

Trabajo de Fin de Ciclo

Desarrollo de Aplicaciones Web

TestPlay

Gestión de alquiler de videojuegos



Autor: Carlos Moraleda Ruiz

12-06-2025

ÍNDICE

I. Introducción.....	3
a. Presentación del proyecto.....	3
b. Objetivos del proyecto.....	3
c. Justificación del proyecto.....	4
II. Análisis de requerimientos.....	5
a. Identificación de necesidades y requerimientos.....	5
b. Identificación de público.....	6
c. Estudio de mercado y competencia.....	6
III. Diseño y planificación.....	8
a. Definición de la arquitectura del proyecto.....	8
b. Diseño de la interfaz de usuario.....	9
c. Planificación de las tareas y los recursos necesarios.....	18
IV. Implementación y pruebas.....	20
a. Desarrollo de las funcionalidades del proyecto.....	20
b. Pruebas unitarias y de integración.....	20
c. Corrección de errores y optimización del rendimiento.....	21
V. Documentación.....	22
a. Documentación técnica.....	22
b. Documentación de usuario.....	49
c. Manual de instalación y configuración.....	50
VI. Mantenimiento y evolución.....	53
a. Plan de mantenimiento y soporte.....	53
b. Identificación de posibles mejoras y evolución del proyecto.....	53
c. Actualizaciones y mejoras futuras.....	54
VII. Conclusiones.....	55
a. Evaluación del proyecto.....	55
b. Cumplimiento de objetivos y requisitos.....	55
c. Lecciones aprendidas y recomendaciones para futuros proyectos.....	56
VIII. Bibliografía y referencias.....	57
a. Fuentes utilizadas en el proyecto.....	57
b. Referencias y enlaces de interés.....	57

I. Introducción

a. Presentación del proyecto

TestPlay es una plataforma web desarrollada para gestionar el alquiler temporal de videojuegos en formato digital, orientada exclusivamente a títulos de PC. El sistema está pensado para ofrecer una experiencia sencilla tanto a usuarios como a administradores, permitiendo la reserva, préstamo y gestión de videojuegos mediante una interfaz moderna e intuitiva.

Además de facilitar el alquiler, TestPlay incluye funcionalidades como puntuaciones, comentarios, historial de juegos probados y control administrativo del catálogo y usuarios.

b. Objetivos del proyecto

Objetivo general:

Desarrollar una plataforma web funcional y escalable para la gestión de alquiler de videojuegos digitales para PC, ofreciendo una experiencia eficiente y atractiva tanto para usuarios como para administradores.

Objetivos específicos:

- Permitir a los usuarios registrarse, iniciar sesión y alquilar videojuegos por tiempo limitado.
- Integrar un sistema de comentarios, videojuegos probados y puntuaciones por parte de los usuarios.
- Gestionar automáticamente los comentarios, puntuaciones, videojuegos probados y reservas mediante una interfaz dinámica con AJAX.
- Brindar a los administradores herramientas para el control del catálogo, usuarios, y configuración del sistema.
- Garantizar la seguridad, integridad y trazabilidad de los datos de usuarios y videojuegos.

c. Justificación del proyecto

El mercado de los videojuegos ha experimentado un crecimiento constante en los últimos años, especialmente en su distribución digital. Sin embargo, no todos los usuarios desean o pueden adquirir cada juego a precio completo, y muchos buscan probarlos antes de realizar una compra definitiva.

TestPlay surge como una solución innovadora para cubrir esta necesidad, ofreciendo un sistema de alquiler temporal de juegos en versión PC. Esta propuesta permite al usuario acceder a una gran variedad de títulos de forma económica, al tiempo que proporciona a los desarrolladores y editores una forma alternativa de distribuir y monetizar sus contenidos. Además, el sistema sirve como espacio de interacción, donde los usuarios pueden comentar y valorar los juegos, enriqueciendo la experiencia de toda la comunidad.

II. Análisis de requerimientos

a. Identificación de necesidades y requerimientos

El desarrollo de **TestPlay** parte de la necesidad de ofrecer una plataforma que permita el acceso temporal a videojuegos de PC, facilitando a los usuarios la posibilidad de probar juegos antes de realizar una compra definitiva o simplemente disfrutar de títulos sin necesidad de adquirirlos de forma permanente. Para ello, se identificaron los siguientes requerimientos:

Necesidades funcionales:

- Registro y autenticación de usuarios.
- Navegación y búsqueda de videojuegos por título o categoría.
- Posibilidad de reservar, alquilar(prestar) y devolver videojuegos.
- Sistema para marcar videojuegos como “probados”.
- Sistema de puntuaciones y comentarios.
- Visualización del historial de juegos alquilados(prestados), reservados y “probados”.
- Interfaz administrativa para gestionar usuarios, videojuegos, préstamos y configuraciones.

Necesidades no funcionales:

- Interfaz clara, responsive e intuitiva.
- Acceso multiusuario con roles diferenciados (usuario y administrador).
- Interacción fluida sin recargas completas de página (uso de AJAX).
- Seguridad en el tratamiento de datos y validación de formularios.
- Escalabilidad para permitir el crecimiento del catálogo y usuarios.

b. Identificación de público

Público objetivo principal:

- **Jugadores de PC**, principalmente adolescentes y adultos jóvenes (de 16 a 35 años), interesados en probar videojuegos antes de comprarlos o en jugar por tiempo limitado.
- Usuarios que buscan alternativas económicas al modelo tradicional de compra de videojuegos.

Público secundario:

- **Desarrolladores o estudios independientes**, que pueden beneficiarse al distribuir versiones de prueba o títulos menos conocidos.
- **Administradores de la plataforma**, encargados de gestionar el catálogo, reservas, préstamos, y controlar el funcionamiento del sistema.

c. Estudio de mercado y competencia

En el mercado actual, predominan plataformas de distribución de videojuegos que ofrecen compra definitiva o suscripciones mensuales (como Steam, Epic Games, Xbox Game Pass o PlayStation Plus). Sin embargo, el modelo de **alquiler temporal** de juegos digitales es mucho menos frecuente, especialmente en plataformas centradas exclusivamente en PC.

Plataformas similares o competidoras:

- **Xbox Game Pass / PlayStation Plus**: ofrecen acceso a un catálogo extenso por suscripción, pero no por alquiler individual ni en PC puro.
- **Steam**: permite probar juegos solo en demos o durante eventos específicos, sin sistema de alquiler.
- **Utomik / Blacknut**: servicios de suscripción en la nube con acceso a juegos por tiempo limitado, pero centrados en streaming y no en descargas.

Ventaja competitiva de TestPlay:

- Modelo de alquiler individual sin necesidad de suscripción.
- Sistema de puntuaciones, comentarios y reseñas colaborativas.
- Interfaz web accesible y ligera, sin necesidad de software adicional.
- Posibilidad de probar juegos concretos por demanda del usuario.

III. Diseño y planificación

a. Definición de la arquitectura del proyecto

El sistema **TestPlay** se ha construido utilizando el patrón **MVC (Modelo-Vista-Controlador)**, lo cual permite una separación clara entre la lógica de negocio, la interfaz de usuario y la gestión de datos. Esto facilita la escalabilidad, el mantenimiento y la reutilización del código.

Tecnologías utilizadas:

- **Backend:** PHP (estructura MVC)
- **Frontend:** HTML, CSS, Bootstrap, JavaScript (con uso de AJAX)
- **Base de datos:** MySQL
- **Librerías y herramientas adicionales:**
 - SweetAlert (para notificaciones y confirmaciones)
 - jQuery (para facilitar llamadas AJAX y manipulación del DOM)
 - Bootstrap (framework para estilos CSS)

Entorno de desarrollo:

- El proyecto ha sido desarrollado usando Visual Studio Code como entorno de programación, dentro de una máquina virtual con Linux Mint.
- Para pruebas locales, se utilizó un entorno configurado manualmente con servidor Apache y base de datos MySQL en Linux.
- Adicionalmente, se ha probado y verificado el funcionamiento del proyecto en Windows 11 utilizando XAMPP, que integra Apache, MySQL y PHP, lo cual permite una configuración rápida para desarrollo y testing.

Estructura general:

- **Modelos:** Manejan la lógica de acceso a datos y representan las entidades principales (usuarios, videojuegos, reservas, comentarios, etc.) que serían las clases.
- **Controladores:** Se encargan de procesar las peticiones, coordinar la lógica del modelo y decidir qué vista mostrar.
- **Vistas:** Encargadas de la presentación de la información al usuario. Contienen HTML y scripts JS que permiten la interacción sin recargas completas.
- **DAO (Data Access Object):** Capa adicional encargada de realizar consultas SQL y aislar el acceso a la base de datos(estos archivos están localizados dentro del directorio “modelos”).
- **AJAX:** Empleado en funcionalidades clave como reservas, puntuaciones, marcar como probado y comentarios, mejorando la experiencia sin recargar páginas.
- **Mapa de enrutamiento:** Se encuentra centralizado en el archivo principal index.php, el cual recibe las peticiones y se encarga de redirigirlas al controlador y acción correspondiente, funcionando como punto de entrada del sistema.

b. Diseño de la interfaz de usuario

El diseño de la interfaz busca ser **intuitivo, limpio y responsive**, adaptándose a diferentes tamaños de pantalla y facilitando la navegación tanto a usuarios como a administradores.

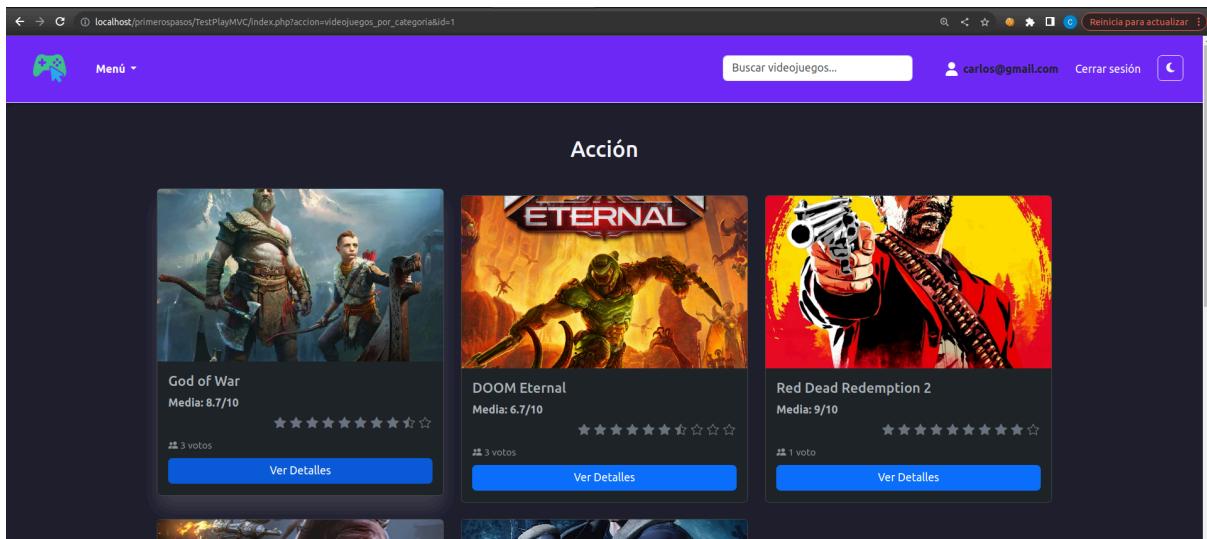
Elementos principales de la interfaz:

- **Cabecera/Header:** Uno de los elementos principales es la **cabecera**, presente en todas las vistas del sitio. Esta incluye:
 - Un **ícono de inicio** para volver a la página principal.

