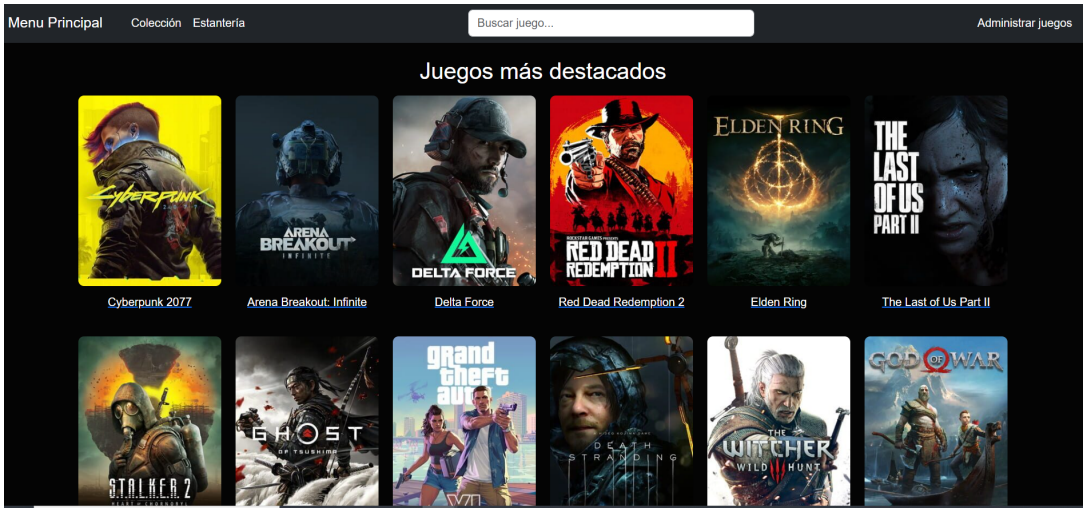
4.1. Demostración

En esta sección se ilustra el flujo de 4 tareas comunes siendo estas:

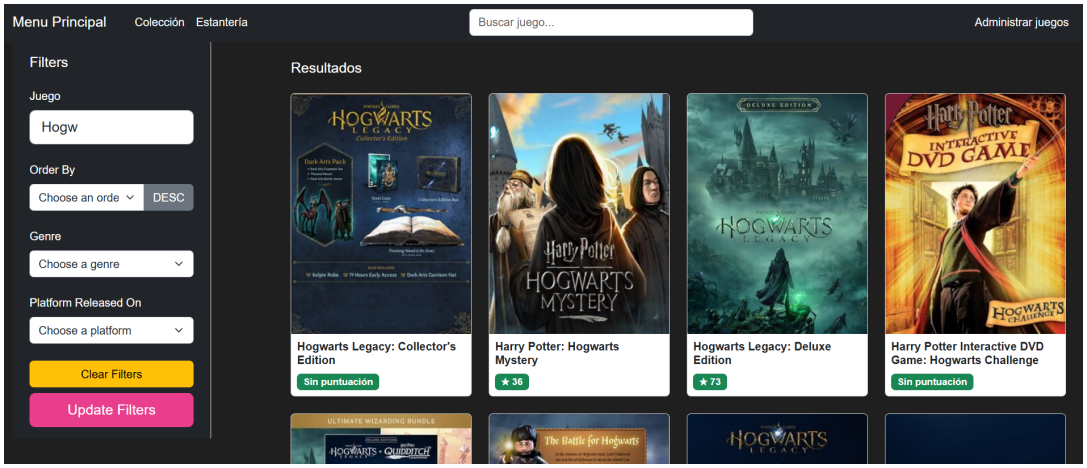
1. Búsqueda del juego *Hogwarts Legacy* y añadirlo a la colección.
2. Registro de notas personales y datos de coleccionismo.
3. Crear una estantería e introducir el juego en ella.
4. Buscar y eliminar el juego de la base de datos.