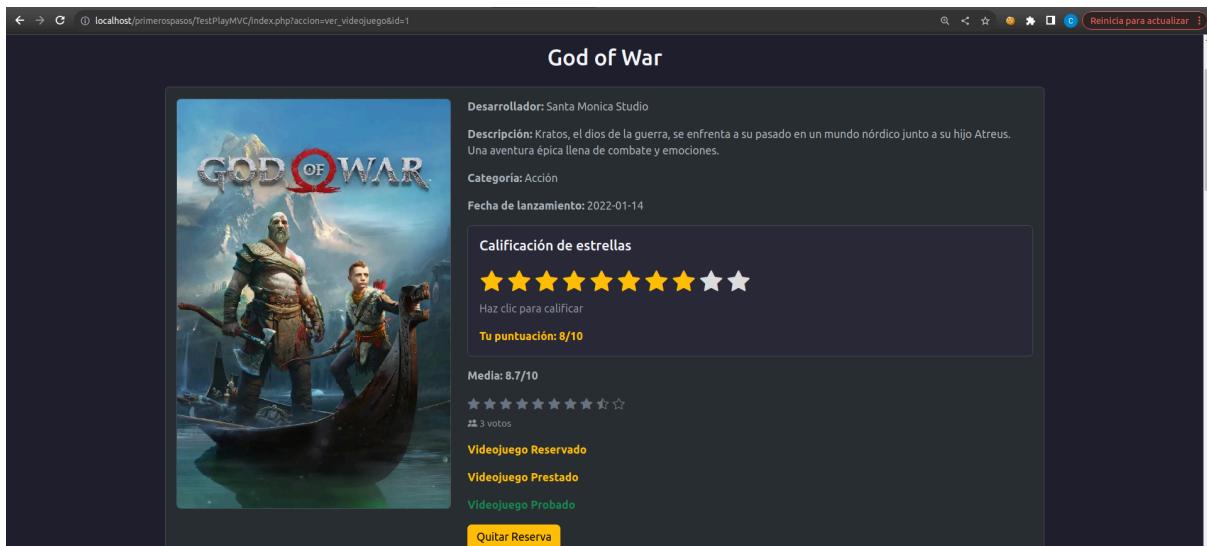
- Una **barra de búsqueda** que permite localizar videojuegos por su título de forma rápida.
 - En caso de sesión iniciada, se muestra el **email del usuario** junto con un ícono representativo del rol: un **usuario** para usuarios comunes o un **escudo** para administradores.
 - Un **botón de cambio de tema** que permite alternar entre modo claro y oscuro, usando los iconos de **sol** y **luna**.
 - Si no hay sesión activa, la cabecera muestra directamente un **formulario de inicio de sesión** (correo y contraseña) y un botón para ir al **registro**.
- **Página de inicio:** Muestra la navegación por categorías.

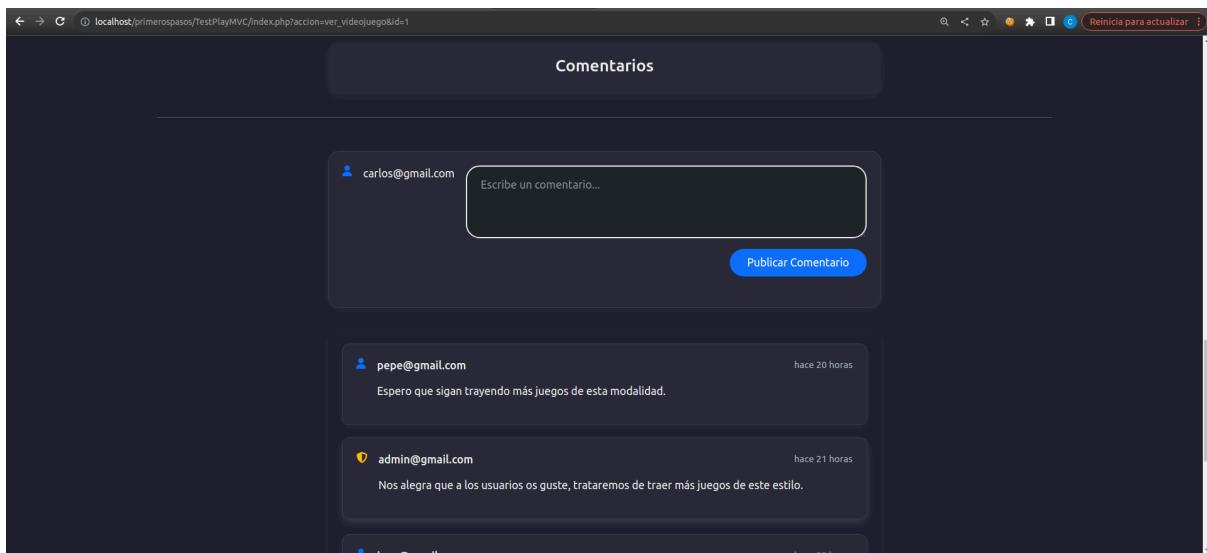
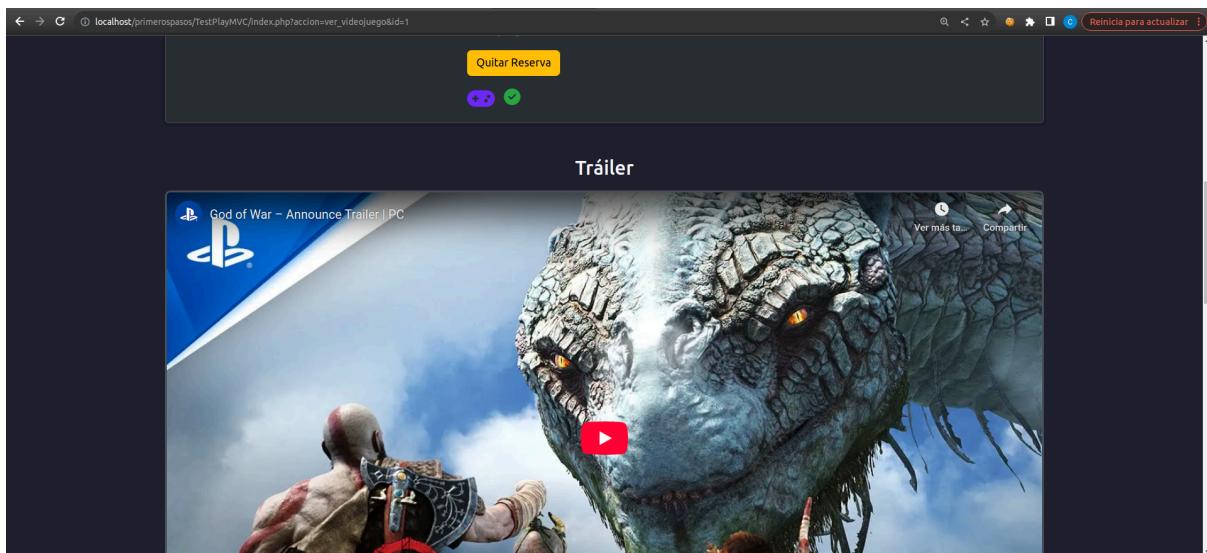
The screenshots illustrate the TestPlay website's design and functionality. The top screenshot shows the site's landing page with a purple header, a central banner with the text 'Bienvenido a TestPlay' and '¡Explora, alquila y juega!', and a footer with social media icons and copyright information. The bottom screenshot shows the same landing page but with a user session active, featuring a user profile icon, a search bar, and a 'Cerrar sesión' button in the header.

- **Listado de videojuegos:** Incluye los títulos y la puntuación media de cada videojuego con un botón para acceder a la vista individual del videojuego seleccionado.



- **Vista individual de videojuego:** Muestra detalles, puntuación media, comentarios, información sobre la disponibilidad de la reserva y el préstamo además de botones dinámicos para reservar, puntuar, comentar(formulario para publicar comentario) y marcar como probado.





- **Zona de usuario:** Historial de juegos prestados, reservados y probados.

Videojuego	Fecha del Préstamo	Estado
DOOM Eternal	2025-03-09 16:11:10	Videojuego devuelto
God of War	2025-03-09 11:07:07	Videojuego devuelto
Frostpunk	2025-03-17 18:42:24	Videojuego devuelto

Aquí se muestra el historial de juegos prestados, donde el videojuego puede ser devuelto o no, ya que puede haber estado prestado de otras ocasiones. Para las reservas y los videojuegos probados sería lo mismo pero sin tener el detalle del videojuego reservado o probado sólo se ven la imagen, el título y la fecha, y no se repiten como sí es en este caso porque se puede haber hecho el préstamo en otro momento.

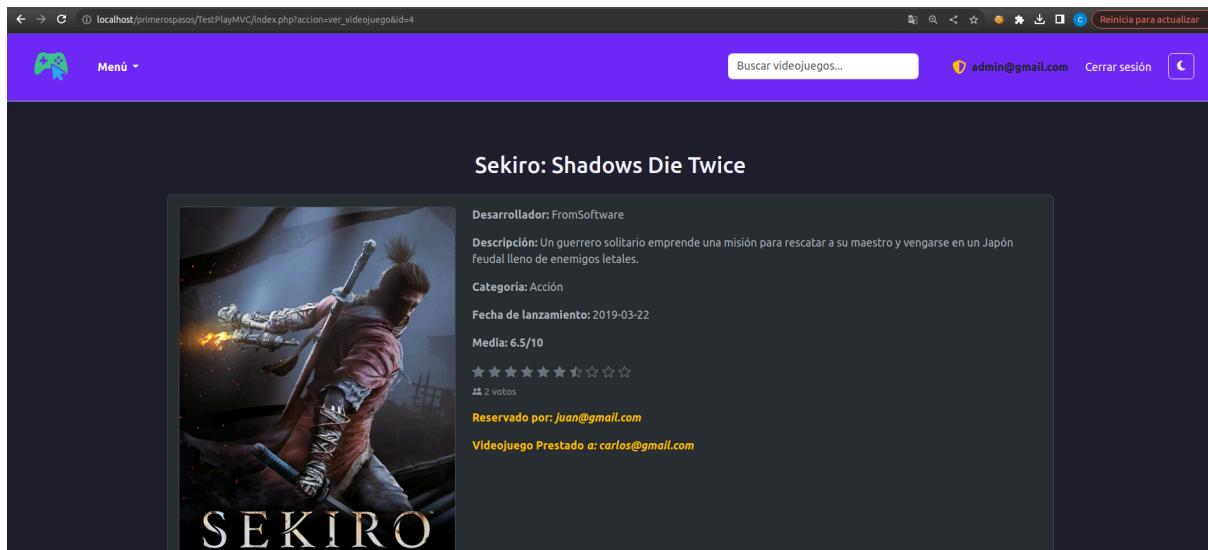
- **Zona de administrador:** Historial de todos los juegos prestados, reservados y probados por usuario.

Usuario	Número de Préstamos
jose@gmail.com	6 préstamos
pepe@gmail.com	2 préstamos

Videojuego	Fecha del préstamo	Estado
Cyberpunk 2077	2025-06-04 01:24:22	Videojuego NO devuelto
Life is Strange: True Colors	2025-03-18 10:08:06	Videojuego devuelto

Aquí se muestra el historial de todos los videojuegos prestados por usuario, donde el videojuego puede ser devuelto o no, ya que puede haber estado prestado de otras ocasiones. Para las reservas y los videojuegos probados

sería lo mismo pero sin tener el detalle del videojuego reservado o probado sólo se ven la imagen, el título y la fecha, y no se repiten como sí es en este caso porque se puede haber hecho el préstamo en otro momento.



Cabe destacar que para el administrador en la vista individual del videojuego puede visualizar a quién está prestado el videojuego en ese momento y por quién se ha reservado en ese momento el videojuego, o si están disponibles.

- **Panel de administración:** Accesible solo por administradores, incluye gestión de videojuegos, usuarios, reservas, préstamos y configuración del sistema.

The screenshot displays the administrator interface with the following sections:

- Configuraciones de Videojuegos:** Shows a table of video games with columns for Nombre (Name) and Descripción (Description).
- Tabla de usuarios:** Shows a table of users with columns for Usuario (User) and Rol (Role). The data is as follows:

Usuario	Rol
carlos@gmail.com	Usuario
paco@gmail.com	Usuario
jose@gmail.com	Usuario
pepe@gmail.com	Usuario
juan@gmail.com	Usuario
admin@gmail.com	Administrador
charlie3-0@testplay.com	Administrador

- Configuración de roles:** A form to change a user's role. It has fields for 'Usuario:' (set to carlos@gmail.com) and 'Rol:' (set to 'Usuario'), and a 'Guardar' (Save) button.

localhost/primerospasos/TestPlayMVC/index.php?accion=configuraciones_videojuegos

Tabla de Videojuegos

Título	Acciones			
God of War	Editar	Eliminar	Quitar Reserva	Devolver
DOOM Eternal	Editar	Eliminar	Quitar Reserva	Devolver
Red Dead Redemption 2	Editar	Eliminar	Quitar Reserva	Préstamo Disponible
Sekiro: Shadows Die Twice	Editar	Eliminar	Quitar Reserva	Devolver
Resident Evil 4 Remake	Editar	Eliminar	Reserva Disponible	Devolver
Alan Wake 2	Editar	Eliminar	Quitar Reserva	Préstamo Disponible
A Plague Tale: Requiem	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Life is Strange: True Colors	Editar	Eliminar	Quitar Reserva	Préstamo Disponible
Stray	Editar	Eliminar	Quitar Reserva	Devolver
Uncharted: Legacy of Thieves Collection	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Age of Empires IV	Editar	Eliminar	Quitar Reserva	Préstamo Disponible

localhost/primerospasos/TestPlayMVC/index.php?accion=configuraciones_videojuegos

Tabla de Videojuegos

Título	Acciones			
The Witness	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Humanity	Editar	Eliminar	Quitar Reserva	Préstamo Disponible
Superliminal	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Portal 2	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Sons of the Forest	Editar	Eliminar	Quitar Reserva	Préstamo Disponible
Green Hell	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Subnautica: Below Zero	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
Valheim	Editar	Eliminar	Reserva Disponible	Préstamo Disponible
The Long Dark	Editar	Eliminar	Reserva Disponible	Préstamo Disponible

[+ Añadir Videojuego](#) [+ Añadir Préstamo](#)

Sobre TestPlay
Alquiler temporal de videojuegos para PC. Explora títulos únicos y conocidos.

© 2025 TestPlay. Todos los derechos reservados.

- **Sobre nosotros:** Área “Sobre Mí” donde se muestra una carta de presentación y un currículum.

The screenshot shows a dark-themed web page with a purple header bar. The header includes a logo, a 'Menú' button, a search bar ('Buscar videojuegos...'), and user information ('admin@gmail.com' and 'Cerrar sesión'). Below the header, there are two main sections: 'Carta de Presentación' (Presentation Letter) and 'Currículum Vitae' (Curriculum Vitae). The presentation letter contains a message from Carlos Moraleda about his background and the project. The curriculum vitae lists personal data: Nombre: Carlos Moraleda, Email: carlosmoraleda96@gmail.com, and Ubicación: Villacañas (Toledo), España.

- **Añadir Préstamo:** En esta página nos encontramos con un formulario que recogerá los datos del usuario(email) y del videojuego seleccionados para insertar un préstamo, se podrá seleccionar con el select o bien buscar en su barra de búsqueda correspondiente escribiendo, esto es gracias a una tecnología de bootstrap que aplica JQuery(se usa Select2, importando el script de JQuery, el script de Select2 y el css de Select2 de bootstrap).

The screenshot shows a dark-themed web page with a purple header bar. The header includes a logo, a 'Menú' button, a search bar ('Buscar videojuegos...'), and user information ('admin@gmail.com' and 'Cerrar sesión'). The main content area is titled 'Añadir Préstamo' (Add Loan). It features two dropdown menus: 'Seleccionar Usuario' (Select User) with 'carlos@gmail.com' selected, and 'Seleccionar Videojuego' (Select Game). A search input field contains 'g' and a dropdown list shows game titles starting with 'God of War'. At the bottom, there is a footer with social media links (Facebook, X, Instagram, YouTube) and copyright information: 'Sobre TestPlay' and 'Alquiler temporal de videojuegos para PC. Explora títulos únicos y conocidos.' followed by '© 2025 TestPlay. Todos los derechos reservados.'

- **Registro:** En esta página nos encontramos con un formulario que recogerá los datos del nuevo usuario(por defecto con rol “U” que sería de usuario normal) que se almacenará en nuestra base de datos para después hacer su correspondiente inicio de sesión.

localhost/primerospasos/testPlayMVC/index.php?accion=registrar

Reinicia para actualizar

Email Password Login Registrarse

Registro de Usuario

Correo electrónico
ejemplo@correo.com

Contraseña
Minimo 4 caracteres

Registrar

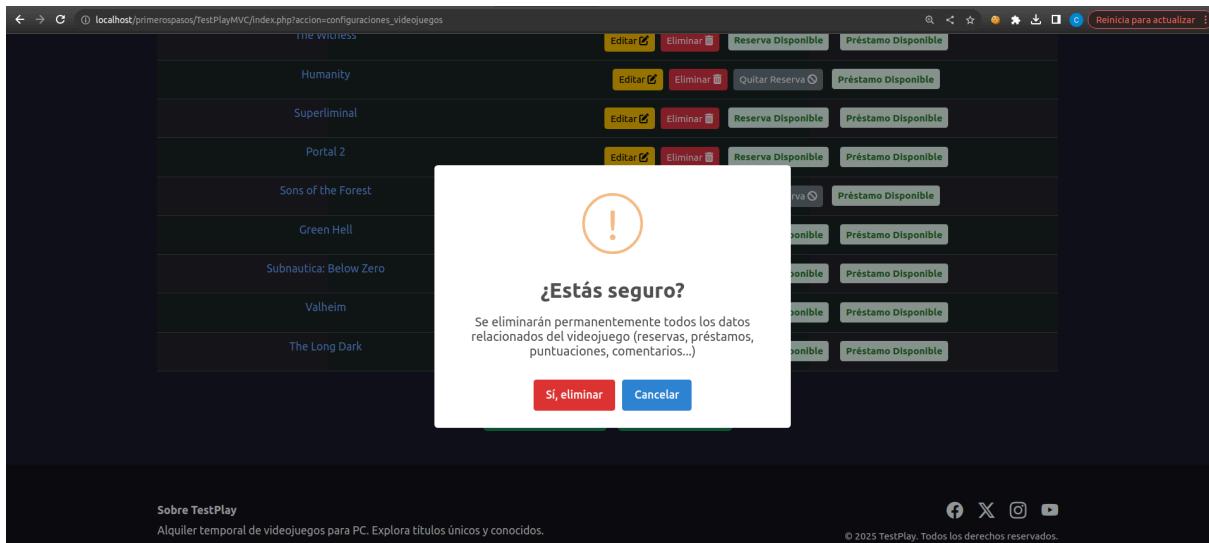
Volver al Inicio

Sobre TestPlay
Alquiler temporal de videojuegos para PC. Explora títulos únicos y conocidos.

© 2025 TestPlay. Todos los derechos reservados.

Los otros formularios siguen los mismos estilos que el de registro(insertar videojuego y editar videojuego).

- **Diseño visual:** Estilo moderno, botones claros, mensajes contextuales y retroalimentación inmediata mediante AJAX y alertas.



Por ejemplo si hacemos click en eliminar nos aparecerá un alert para confirmar la acción e informarnos de lo que ocurrirá si la ejecutamos al eliminar el videojuego.

c. Planificación de las tareas y los recursos necesarios

Durante el desarrollo del proyecto se definieron las siguientes etapas y tareas principales:

1. Análisis y diseño:

- Definición de requisitos funcionales y no funcionales.
- Diseño de la base de datos relacional.
- Estructura de carpetas y arquitectura MVC.

2. Desarrollo del backend:

- Implementación de modelos, controladores y DAOs.
- Configuración de sesiones, autenticación y roles.
- Lógica para reservas, préstamos, videojuegos probados, puntuaciones, comentarios, historial y gestión de videojuegos.

3. Desarrollo del frontend:

- Diseño de vistas con HTML, CSS y BootStrap.
- Implementación de formularios, botones y vistas condicionales por rol.
- Programación de interactividad con JavaScript y AJAX.

4. Pruebas:

- Pruebas unitarias y funcionales.
- Validación de formularios y flujos críticos (registro, préstamo, devolución, videojuegos, comentarios).
- Simulación de errores y carga.

Recursos necesarios:

- Un servidor local para desarrollo (XAMPP o similar).
- Navegadores para pruebas de compatibilidad.
- Editor de código (Visual Studio Code u otro).
- Base de datos MySQL.
- Repositorio de control de versiones (Git).

IV. Implementación y pruebas

a. Desarrollo de las funcionalidades del proyecto

El desarrollo del sistema **TestPlay** se realizó siguiendo una estructura modular basada en el patrón **MVC**, permitiendo organizar el código en controladores, modelos, vistas y DAOs. Las funcionalidades se implementaron de forma progresiva, priorizando la experiencia del usuario y el cumplimiento de los requisitos:

- Registro e inicio de sesión de usuarios.
- Sistema de roles: usuario normal ('U') y administrador ('A').
- Listado de videojuegos con búsqueda por título.
- Sistema de reservas con AJAX.
- Marcado de videojuegos como “probados”.
- Valoración mediante puntuaciones del 1 al 10.
- Sistema de comentarios con edición y eliminación (según rol).
- Panel de administración para gestión de usuarios y videojuegos.
- Modo oscuro/claro configurable desde la cabecera.
- Visualización de historiales por usuario y agrupados por administrador.