4.1.1. Tarea 1: Búsqueda de juego

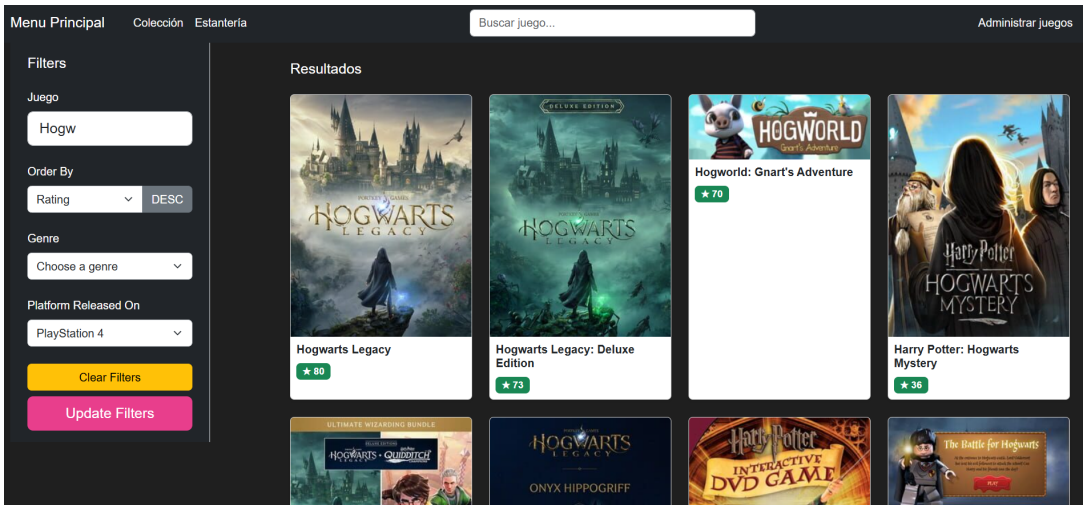
Al entrar en la página, se muestra el menú principal donde se muestran los juegos más populares actualmente. Como el objetivo es encontrar el juego de Hogwarts Legacy, se busca directamente en el buscador introduciendo "Hogw". En la pantalla de búsqueda no aparece el Título que se busca. Para ello se uso los filtros e introducimos que se ordene por *ranking* y en la plataforma de *playstation 4*. Tras aplicar el filtro, se puede ver que el juego que se buscaba aparece en la primera posición. Al entrar (Ver Figura 4.2), nos muestra un botón de añadir a la colección.



(a) Menú principal

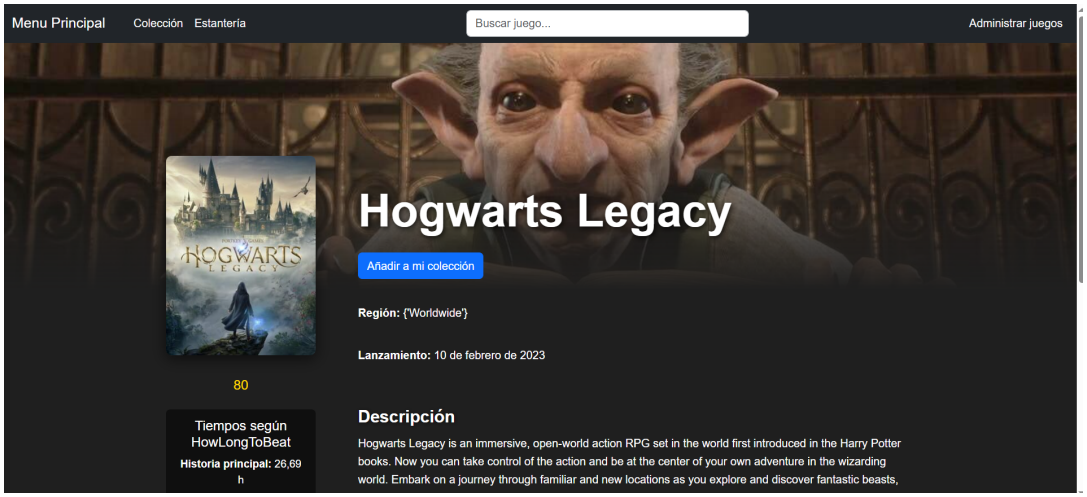


(b) Búsqueda inicial «Hogw»