Cada funcionalidad se validó de forma individual tras su implementación para asegurar su correcto funcionamiento antes de integrar con otras partes del sistema.

b. Pruebas unitarias y de integración

Durante el desarrollo se realizaron **pruebas manuales** de cada módulo, simulando los distintos roles de usuario y estados del sistema. Se comprobaron acciones como reservar, puntuar, comentar, modificar datos y gestionar contenidos.

Se validaron también los **mensajes de error y confirmación**, así como el comportamiento dinámico con **AJAX**, asegurando que el contenido se actualizara correctamente sin recargar la página.

La integración entre vistas, controladores y modelos se revisó exhaustivamente para garantizar la coherencia de datos en toda la aplicación.

c. Corrección de errores y optimización del rendimiento

A lo largo del desarrollo se detectaron y solucionaron errores relacionados con validaciones de formularios, duplicación de acciones AJAX, control de sesiones y restricciones de acceso por rol.

Se mejoró el rendimiento mediante:

- Uso de **consultas SQL optimizadas** y control de índices en la base de datos.
- Minimización de recargas de página usando AJAX.
- Reutilización de vistas y componentes comunes.
- Separación de lógica del servidor y del cliente para una carga más ágil.

Gracias a este enfoque iterativo de implementación, prueba y mejora, el sistema se fue consolidando de forma estable y eficiente.

V. Documentación

a. Documentación técnica

La documentación técnica del proyecto TestPlay detalla la estructura del código, la base de datos y los componentes principales del sistema. Se organiza siguiendo el patrón **MVC**, e incluye:

- Estructura de archivos y carpetas (controladores, modelos, DAOs, vistas, JS, CSS...):

```
alumno@alumno-VirtualBox-DWES:/var/www/html/primerospasos/TestPlayMVC$ tree
.
└── app
    ├── config
    │   └── config.php
    ├── controladores
    │   ├── ControladorComentarios.php
    │   ├── ControladorPrestamos.php
    │   ├── ControladorPuntuaciones.php
    │   ├── ControladorReservas.php
    │   ├── ControladorUsuarios.php
    │   ├── ControladorVideojuegos.php
    │   └── ControladorVideojuegosProbados.php
    ├── modelos
    │   ├── Categoria.php
    │   ├── CategoriasDAO.php
    │   ├── Comentario.php
    │   ├── ComentariosDAO.php
    │   ├── ConnexionDB.php
    │   ├── Prestamo.php
    │   ├── PrestamosDAO.php
    │   ├── PuntuacionesDAO.php
    │   ├── Puntuacion.php
    │   ├── Reserva.php
    │   ├── ReservasDAO.php
    │   ├── Sesion.php
    │   ├── Usuario.php
    │   ├── UsuariosDAO.php
    │   ├── Videojuego.php
    │   ├── Videojuego_probado.php
    │   ├── VideojuegosDAO.php
    │   └── Videojuegos_probadosDAO.php
    └── utils
        └── funciones.php
    └── vistas
        ├── configuraciones.php
        ├── editar_videojuego.php
        ├── inicio.php
        ├── insertar_prestamo.php
        ├── insertar_videojuego.php
        ├── registrar.php
        ├── sobre_nosotros.php
        ├── ver_prestados.php
        ├── ver_reservas.php
        ├── ver_todas_reservas.php
        ├── ver.todos_prestamos.php
        ├── ver.todos_videojuegos_probados.php
        ├── ver.videojuego.php
        └── ver.videojuegos_probados.php
```

```

└── útiles
    └── funciones.php
vistas
    ├── configuraciones.php
    ├── editar_videojuego.php
    ├── inicio.php
    ├── insertar_prestamo.php
    ├── insertar_videojuego.php
    ├── registrar.php
    ├── sobre_nosotros.php
    ├── ver_prestados.php
    ├── ver_reservas.php
    ├── ver_todas_reservas.php
    ├── ver.todos_prestamos.php
    ├── ver.todos_videojuegos_probados.php
    ├── ver_videojuego.php
    ├── ver_videojuegos_probados.php
    └── videojuegos_por_categoria.php
index.php
js.js
README.md
└── web
    ├── CSS
    │   └── estilos.css
    ├── icons
    │   ├── Logo_TestPlay_2.png
    │   ├── Logo_TestPlay.png
    │   └── TestPlay-icon.png
    └── Images
        ├── Alicia_en_pais_de_maravillas-large.jpg
        ├── animal.jpg
        ├── avatar.jpg
        └── Blade_Runner_2049.jpg

```

- Explicación del **mapa de enrutamiento** definido en index.php, que interpreta las peticiones entrantes y carga los controladores y vistas correspondientes:

Se inicia la sesión con session_start() para poder utilizar variables de sesión.

Se incluyen los archivos necesarios, como la configuración, controladores y modelos.

Se realiza el parseo de la ruta para determinar la acción a ejecutar. Si no se proporciona una acción en la URL, se establece la acción por defecto como "inicio".

Se realiza un control de acceso basado en variables de sesión.

Finalmente, se ejecuta el controlador y el método correspondiente. Se crea una instancia del controlador necesario y se invoca el método correspondiente utilizando la sintaxis `$objeto->$metodo()`.

```
index.php > ...
1  <?php
2
3  require_once 'app/config/config.php';
4  require_once 'app/modelos/ConnexionDB.php';
5  require_once 'app/modelos/Videojuego.php';
6  require_once 'app/modelos/VideojuegosDAO.php';
7  require_once 'app/modelos/Usuario.php';
8  require_once 'app/modelos/UsuariosDAO.php';
9  require_once 'app/modelos/Categoría.php';
10 require_once 'app/modelos/CategoríasDAO.php';
11 require_once 'app/modelos/Prestamo.php';
12 require_once 'app/modelos/PrestamosDAO.php';
13 require_once 'app/modelos/Reserva.php';
14 require_once 'app/modelos/ReservasDAO.php';
15 require_once 'app/modelos/Comentario.php';
16 require_once 'app/modelos/ComentariosDAO.php';
17 require_once 'app/modelos/Videojuego_probado.php';
18 require_once 'app/modelos/Videojuegos_probadosDAO.php';
19 require_once 'app/modelos/Puntuacion.php';
20 require_once 'app/modelos/PuntuacionesDAO.php';
21 require_once 'app/controladores/ControladorUsuarios.php';
22 require_once 'app/controladores/ControladorVideojuegos.php';
23 require_once 'app/controladores/ControladorReservas.php';
24 require_once 'app/controladores/ControladorPrestamos.php';
25 require_once 'app/controladores/ControladorComentarios.php';
26 require_once 'app/controladores/ControladorVideojuegosProbados.php';
27 require_once 'app/controladores/ControladorPuntuaciones.php';
28
29 require_once 'app/utils/funciones.php';
30 require_once 'app/modelos/Sesion.php';
31
32 //Uso de variables de sesión
33 session_start();
34
```

```

35 //Mapa de enrutamiento
36 $mapa = array(
37     'inicio'=>array('controlador'=>'ControladorVideojuegos',
38                     'metodo'=>'inicio',
39                     'privada'=>false),
40     'ver_videojuego'=>array('controlador'=>'ControladorVideojuegos',
41                             'metodo'=>'verVideojuego',
42                             'privada'=>false),
43     'insertar_videojuego'=>array('controlador'=>'ControladorVideojuegos',
44                                     'metodo'=>'insertarVideojuego',
45                                     'privada'=>true),
46     'editar_videojuego'=>array('controlador'=>'ControladorVideojuegos',
47                                 'metodo'=>'editarVideojuego',
48                                 'privada'=>true),
49     'eliminar_videojuego'=>array('controlador'=>'ControladorVideojuegos',
50                                     'metodo'=>'eliminarVideojuego',
51                                     'privada'=>true),
52     'login'=>array('controlador'=>'ControladorUsuarios',
53                     'metodo'=>'login',
54                     'privada'=>false),
55     'logout'=>array('controlador'=>'ControladorUsuarios',
56                      'metodo'=>'logout',
57                      'privada'=>true),
58     'registrar'=>array('controlador'=>'ControladorUsuarios',
59                          'metodo'=>'registrar',
60                          'privada'=>false),
61     'cambiar_rol_usuario'=>array('controlador'=>'ControladorUsuarios',
62                                    'metodo'=>'cambiarRolUsuario',
63                                    'privada'=>true),
64     'videojuegos_por_categoria'=>array('controlador'=>'ControladorVideojuegos',
65                                         'metodo'=>'verVideojuegosPorCategoria',
66                                         'privada'=>false),
67     'poner_reserva'=>array('controlador'=>'ControladorReservas',
68                             'metodo'=>'insertarReserva',
69                             'privada'=>false),
70     'quitar_reserva'=>array('controlador'=>'ControladorReservas',
71                             'metodo'=>'borrarReserva',
72                             'privada'=>false),
73     'quitar_reserva_admin'=>array('controlador'=>'ControladorReservas',

```

```

73     'quitar_reserva_admin'=>array('controlador'=>'ControladorReservas',
74             'metodo'=>'quitarReservaAdmin',
75             'privada'=>true),
76     'ver_reservas'=>array('controlador'=>'ControladorReservas',
77             'metodo'=>'verReservas',
78             'privada'=>false),
79     'ver_todas_reservas'=>array('controlador'=>'ControladorReservas',
80             'metodo'=>'verTodasReservas',
81             'privada'=>false),
82     'poner_prestamo'=>array('controlador'=>'ControladorPrestamos',
83             'metodo'=>'insertarPrestamo',
84             'privada'=>false),
85     'devolver_videojuego'=>array('controlador'=>'ControladorPrestamos',
86             'metodo'=>'devolverVideojuego',
87             'privada'=>false),
88     'devolver_prestamo_admin'=>array('controlador'=>'ControladorPrestamos',
89             'metodo'=>'devolverPrestamoAdmin',
90             'privada'=>true),
91     'ver_prestamos'=>array('controlador'=>'ControladorPrestamos',
92             'metodo'=>'verPrestamos',
93             'privada'=>false),
94     'ver.todos.prestamos'=>array('controlador'=>'ControladorPrestamos',
95             'metodo'=>'verTodosPrestamos',
96             'privada'=>false),
97     'poner_videojuego_probado'=>array('controlador'=>'ControladorVideojuegosProbados',
98             'metodo'=>'ponerVideojuegoProbado',
99             'privada'=>false),
100    'quitar_videojuego_probado'=>array('controlador'=>'ControladorVideojuegosProbados',
101            'metodo'=>'quitarVideojuegoProbado',
102            'privada'=>false),
103    'ver.videojuegos_probados'=>array('controlador'=>'ControladorVideojuegosProbados',
104            'metodo'=>'verVideojuegosProbados',
105            'privada'=>false),
106    'ver.todos.videojuegos_probados'=>array('controlador'=>'ControladorVideojuegosProbados',
107            'metodo'=>'verTodosVideojuegosProbados',
108            'privada'=>false),
109    'guardar_puntuacion'=>array('controlador'=>'ControladorPuntuaciones',
110            'metodo'=>'guardarPuntuacion',
111            'privada'=>false),
112    'guardar_comentario'=>array('controlador'=>'ControladorComentarios',
113            'metodo'=>'guardarComentario').

```

```

111 |           'privada'=>false),
112 |     'guardar_comentario'=>array('controlador'=>'ControladorComentarios',
113 |                                     'metodo'=>'guardarComentario',
114 |                                     'privada'=>false),
115 |     'editar_comentario'=>array('controlador'=>'ControladorComentarios',
116 |                                     'metodo'=>'editarComentario',
117 |                                     'privada'=>false),
118 |     'eliminar_comentario'=>array('controlador'=>'ControladorComentarios',
119 |                                     'metodo'=>'eliminarComentario',
120 |                                     'privada'=>false),
121 |     'configuraciones_videojuegos'=>array('controlador'=>'ControladorVideojuegos',
122 |                                             'metodo'=>'configuracionesVideojuegos',
123 |                                             'privada'=>false),
124 |     'sobre_nosotros'=>array('controlador'=>'ControladorUsuarios',
125 |                               'metodo'=>'sobreNosotros',
126 |                               'privada'=>false),
127 |     'buscar_videojuegos_ajax'=>array('controlador'=>'ControladorVideojuegos',
128 |                                         'metodo'=>'buscarVideojuegosAjax',
129 |                                         'privada'=>false)
130 );
131

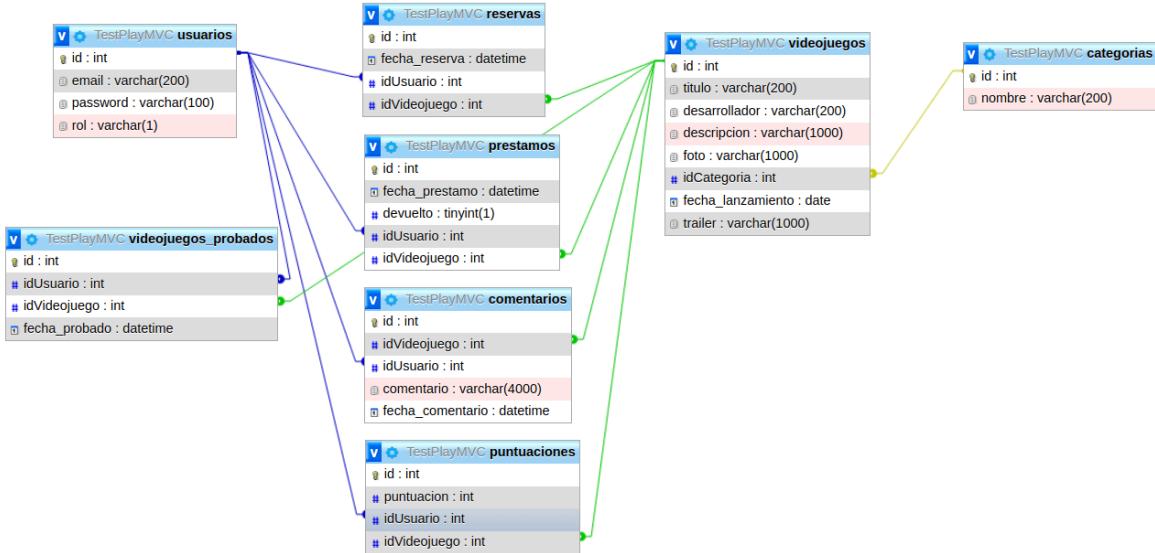
```

```

133 //Parseo de la ruta
134 if(isset($_GET['accion'])){ //Compruebo si me han pasado una acción concreta, sino pongo la acción por defecto inicio
135   if(isset($mapa[$_GET['accion']])){
136     $accion = $_GET['accion'];
137   }
138   else{
139     //La acción no existe
140     header('Status: 404 Not found');
141     echo 'Página no encontrada';
142     die();
143   }
144 }else{
145   $accion='inicio'; //Acción por defecto
146 }
147
148 //Si la acción es privada compruebo que ha iniciado sesión, sino, lo echamos a index
149 if(!Sesion::existeSesion() && $mapa[$accion]['privada']){
150   header('location: index.php');
151   guardarMensaje("Debes iniciar sesión para acceder a $accion");
152   die();
153 }
154
155
156 //La acción ya tiene la acción a ejecutar, cogemos el controlador y metodo a ejecutar del mapa
157 $controlador = $mapa[$accion]['controlador'];
158 $metodo = $mapa[$accion]['metodo'];
159
160 $objeto = new $controlador();
161 $objeto->$metodo();
162
163 |

```

- **Base de datos(sql)**, incluyendo las tablas principales (usuarios, videojuegos, reservas, etc.), claves foráneas y relaciones.



- Funcionalidades implementadas con detalle por archivo (clases, DAOs, controladores, etc.):

La clase **ConnexionDB** es la encargada de gestionar la conexión con la base de datos MySQL. Utiliza un constructor que recibe los parámetros necesarios para establecer la conexión (usuario, contraseña, host y nombre de la base de datos) y crea una instancia del objeto mysqli.

Si la conexión falla, el sistema detiene la ejecución mostrando un mensaje de error. La clase proporciona el método `getConnexion()` para obtener el objeto de conexión activa y poder usarlo en otras partes del proyecto (como los DAO).

```

<?php

class ConnexionDB {

    private $conn;

    function __construct($user, $password, $host, $database)
    {
        $this->conn = new mysqli($host,$user,$password,$database);
        if($this->conn->connect_error){
            die('Error al conectar con MySQL');
        }
    }

    function getConnexion(){
        return $this->conn;
    }
}

```

Configuración con la Base de Datos:

```

<?php

define('MYSQL_USER', 'root');
define('MYSQL_PASS', '');
define('MYSQL_DB', 'TestPlayMVC');
define('MYSQL_HOST', 'localhost');

```

Clases del modelo. Cada clase representa una entidad del sistema y define sus propiedades y métodos para acceder a ellas.

Videojuego.php, es la clase que representa un videojuego en la aplicación. Contiene propiedades como:

- id, titulo, desarrollador, descripcion, foto, idCategoria, etc.

Incluye métodos “getXXX()” que devuelven el valor de la propiedad correspondiente, mientras que los métodos “setXXX()” permiten establecer un nuevo valor para la propiedad.

Esta estructura es común para las otras clases del sistema, como Usuario.php, Prestamo.php, Comentario.php, etc. Solamente cambian las propiedades.

```
<?php

class Videojuego {
    private $id;
    private $titulo;
    private $desarrollador;
    private $descripcion;
    private $foto;
    private $idCategoria;
    private $fecha_lanzamiento;
    private $trailer;

    /**
     * Get the value of id
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set the value of id
     */
    public function setId($id): self
    {
        $this->id = $id;

        return $this;
    }

    /**
     * Get the value of titulo
     */
    public function getTitulo()
    {
        return $this->titulo;
    }

    /**
     * Set the value of titulo
     */
    public function setTitulo($titulo): self
    {
        $this->titulo = $titulo;

        return $this;
    }

    /**
     * Get the value of desarrollador
     */
    public function getDesarrollador()
    {
        return $this->desarrollador;
    }

    /**
     * Set the value of desarrollador
     */
    public function setDesarrollador($desarrollador): self
    {
        $this->desarrollador = $desarrollador;

        return $this;
    }

    /**
     * Get the value of descripcion
     */
    public function getDescripcion()
    {
        return $this->descripcion;
    }
}
```

Los **DAO** actúan como intermediarios entre la aplicación y la base de datos. Permiten acceder, insertar, modificar y eliminar registros de forma estructurada y segura. Gracias a los DAO, evitamos escribir directamente sentencias SQL en los controladores, manteniendo un código más limpio y estructurado.

En **VideojuegosDAO.php**, se encapsulan las operaciones de base de datos relacionadas con los videojuegos. Incluyendo métodos como: “getAll()”, `getById($id)`, `insert($videojuego)`, `update($videojuego)`, `delete($id)`.

```
<?php

class VideojuegosDAO {
    private mysqli $conn;

    public function __construct($conn) {
        $this->conn = $conn;
    }

    /**
     * Obtener un videojuego de la BD en función del id pasado
     * @return Videojuego Devuelve el objeto Videojuego o null si no lo encuentra
     */
    public function getById($id):Videojuego|null {
        if(!$stmt = $this->conn->prepare("SELECT * FROM videojuegos WHERE id = ?")){
            echo "Error en la SQL: " . $this->conn->error;
        }
        //Asociar las variables a las interrogaciones(parámetros)
        $stmt->bind_param('i',$id);
        //Ejecutamos la SQL
        $stmt->execute();
        //Obtener el objeto mysql_result
        $result = $stmt->get_result();

        //Si ha encontrado algún resultado devolvemos un objeto de la clase Videojuego, sino null
        if($result->num_rows == 1){
            $videojuego = $result->fetch_object(Videojuego::class);
            return $videojuego;
        }
        else{
            return null;
        }
    }
}
```

```
/**
 * Obtener todas los videojuegos de la tabla videojuegos
 * @return array Devuelve un array de objetos Videojuego
 */
public function getAll():array {
    if(!$stmt = $this->conn->prepare("SELECT * FROM videojuegos")){
        echo "Error en la SQL: " . $this->conn->error;
    }
    //Ejecutamos la SQL
    $stmt->execute();
    //Obtener el objeto mysql_result
    $result = $stmt->get_result();

    $array_videojuegos = array();

    while($videojuego = $result->fetch_object(Videojuego::class)){
        $array_videojuegos[] = $videojuego;
    }
    return $array_videojuegos;
}
```

```

/**
 * Insertar en la base de datos el videojuego que recibe como parámetro
 * @return idVideojuego Devuelve el id autonumérico que se le ha asignado al videojuego o false en caso de error
 */
function insert(Videojuego $videojuego): int|bool{
    if($stmt = $this->conn->prepare("INSERT INTO videojuegos (titulo, desarrollador, descripcion, foto, idCategoria, fecha_lanzamiento, trailer)
VALUES (?, ?, ?, ?, ?, ?, ?)")){
        die("Error al preparar la consulta insert: " . $this->conn->error );
    }
    $titulo = $videojuego->getTitulo();
    $desarrollador = $videojuego->getDesarrollador();
    $descripcion = $videojuego->getDescripcion();
    $foto = $videojuego->getFoto();
    $idCategoria = $videojuego->getIdCategoria();
    $fechaLanzamiento = $videojuego->getFechaLanzamiento();
    $trailer = $videojuego->getTrailer();
    $stmt->bind_param('ssssiss',$titulo, $desarrollador, $descripcion, $foto, $idCategoria, $fechaLanzamiento, $trailer);
    if($stmt->execute()){
        return $stmt->insert_id;
    }
    else{
        return false;
    }
}

```

El resto de los DAO implementados en TestPlay (como los correspondientes a reservas, préstamos, usuarios, puntuaciones, comentarios o videojuegos probados) mantienen la misma estructura y lógica de funcionamiento. En general:

- Cada DAO está vinculado a una clase/tabla concreta del sistema.
- Se encarga de ejecutar consultas SELECT, INSERT, UPDATE y DELETE según las necesidades de la funcionalidad.
- Sus métodos abstraen el acceso a la base de datos, permitiendo al controlador trabajar con métodos claros y reutilizables sin escribir SQL directamente.
- Devuelven objetos del modelo correspondiente o resultados asociados para ser usados por las vistas.

Gracias a este enfoque consistente, el sistema es fácil de mantener, escalar y entender, ya que todas las clases DAO siguen una estructura común con funciones adaptadas a cada caso específico.