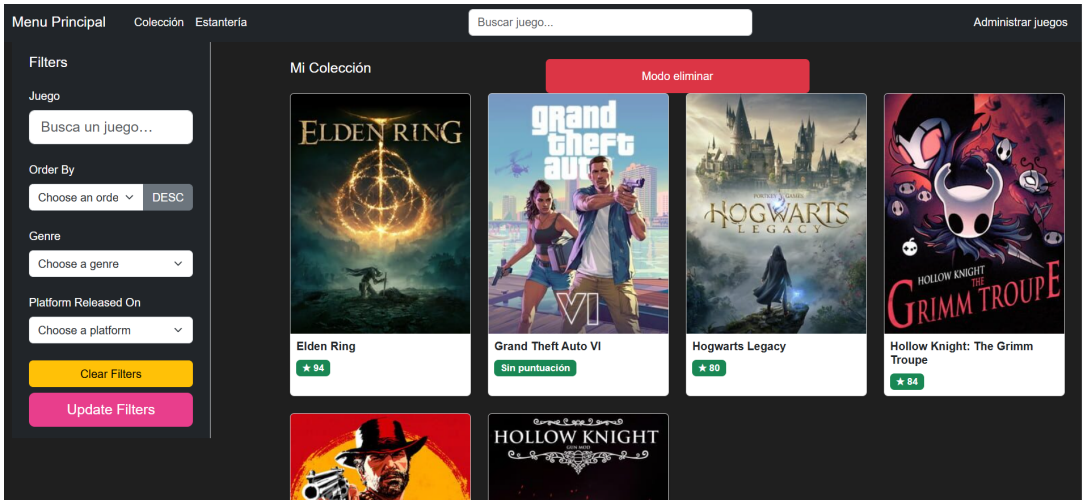


(c) Filtrado por plataforma y orden

Figura 4.1: Flujo la Tarea 1 - Búsqueda y filtrado



(a) Pantalla juego



(b) Pantalla de colección

Figura 4.2: Flujo la Tarea 1 - Añadir a Colección

4.1.2. Tarea 2: registro de notas y datos de coleccionismo

Una vez añadido *Hogwarts Legacy* a la colección, se encuentra disponible las opciones de añadir notas y datos de coleccionista. La Figura 4.3 muestra la apertura de la edición de notas y el guardado de datos de coleccionista.