Controladores. Los controladores gestionan la lógica principal de la aplicación. Reciben las solicitudes del usuario, interactúan con el modelo y devuelven la vista correspondiente.

ControladorUsuarios.php, gestiona todas las acciones relacionadas con los usuarios de la plataforma. Entre sus funciones principales se encuentran:

- Registro de nuevos usuarios.
- Inicio y cierre de sesión.
- Validación de datos como correos electrónicos duplicados.
- Acceso al área de administración.

También gestiona el almacenamiento seguro de contraseñas (hash) y el control de acceso dependiendo del rol del usuario.

```
1  <?php
2
3  class ControladorUsuarios {
4      public function registrar(){
5          $error = '';
6
7          if ($_SERVER['REQUEST_METHOD'] == 'POST') {
8
9              // Limpiamos los datos
10             $email = trim(htmlentities($_POST['email']));
11             $password = trim(htmlentities($_POST['password']));
12
13             // Validación: campos vacíos
14             if (empty($email) || empty($password)) {
15                 $error = "Los campos email y contraseña no pueden estar vacíos.";
16             }
17             // Validación: longitud mínima del password
18             elseif (strlen($password) < 4) {
19                 $error = "La contraseña debe tener al menos 4 caracteres.";
20             }
21             else [
22                 // Conectamos con la BD
23                 $connexionDB = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
24                 $conn = $connexionDB->getConnexion();
25
26                 // Compruebo que no haya un usuario registrado con el mismo email
27                 $usuariosDAO = new UsuariosDAO($conn);
28                 if ($usuariosDAO->getByEmail($email) != null) {
29                     $error = "Ya existe un usuario con ese email.";
30                 } else {
31                     // Si no hay error, insertamos
32                     $usuario = new Usuario();
33                     $usuario->setEmail($email);
34                     $passwordCifrado = password_hash($password, PASSWORD_DEFAULT);
35                     $usuario->setPassword($passwordCifrado);
36                     $usuario->setRol('U'); // Rol por defecto
37
38                     if ($usuariosDAO->insert($usuario)) {
39                         header("location: index.php?registro=ok");
40                         die();
41                     } else {
42                         $error = "No se ha podido insertar el usuario.";
```

```

39             header("location: index.php?registro=ok");
40             die();
41         } else {
42             $error = "No se ha podido insertar el usuario.";
43         }
44     }
45 }
46
47 require 'app/vistas/registrar.php';
48
49 // Si hay un error, lo mostramos con SweetAlert
50 if (!empty($error)) {
51     echo "<script>
52         document.addEventListener('DOMContentLoaded', function() {
53             Swal.fire({
54                 icon: 'error',
55                 title: 'Error',
56                 text: '".addslashes($error)."' });
57         });
58     </script>";
59 }
60
61 }
62
63 public function login(){
64     $error = '';
65
66     //Creamos la conexión utilizando la clase que hemos creado
67     $connexionDB = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
68     $conn = $connexionDB->getConnexion();
69
70     //limpiamos los datos que vienen del usuario
71     $email = htmlspecialchars($_POST['email']);
72     $password = htmlspecialchars($_POST['password']);
73
74     //Validamos el usuario
75     $usuariosDAO = new UsuariosDAO($conn);
76     if ($usuario = $usuariosDAO->getByEmail($email)) {

```

```

75         //Validamos el usuario
76         $usuariosDAO = new UsuariosDAO($conn);
77         if ($usuario = $usuariosDAO->getByEmail($email)) {
78             if (password_verify($password, $usuario->getPassword())) {
79                 //email y password correctos. Inciamos sesión
80                 Sesion::iniciarSesion($usuario);
81
82                 header('location: index.php');
83                 die();
84             }
85         }
86
87         // Si llega aquí, hay error
88         $error = "Email o contraseña incorrectos.";
89
90         // Cargamos la vista
91         require 'app/vistas/inicio.php';
92
93         // Lanzamos SweetAlert2 con el error
94         if (!empty($error)) {
95             echo "<script>
96                 document.addEventListener('DOMContentLoaded', function() {
97                     Swal.fire({
98                         icon: 'error',
99                         title: 'Error de acceso',
100                         text: '" . addslashes($error) . "' });
101                     });
102                 </script>";
103         }
104     }
105

```

```

102     });
103     </script>";
104 }
105 }
106 }
107
108 public function logout(){
109     Sesion::cerrarSesion();
110     header('location: index.php');
111 }
112
113
114 public function cambiarRolUsuario() {
115     $usuario = Sesion::getUsuario();
116     if (!$usuario || $usuario->getRol() !== 'A') {
117         $_SESSION['mensaje_error'] = 'Acceso denegado.';
118         header('location: index.php');
119         exit;
120     }
121
122     $idUsuario = $_POST['id_usuario'];
123     $nuevoRol = $_POST['nuevo_rol'];
124
125     if ($idUsuario && in_array($nuevoRol, ['A', 'U'])) {
126         $conexion = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
127         $usuariosDAO = new UsuariosDAO($conexion->getConnexion());
128
129         $usuarioActualizar = $usuariosDAO->getById($idUsuario);
130         if ($usuarioActualizar && $usuarioActualizar->getEmail() !== 'admin@gmail.com') {
131             $usuarioActualizar->setRol($nuevoRol);
132             $usuariosDAO->updateRol($usuarioActualizar);
133         }
134     }
135
136     header('location: index.php?accion=configuraciones_videojuegos');
137 }
138
139
140
141 public function sobreNosotros() {
142     // Comprobamos si hay sesión y si hay un usuario conectado
143     $usuario = Sesion::getUsuario();
144     if (!$usuario) {
145         $_SESSION['mensaje_error'] = 'Acceso denegado.';
146         header('location: index.php');
147         exit;
148     }
149
150     // Cargamos la vista
151     require 'app/vistas/sobre_nosotros.php';
152 }
153
154 }
155

```

```

127     $usuariosDAO = new UsuariosDAO($conexion->getConnexion());
128
129     $usuarioActualizar = $usuariosDAO->getById($idUsuario);
130     if ($usuarioActualizar && $usuarioActualizar->getEmail() !== 'admin@gmail.com') {
131         $usuarioActualizar->setRol($nuevoRol);
132         $usuariosDAO->updateRol($usuarioActualizar);
133         $_SESSION['mensaje_ok'] = 'Rol actualizado correctamente.';
134     }
135
136     header('location: index.php?accion=configuraciones_videojuegos');
137 }
138
139
140
141 public function sobreNosotros() {
142     // Comprobamos si hay sesión y si hay un usuario conectado
143     $usuario = Sesion::getUsuario();
144     if (!$usuario) {
145         $_SESSION['mensaje_error'] = 'Acceso denegado.';
146         header('location: index.php');
147         exit;
148     }
149
150     // Cargamos la vista
151     require 'app/vistas/sobre_nosotros.php';
152 }
153
154 }
155

```

ControladorVideojuegos.php, controlador encargado de las funcionalidades relacionadas con los videojuegos, como:

- El inicio de la aplicación.
- Mostrar el listado completo o por categoría.
- Gestionar la creación, edición y eliminación de videojuegos (sólo para administradores).
- Mostrar la vista detallada de un videojuego.

```
1 <?php
2
3 class ControladorVideojuegos {
4
5     public function inicio(){
6
7         //Creamos la conexión utilizando la clase que hemos creado
8         $connexionDB = new ConnexionDB(MYSQL_USER,MYSQL_PASS,MYSQL_HOST,MYSQL_DB);
9         $conn = $connexionDB->getConnexion();
10
11
12         if(Sesion::existeSesion()){
13             //Creamos el objeto VideojuegosDAO para acceder a BBDD a través de este objeto
14             $videojuegosDAO = new VideojuegosDAO($conn);
15
16             // Obtener todas las categorías
17             $categorias = $videojuegosDAO->obtenerTodasLasCategorias();
18         }
19
20         //Incluyo la vista
21         require 'app/vistas/inicio.php';
22     }
23
24
25     // FUNCIÓN PARA VER CADA VIDEOJUEGO
26     public function verVideojuego(){
27         // Comprobamos si hay sesión y si hay un usuario conectado
28         $usuario = Sesion::getUsuario();
29         if (! $usuario) {
30             $_SESSION['mensaje_error'] = 'Acceso denegado.';
31             header('location: index.php');
32             exit;
33         }
34
35         // Creamos la conexión utilizando la clase que hemos creado
36         $connexionDB = new ConnexionDB(MYSQL_USER,MYSQL_PASS,MYSQL_HOST,MYSQL_DB);
37         $conn = $connexionDB->getConnexion();
38
39         // Creamos los objetos DAO necesarios para acceder a BBDD a través de estos objetos
40         $videojuegosDAO = new VideojuegosDAO($conn);
41         $reservasDAO = new ReservasDAO($conn);
```

```

39 // Creamos los objetos DAO necesarios para acceder a BBDD a través de estos objetos
40 $videojuegosDAO = new VideojuegosDAO($conn);
41 $reservasDAO = new ReservasDAO($conn);
42 $categoriasDAO = new CategoriasDAO($conn);
43 $prestamosDAO = new PrestamosDAO($conn);
44 $videojuegosProbadosDAO = new Videojuegos_probadosDAO($conn);
45 $comentariosDAO = new ComentariosDAO($conn);
46 $puntuacionesDAO = new PuntuacionesDAO($conn);
47 $usuariosDAO = new UsuariosDAO($conn);
48
49 // Obtener el videojuego
50 $idVideojuego = htmlspecialchars($_GET['id']);
51 $videojuego = $videojuegosDAO->getById($idVideojuego);
52
53 // Obtener la categoría del videojuego para poder obtener el nombre de la categoría
54 $categoriaId = $videojuego->getIdCategoria();
55 $categoria = $categoriasDAO->getById($categoriaId);
56
57 // Inicializar variables comunes
58 $videojuegoReservado = $reservasDAO->countByIdVideojuego($idVideojuego);
59 $videojuegoPrestado = $prestamosDAO->countByIdVideojuego($idVideojuego);
60 $videojuegoProbado = $videojuegosProbadosDAO->countByIdVideojuego($idVideojuego);
61 $marcadoProbado = false;
62 $existeReserva = false;
63 $usuarioReservado = null;
64 $usuarioPrestamo = null;
65
66 // Obtener el total de votos del videojuego
67 $totalVotos = $puntuacionesDAO->contarVotosVideojuego($videojuego->getId());
68
69 // Obtener comentarios de este videojuego
70 $comentarios = $comentariosDAO->getComentariosPorVideojuego($idVideojuego);
71
72
73 // Solo si hay sesión
74 if (Sesion::existeSesion()) {
75     $usuario = Sesion::getUsuario();
76
77     $usuarioPrestamo = $prestamosDAO->getUsuarioPrestamoPorVideojuegoId($idVideojuego);
78
79     if ($usuario->getRol() === 'U') {
80         // Lógica para usuarios normales
81         $existeReserva = $reservasDAO->existByIdUsuarioIdVideojuego($usuario->getId(), $idVideojuego);
82         $marcadoProbado = $videojuegosProbadosDAO->estaMarcadoComoProbado($usuario->getId(), $idVideojuego);
83
84     } elseif ($usuario->getRol() === 'A') {
85         // Lógica para administradores
86         $usuarioReservado = $videojuegoReservado ? $reservasDAO->getUsuarioReservaPorVideojuegoId($idVideojuego) : null;
87     }
88
89     require 'app/vistas/ver_videojuego.php';
90 }
91
92
93 // FUNCIÓN PARA VER LOS VIDEOJUEGOS POR CATEGORÍA
94 public function verVideojuegosPorCategoria() {
95     // Comprobamos si hay sesión y si hay un usuario conectado
96     $usuario = Sesion::getUsuario();
97     if (!$usuario) {
98         $_SESSION['mensaje_error'] = 'Acceso denegado.';
99         header('location: index.php');
100        exit;
101    }
102
103
104    //Creamos la conexión utilizando la clase que hemos creado
105    $connexionDB = new ConnexionDB(MYSQL_USER,MYSQL_PASS,MYSQL_HOST,MYSQL_DB);
106    $conn = $connexionDB->getConnexion();
107
108    $idCategoria = $_GET['id'];
109
110    //Creamos el objeto VideojuegosDAO para acceder a BBDD a través de este objeto
111    $videojuegosDAO = new VideojuegosDAO($conn);