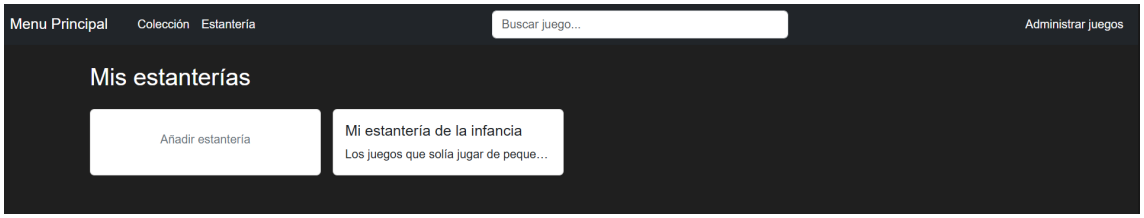
(a) Formulario de datos de notas

(b) Formulario de datos de coleccionismo

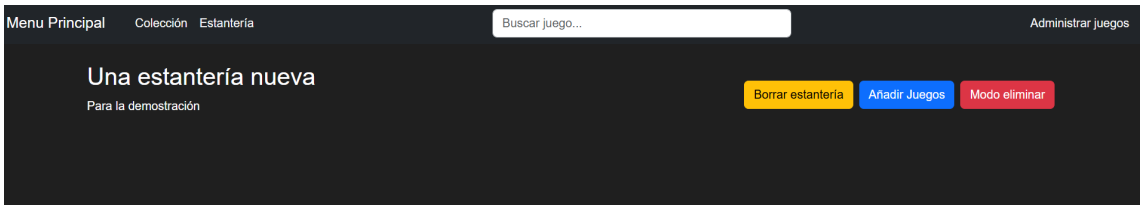
Figura 4.3: Flujo de la Tarea 2 – Registro de notas y datos de coleccionismo

4.1.3. Tarea 3: creación de estantería e inserción del juego

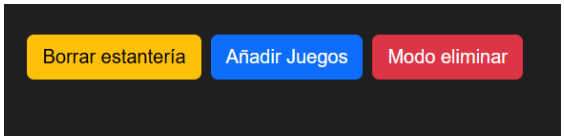
El siguiente paso consiste en organizar los títulos de la colección en estanterías. Primero se debe elegir/crear una estatería. A continuación se añade cualquier juego de la colección con el botón añadir juego. Tras ello, se guarda esta en la estantería. El proceso entero se aprecia en la Figura 4.4.