```

```

74 if (Sesion::existeSesion()) {
75     $usuario = Sesion::getUsuario();
76
77     $usuarioPrestamo = $prestamosDAO->getUsuarioPrestamoPorVideojuegoId($idVideojuego);
78
79     if ($usuario->getRol() === 'U') {
80         // Lógica para usuarios normales
81         $existeReserva = $reservasDAO->existByIdUsuarioIdVideojuego($usuario->getId(), $idVideojuego);
82         $marcadoProbado = $videojuegosProbadosDAO->estaMarcadoComoProbado($usuario->getId(), $idVideojuego);
83
84     } elseif ($usuario->getRol() === 'A') {
85         // Lógica para administradores
86         $usuarioReservado = $videojuegoReservado ? $reservasDAO->getUsuarioReservaPorVideojuegoId($idVideojuego) : null;
87     }
88
89     require 'app/vistas/ver_videojuego.php';
90 }
91
92
93 // FUNCIÓN PARA VER LOS VIDEOJUEGOS POR CATEGORÍA
94 public function verVideojuegosPorCategoria() {
95     // Comprobamos si hay sesión y si hay un usuario conectado
96     $usuario = Sesion::getUsuario();
97     if (!$usuario) {
98         $_SESSION['mensaje_error'] = 'Acceso denegado.';
99         header('location: index.php');
100        exit;
101    }
102
103
104    //Creamos la conexión utilizando la clase que hemos creado
105    $connexionDB = new ConnexionDB(MYSQL_USER,MYSQL_PASS,MYSQL_HOST,MYSQL_DB);
106    $conn = $connexionDB->getConnexion();
107
108    $idCategoria = $_GET['id'];
109
110    //Creamos el objeto VideojuegosDAO para acceder a BBDD a través de este objeto
111    $videojuegosDAO = new VideojuegosDAO($conn);

```

```

110     //Creamos el objeto VideojuegosDAO para acceder a BBDD a través de este objeto
111     $videojuegosDAO = new VideojuegosDAO($conn);
112
113     $categoriasDAO = new CategoriasDAO($conn);
114     $puntuacionesDAO = new PuntuacionesDAO($conn);
115
116     // Obtener el videojuego
117     $idVideojuego = htmlspecialchars($_GET['id']);
118     $videojuego = $videojuegosDAO->getById($idVideojuego);
119
120     // Obtener los videojuegos por categoría
121     $videojuegos = $videojuegosDAO->obtenerVideojuegosPorCategoria($idCategoria);
122
123     $idCategoria = htmlspecialchars($_GET['id']);
124     $categoria = $categoriasDAO->getById($idCategoria);
125
126     $totalVotos = $puntuacionesDAO->contarVotosVideojuego($videojuego->getId());
127
128     // Incluir la vista de videojuegos por categoría
129     require 'app/vistas/videojuegos_por_categoria.php';
130 }
131
132
133 public function insertarVideojuego() {
134     // Comprobamos si hay sesión y si el usuario es administrador
135     $usuario = Sesion::getUsuario();
136     if (!$usuario || $usuario->getRol() !== 'A') {
137         $_SESSION['mensaje_error'] = 'Acceso denegado.';
138         header('location: index.php');
139         exit;
140     }
141
142     // Creamos la conexión utilizando la clase que hemos creado
143     $connexionDB = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
144     $conn = $connexionDB->getConnexion();
145
146     // Creamos los objetos DAO necesarios para acceder a BBDD a través de estos objetos
147     $categoriasDAO = new CategoriasDAO($conn);

```

```

146     // Creamos los objetos DAO necesarios para acceder a BBDD a través de estos objetos
147     $categoriasDAO = new CategoriasDAO($conn);
148
149     // Obtener todas las categorías
150     $categorias = $categoriasDAO->getAll();
151
152     if ($_SERVER['REQUEST_METHOD'] == 'POST') {
153         // Limpiamos los datos que vienen del formulario
154         $titulo = htmlspecialchars($_POST['titulo']);
155         $desarrollador = htmlspecialchars($_POST['desarrollador']);
156         $descripcion = htmlspecialchars($_POST['descripcion']);
157         $idCategoria = htmlspecialchars($_POST['idCategoria']);
158         $fechaLanzamiento = htmlspecialchars($_POST['fecha_lanzamiento']);
159         $trailer = htmlspecialchars($_POST['trailer']);
160
161         // Aquí manejamos la foto
162         $foto_nombre = $_FILES['foto']['name'];    // Nombre de la foto
163         $foto_temporal = $_FILES['foto']['tmp_name']; // Ubicación temporal de la foto
164         $ruta_destino = 'web/images/' . $foto_nombre; // Ruta donde se guardará la foto
165
166
167         // Validamos los datos
168         if (empty($titulo) || empty($desarrollador) || empty($descripcion) || empty($idCategoria) || empty($fechaLanza
169             $_SESSION['mensaje_error'] = "Todos los campos son obligatorios.";
170         } else {
171
172             // Creamos el objeto DAO necesario para acceder a los videojuegos en la base de datos
173             $videojuegosDAO = new VideojuegosDAO($conn);
174
175             // Insertamos el videojuego
176             $videojuego = new Videojuego();
177             $videojuego->setTitulo($titulo);
178             $videojuego->setDesarrollador($desarrollador);
179             $videojuego->setDescripcion($descripcion);
180             $videojuego->setIdCategoria($idCategoria);
181             $videojuego->setFechaLanzamiento($fechaLanzamiento);
182             $videojuego->setTrailer($trailer);
183

```

```

174
175 // Insertamos el videojuego
176 $videojuego = new Videojuego();
177 $videojuego->setTitulo($titulo);
178 $videojuego->setDesarrollador($desarrollador);
179 $videojuego->setDescripcion($descripcion);
180 $videojuego->setIdCategoria($idCategoria);
181 $videojuego->setFechaLanzamiento($fechaLanzamiento);
182 $videojuego->setTrailer($trailer);
183
184
185 // Movemos la foto al directorio final y guardamos la ruta en el objeto Videojuego
186 if (move_uploaded_file($foto_temporal, $ruta_destino)) {
187     $videojuego->setFoto($foto_nombre);
188 } else {
189     $_SESSION['mensaje_error'] = "Error al cargar la foto.";
190 }
191
192 // Insertamos el videojuego en la base de datos
193 if ($videojuegosDAO->insert($videojuego)) {
194     $_SESSION['mensaje_ok'] = "El videojuego se ha creado correctamente.";
195     header('location: index.php?accion=videojuegos_por_categoria&id=' . $videojuego->getIdCategoria());
196     die();
197 } else {
198     $_SESSION['mensaje_error'] = "Error al insertar el videojuego.";
199 }
200
201 }
202
203 // Cargamos la vista de insertar videojuego
204 require 'app/vistas/insertar_videojuego.php';
205
206 }
```

```

288 public function eliminarVideojuego(){
289     // Comprobamos si hay sesión y si el usuario es administrador
290     $usuario = Sesion::getUsuario();
291     if (!$usuario || $usuario->getRol() !== 'A') {
292         $_SESSION['mensaje_error'] = 'Acceso denegado.';
293         header('location: index.php');
294         exit;
295     }
296
297     // Creamos la conexión utilizando la clase que hemos creado
298     $connexionDB = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
299     $conn = $connexionDB->getConnexion();
300
301     //Creamos el objeto VideojuegosDAO para acceder a BBDD a través de este objeto
302     $videojuegosDAO = new VideojuegosDAO($conn);
303
304     //Obtener el videojuego que se quiere eliminar
305     $idVideojuego = htmlspecialchars($_GET['id']);
306     $videojuego = $videojuegosDAO->getById($idVideojuego);
307
308     // Verificamos si existe el videojuego antes de eliminar
309     $videojuego = $videojuegosDAO->getById($idVideojuego);
310     if (!$videojuego) {
311         $_SESSION['mensaje_error'] = 'El videojuego no existe.';
312         header('location: index.php?accion=configuraciones_videojuegos');
313         exit;
314     }
315
316     // Tratamos de eliminar el videojuego
317     if ($videojuegosDAO->delete($idVideojuego)) {
318         $_SESSION['mensaje_ok'] = 'Videojuego eliminado correctamente.';
319     } else {
320         $_SESSION['mensaje_error'] = 'Error al eliminar el videojuego.';
321     }
322
323     // Redirigimos de nuevo a configuraciones
324     header('location: index.php?accion=configuraciones_videojuegos');
325     exit;
326
327 }
```

```

330     public function configuracionesVideojuegos() {
331         // Comprobamos si hay sesión y si el usuario es administrador
332         $usuario = Sesion::getUsuario();
333         if (!$usuario || $usuario->getRol() !== 'A') {
334             $_SESSION['mensaje_error'] = 'Acceso denegado.';
335             header('location: index.php');
336             exit;
337         }
338
339         // Creamos la conexión utilizando la clase que hemos creado
340         $connexionDB = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
341         $conn = $connexionDB->getConnexion();
342
343         // Creamos el objeto VideojuegosDAO para acceder a BBDD a través de este objeto
344         $videojuegosDAO = new VideojuegosDAO($conn);
345         $usuariosDAO = new UsuariosDAO($conn);
346         $categoriasDAO = new CategoriasDAO($conn);
347         $reservasDAO = new ReservasDAO($conn);
348         $prestamosDAO = new PrestamosDAO($conn);
349
350         // Obtenemos todos los videojuegos
351         $videojuegos = $videojuegosDAO->getAll();
352         // Obtenemos todos los usuarios
353         $usuarios = $usuariosDAO->getAll();
354         // Obtenemos todas las reservas
355         $reservas = $reservasDAO->getAll();
356         // Obtenemos todos los préstamos
357         $prestamos = $prestamosDAO->getAll();
358
359         // Inicializar variables comunes
360         $existeReserva = false;
361         $usuarioReservado = null;
362         $usuarioPrestamo = null;
363         $prestamoActivo = false;
364
365
366         require 'app/vistas/configuraciones.php';
367     }
368 }
```

```

370     public function buscarVideojuegosAjax() {
371         if (!Sesion::getUsuario()) {
372             echo json_encode([]);
373             return;
374         }
375
376         // Crear conexión a la base de datos
377         $connexionDB = new ConnexionDB(MYSQL_USER, MYSQL_PASS, MYSQL_HOST, MYSQL_DB);
378         $conn = $connexionDB->getConnexion();
379
380         $query = isset($_GET['query']) ? trim($_GET['query']) : '';
381         require_once 'app/modelos/VideojuegosDAO.php';
382         $videojuegosDAO = new VideojuegosDAO($conn);
383         $resultados = $videojuegosDAO->buscarPorTitulo($query);
384
385         $respuesta = array_map(function($videojuego) {
386             return [
387                 'id' => $videojuego->getId(),
388                 'titulo' => $videojuego->getTitulo()
389             ];
390         }, $resultados);
391
392         header('Content-Type: application/json');
393         echo json_encode($respuesta);
394     }
395
396
397 }
```

Control de acceso a vistas mediante sesión

Para asegurar que solo los usuarios autorizados puedan acceder a determinadas vistas, se utilizan comprobaciones en los **controladores** que gestionan esas secciones. Esto nos permite proteger tanto el acceso por navegación directa como mediante URL manual.