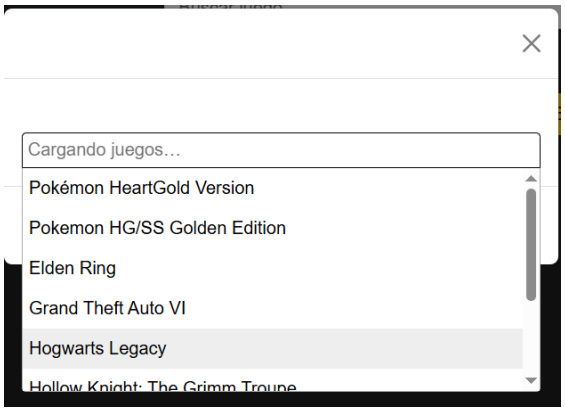
(a) Alta de estantería



(b) Estantería creada



(c) Botones de control de estantería

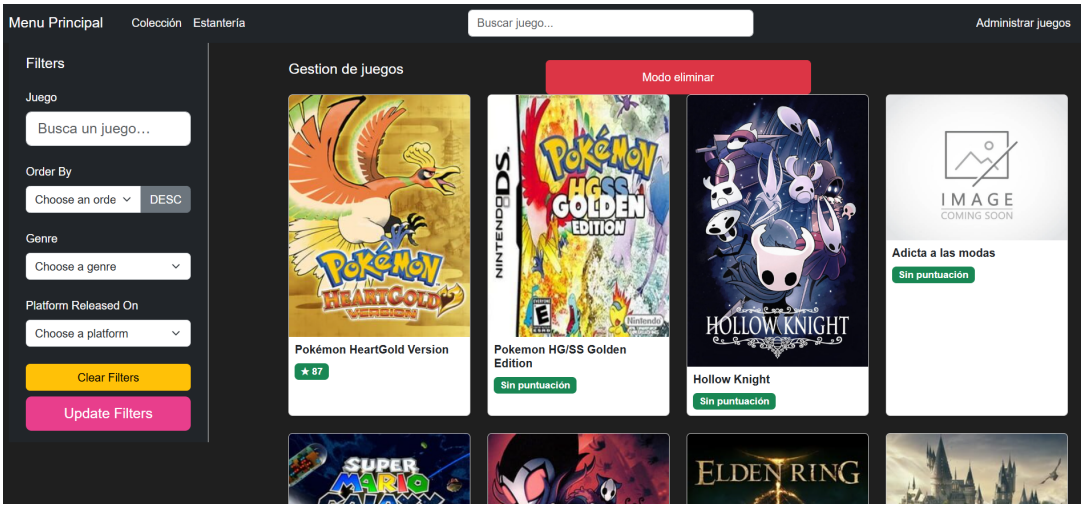


(d) Añadiendo el juego indicado

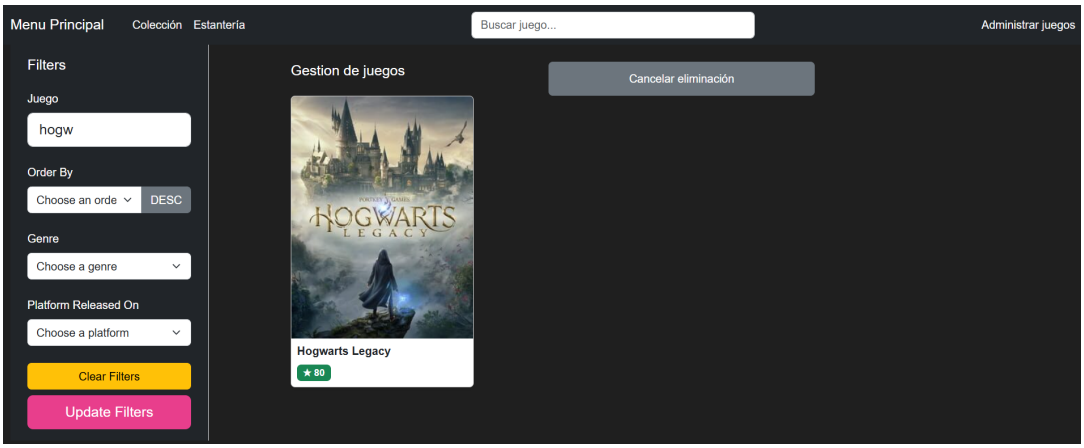
Figura 4.4: Flujo de la Tarea 3 – Creación de estantería y asignación del juego

4.1.4. Tarea 4: eliminación del juego de la base de datos

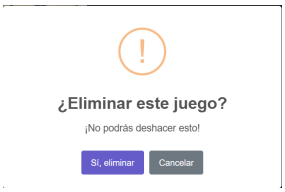
Por último, se muestra el procedimiento de eliminación de un videojuego en la base de datos. Para ello se accede a *Administrar juegos* en la parte superior derecha. Una vez dentro, se procede a buscar el juego a con los filtros y seguido activar el modo eliminar para eliminarlo de la base de datos. Todas las acciones de eliminación llevan alertas para prevenir borrados sin querer. La Figura 4.5 resume el flujo.



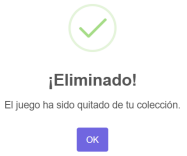
(a) Búsqueda del juego en el panel de administración



(b) Confirmación de borrado



(c) Aviso de acción de eliminación



(d) Aviso de eliminación completada

Figura 4.5: Flujo de la Tarea 4 – Eliminación de *Hogwarts Legacy*

4.2. Conclusiones

El trabajo de Fin de Grado ha terminado con el desarrollo de la aplicación web para gestionar videojuegos. Durante el proceso de creación de esta aplicación, me he sentido satisfecho con todas las nuevas tecnologías aprendidas. Entre los hitos conseguidos destacan:

La exitosa implementación de los métodos CRUD para la entidad de `Game`,

junto con la implementación de datos más personalizables como los de coleccionismo y la posibilidad de ordenarlos por estanterías.

La correcta integración de la API de IGDB junto con el webscraping de HLTB, que ha dado la posibilidad de poblar la base de datos de forma incremental. Resultando MariaDB una buena elección de sistema.

La correcta elección de la arquitectura MVC, demostrando ser suficiente para un proyecto con un desarrollador único.

En conclusión, el proyecto no sólo ha cumplido como objetivo académico sino que ha sentado una base sólida para seguir con el desarrollo de aplicaciones en relación a gestiones de bases de datos.

Aunque la aplicación final cumple con los objetivos establecidos en el comienzo del proyecto. Han habido ideas que surgieron durante la implementación de esta que puede servir como idea para futuros trabajos.

- Para empezar la ampliación e implementación de usuarios es sin duda uno de los objetivos futuros.
- Junto a ello aparece la posibilidad de visualizar los juegos guardados en las estanterías de otros usuarios, con el propósito de poder realizar intercambios de dichos juegos de forma virtual.
- Búsqueda de API con mayor concurrencia de usuarios, pues la usada en este proyecto solo permite una latencia de 4 llamadas por segundo.

Capítulo 5

Análisis de impacto

Gracias a este proyecto, he dado un paso firme como un desarrollador *full-stack*: al trabajar con Python y Django del back-end, plantillas y js en el front-end y MariaDB en la capa de base de datos.

Por otro lado, los impactos sociales del desarrollo de esta aplicación, influyen a las comunidades de jugadores que predominan el formato físico, disponiendo de una herramienta para ordenar, anotar y editar sus videojuegos con la misma comodidad que con los títulos virtuales.

Además que el sistema sienta como base para una futura aplicación de intercambio/venta de videojuegos físicos en el ambiente virtual, de modo que el proyecto podría evolucionar a un modelo de negocios especializado.

Bibliografía

[1] Asociación Española de Videojuegos (AEVI), *El sector del videojuego creció en 2024 con una facturación récord de más de 2 408 millones de euros y 22 millones de jugadores*, https://www.aevi.org.es/web/el-sector-del-videojuego-crecio-en-2024-con-una-facturacion-record-de-mas-de-2-408-millones-de-euros-y-22-millones-de-jugadores/, [Consultado: 20 May 2025], mayo de 2025.

[2] CrowdStrike. «CRUD Explained». (2025), dirección: https://www.crowdstrike.com/en-us/cybersecurity-101/observability/crud/ (visitado 02-07-2025).

[3] HowLongToBeat. «HowLongToBeat — Game Lengths Database». (2025), dirección: https://howlongtobeat.com/ (visitado 02-07-2025).

[4] Wikipedia contributors. «IGDB». (2025), dirección: https://en.wikipedia.org/wiki/IGDB (visitado 02-07-2025).

[5] Wikipedia contributors. «Modelo–vista–controlador». (2025), dirección: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador (visitado 02-07-2025).

[6] L. LLC. «Laravel — The PHP Framework for Web Artisans». (2025), dirección: https://laravel.com/ (visitado 02-07-2025).

[7] D. S. Foundation. «The Web framework for perfectionists with deadlines». (2025), dirección: https://www.djangoproject.com/ (visitado 02-07-2025).

[8] H. México. «MySQL: qué es y qué ventajas tiene». (2024), dirección: https://www.hostgator.mx/blog/mysql-conoce-que-es-y-que-ventajas-tiene/ (visitado 02-07-2025).

[9] Wikipedia contributors. «API». (2025), dirección: https://es.wikipedia.org/wiki/API (visitado 02-07-2025).

[10] W. Bastidas. «Cómo llamar a una API REST desde JavaScript». (2023), dirección: https://medium.com/williambastidasblog/c%C3%B3mo-llamar-a-una-api-rest-desde-javascript-4c5a42fb331 (visitado 02-07-2025).

[11] Wikipedia contributors. «Website wireframe». (2025), dirección: https://en.wikipedia.org/wiki/Website_wireframe (visitado 02-07-2025).

[12] A. W. Services. «MySQL vs. PostgreSQL: key differences». (2024), dirección: https://aws.amazon.com/es/compare/the-difference-between-mysql-vs-postgresql/ (visitado 02-07-2025).

[13] HowLongToBeatPy contributors. «HowLongToBeatPy – Unofficial Python API wrapper for HowLongToBeat». (2025), dirección: https://github.com/andreis13/howlongtobeatpy (visitado 02-07-2025).

Anexos

Apéndice A

Repositorio de github

El código del proyecto está en el siguiente repositorio de Github:

`https://github.com/jinxand159/VideoGame-Admin-web.git`

Apéndice B

Código de utilidades

En este anexo se incluye el contenido de `utilidades.py`, que engloba la función `fetch_igdb` y la recogida de datos de HLTB:

```
from __future__ import annotations

from typing import Any, Dict, List, Optional, Tuple
import requests
from django.conf import settings
from howlongtobeatpy import HowLongToBeat

from django.http import JsonResponse, HttpResponseBadRequest
from django.apps import apps
from django.views import View
from games.models import *

IGDB_API_URL = "https://api.igdb.com/v4"


class IGDBError(Exception):
    """Custom exception for IGDB API errors."""
    pass


def fetch_igdb(
    endpoint: str,
    fields: List[str] = ["name"],
    limit: int = 100,
    offset: int = 0,
    search: Optional[str] = None,
    filters: Optional[str] = None,
    sort: Optional[str] = None,
) -> List[Dict[str, Any]]:
```

```
    url = f"{IGDB_API_URL}/{endpoint}"
    headers = {
        "Client-ID": settings.CLIENT_ID,
        "Authorization": f"Bearer {settings.ACCESS_TOKEN}",
    }

    if "id" not in fields:
        fields = ["id"] + fields

    clauses: List[str] = []
    if search:
        clauses.append(f'search "{search}";')
    if filters:
        f = filters.strip()
        if not f.lower().startswith("where"):
            f = f"where {f}"
        if not f.endswith(";"):
            f += ";"
        clauses.append(f)
    if sort:
        s = sort.strip()
        if not s.lower().startswith("sort"):
            s = f"sort {s}"
        if not s.endswith(";"):
            s += ";"
        clauses.append(s)

    clauses.append(f"fields {', '.join(fields)};")
    if offset > 0:
        clauses.append(f"offset {offset};")
    clauses.append(f"limit {limit};")
    body = " ".join(clauses)
    try:
        resp = requests.post(url, headers=headers, data=body, timeout=10)
        if resp.status_code == 401:
            raise IGDBError(
                "Unauthorized: Check your Client-ID and Access Token.")
        resp.raise_for_status()
        print(resp.json())
        return resp.json()
    except requests.RequestException as e:
        raise IGDBError(f"Error fetching {endpoint}: {e}")


def get_region_map() -> Dict[int, str]:
    return {
        0:  "",
```

```
        1:  "Europe",
        2:  "North America",
        3:  "Australia",
        4:  "New Zealand",
        5:  "Japan",
        6:  "China",
        7:  "Asia",
        8:  "Worldwide",
        9:  "Korea",
        10: "Brazil",
    }


def fetch_hltb_minutes(title: str) -> int | None:
    """
    Devuelve la duración principal del juego (en minutos)
    o None si no lo encuentra.
    """
    try:
        resultados = HowLongToBeat().search(title)
        if not resultados:
            return {
                "hltb_main": None,
                "hltb_main_extra": None,
                "hltb_completionist": None,
                "hltb_rush": None,
            }
        encontrado = resultados[0]
        return {
            "hltb_main": getattr(encontrado, "main_story", None),
            "hltb_main_extra": getattr(encontrado, "main_extra", None),
            "hltb_completionist": getattr(encontrado, "completionist", Non
        }
    except Exception:
        # En caso de fallo (timeout, etc.), devolvemos None para todo
        return {
            "hltb_main": None,
            "hltb_main_extra": None,
            "hltb_completionist": None,
            "hltb_rush": None,
        }
```

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 22:58:13 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)