Acceso solo para usuarios conectados. Cuando una vista requiere que el usuario haya iniciado sesión, se comprueba si existe un usuario activo en la sesión:

```
// Comprobamos si hay sesión y si hay un usuario conectado
$usuario = Sesion::getUsuario();
if (! $usuario) {
    $_SESSION['mensaje_error'] = 'Acceso denegado.';
    header('location: index.php');
    exit;
}
```

Esta validación redirige automáticamente a la página principal (index.php) e informa con un mensaje de "Acceso denegado" si no hay un usuario logueado. De esta forma, si la sesión expira o alguien intenta acceder escribiendo directamente la URL de una vista restringida, será redirigido.

Acceso exclusivo para administradores. En las vistas donde solo debe acceder el administrador (por ejemplo, gestión de usuarios o videojuegos), se añade una comprobación adicional del rol:

```
// Comprobamos si hay sesión y si el usuario es administrador
$usuario = Sesion::getUsuario();
if (! $usuario || $usuario->getRol() !== 'A') {
    $_SESSION['mensaje_error'] = 'Acceso denegado.';
    header('location: index.php');
    exit;
}
```

Esto impide que un usuario normal, incluso si está logueado, acceda a secciones como "http...?accion=configuraciones_videojuegos". Al intentarlo, será igualmente redirigido al index.php y verá el mensaje de "Acceso denegado".

Gracias a estas comprobaciones:

- El sistema protege las rutas sensibles incluso si se intenta acceder manualmente por URL.
- Si la sesión ha expirado y el usuario intenta recargar la vista, será expulsado automáticamente.
- Solo los usuarios correctos pueden ver y utilizar las funcionalidades asignadas a su rol.

Esto refuerza la seguridad y coherencia del flujo de navegación dentro de la plataforma.

También es importante señalar que el **resto de controladores** del proyecto (como los encargados de reservas, préstamos, comentarios, puntuaciones o videojuegos probados) **siguen una estructura muy similar**. Cada uno:

- Hereda el mismo patrón de diseño y organización.
- Define métodos específicos para sus funcionalidades concretas.
- Realiza validaciones de sesión y rol cuando es necesario.
- Se comunica con sus respectivos modelos y DAO para interactuar con la base de datos.
- Devuelve o carga las vistas adecuadas según la acción del usuario o administrador.

Esto garantiza una arquitectura coherente, mantenible y escalable, en la que cada módulo está bien separado pero funciona de manera uniforme dentro del sistema.

- Uso de AJAX para interacción dinámica sin recargar la página (reservas, comentarios, puntuaciones...).

Cambiar de tema claro/oscuro dinámicamente. Se almacena la preferencia en localStorage y se actualiza la interfaz sin recargar.

```
// SCRIPT: Toggle entre claro/oscuro solo en main y footer
const btn = document.getElementById("themeToggle");
const icon = document.getElementById("themeIcon");
const main = document.querySelector("main");
const footer = document.querySelector("footer");
const wrapperMain = document.getElementById("mainWrapper");
const wrapperFooter = document.getElementById("footerWrapper");

// Función para aplicar el tema claro u oscuro
function aplicarTema(theme) {
    main.setAttribute("data-bs-theme", theme);
    footer.setAttribute("data-bs-theme", theme);

    if (theme === "dark") {
        wrapperMain.classList.remove("bg-main-light", "text-dark");
        wrapperMain.classList.add("bg-main-dark", "text-light");

        wrapperFooter.classList.remove("bg-footer-light", "text-dark");
        wrapperFooter.classList.add("bg-footer-dark", "text-light");

        document.querySelectorAll('.rating-card').forEach(card => {
            card.classList.remove("bg-rating-light");
            card.classList.add("bg-rating-dark", "text-light");
        });

        document.querySelectorAll('.comment-card').forEach(comment => {
            comment.classList.remove("bg-comment-light", "text-dark");
            comment.classList.add("bg-comment-dark", "text-light");
        });
    } else {
        wrapperMain.classList.remove("bg-main-dark", "text-light");
        wrapperMain.classList.add("bg-main-light", "text-dark");

        wrapperFooter.classList.remove("bg-footer-dark", "text-light");
        wrapperFooter.classList.add("bg-footer-light", "text-dark");
    }
}
```

```

    '',
} else [
    wrapperMain.classList.remove("bg-main-dark", "text-light");
    wrapperMain.classList.add("bg-main-light", "text-dark");

    wrapperFooter.classList.remove("bg-footer-dark", "text-light");
    wrapperFooter.classList.add("bg-footer-light", "text-dark");

    document.querySelectorAll('.rating-card').forEach(card => {
        card.classList.remove("bg-rating-dark", "text-light");
        card.classList.add("bg-rating-light");
    });

    document.querySelectorAll('.comment-card').forEach(comment => {
        comment.classList.remove("bg-comment-dark", "text-light");
        comment.classList.add("bg-comment-light", "text-dark");
    });
]

icon.classList.toggle("fa-sun", theme === "light");
icon.classList.toggle("fa-moon", theme === "dark");
}

// Al cargar la página: leer preferencia o usar claro por defecto
const temaGuardado = localStorage.getItem("modo-tema") || "light";
aplicarTema(temaGuardado);

// Al hacer clic: cambiar el tema y guardarlo
btn.addEventListener("click", () => {
    const actual = main.getAttribute("data-bs-theme");
    const nuevo = actual === "dark" ? "light" : "dark";
    aplicarTema(nuevo);
    localStorage.setItem("modo-tema", nuevo);
});

```

Sistema de reservas. Cada videojuego tiene un botón para "Poner reserva" o "Quitar reserva". Estas acciones se gestionan mediante peticiones AJAX que actualizan el estado en la base de datos y en la interfaz. Para videojuegos probados sería similar.

```
/* Funciones para poner y quitar la reserva con AJAX */
let reservaOn = document.querySelector('.quitarReserva');
if(reservaOn!= null){
    reservaOn.addEventListener('click', quitarReserva);
}

let reservaOff = document.querySelector('.ponerReserva');
if(reservaOff!= null){
    reservaOff.addEventListener('click', ponerReserva);
}

function ponerReserva() {
    let idVideojuego = this.getAttribute('data-idVideojuego');
    fetch('index.php?accion=poner_reserva&id=' + idVideojuego)
        .then(response => response.json())
        .then(data => {
            console.log(data);
            if (data.respuesta === 'ok') {
                console.log('Reserva puesta con éxito');
                actualizarEstadoReserva('reservado');
                let botonReserva = document.getElementById('botonReserva');
                botonReserva.removeEventListener('click', ponerReserva);
                botonReserva.addEventListener('click', quitarReserva);
                botonReserva.innerHTML = "Quitar Reserva";

                Swal.fire({
                    title: '¡Reserva realizada con éxito!',
                    icon: 'success',
                    toast: true,
                    timer: 2500,
                    showConfirmButton: false
                });
            } else {
                console.error('Error al poner reserva');

                Swal.fire({
                    title: 'Error al poner la reserva',
                    icon: 'error',
                    toast: true,
                    timer: 2500.
                });
            }
        })
}
```

```

        } else {
            console.error('Error al poner reserva');

            Swal.fire({
                title: 'Error al poner la reserva',
                icon: 'error',
                toast: true,
                timer: 2500,
                showConfirmButton: false
            });
        });
    }

function quitarReserva() {
    let idVideojuego = this.getAttribute('data-idVideojuego');
    fetch('index.php?accion=quitar_reserva&id=' + idVideojuego)
        .then(response => response.json())
        .then(data => {
            console.log(data);
            if (data.respuesta === 'ok') {
                console.log('Reserva quitada con éxito');
                actualizarEstadoReserva('no_reservado');
                let botonReserva = document.getElementById('botonReserva');
                botonReserva.removeEventListener('click', quitarReserva);
                botonReserva.addEventListener('click', ponerReserva);
                botonReserva.innerHTML = "Poner Reserva";

                Swal.fire({
                    title: 'Reserva eliminada con éxito',
                    icon: 'success',
                    toast: true,
                    timer: 2500,
                    showConfirmButton: false
                });
            }
        });
}

```

```

        timer: 2500,
        showConfirmButton: false
    });

} else {
    console.error('Error al quitar reserva');

    Swal.fire({
        title: 'Error al quitar la reserva',
        icon: 'error',
        toast: true,
        timer: 2500,
        showConfirmButton: false
    });
}

// Función para actualizar el estado de reserva del videojuego
function actualizarEstadoReserva(estado) {
    const estadoReservaContenedor = document.getElementById('estadoReservaContenedor');
    if (estado === 'reservado') {
        estadoReservaContenedor.innerHTML = '<strong class="estadoReservado text-warning">Videojuego Reservado</strong>';
    } else {
        estadoReservaContenedor.innerHTML = '';
    }
}

```

Parte del código de la vista(**ver_videojuego.php**) donde aparecen si el usuario es un usuario común(con rol “U”) los botones para poner y quitar la reserva del videojuego y cambiar el icono de videojuego probado a no probado y viceversa.

```
<?php if ($sesion::existeSession() && $sesion::getUsuario()->getRol() === 'U'): ?>
<div id="estadoProbadoContenedor" class="mt-3">
    <?php if ($marcadoProbado): ?>
        <strong class="estadoProbado text-success">Videojuego Probado</strong>
    <?php endif; ?>
</div>

<!-- Botones reserva / probado -->
<?php if (!$videojuegoReservado || $existeReserva): ?>
<div class="mt-3">
    <?php if ($existeReserva): ?>
        <button class="quitarReserva btn btn-warning" data-idVideojuego=<?= $videojuego->getId() ?>" id="botonReserva">Quitar Reserva</button>
    <?php else: ?>
        <button class="ponerReserva btn btn-warning" data-idVideojuego=<?= $videojuego->getId() ?>" id="botonReserva">Poner Reserva</button>
    <?php endif; ?>
</div>
<?php endif; ?>

<div class="mt-3 estado-probado">
    <?php if ($marcadoProbado): ?>
        <i id="botonProbado" class="fas fa-gamepad quitarProbado icono-probado"
            data-idVideojuego=<?= $videojuego->getId() ?>" style="cursor: pointer; font-size: 2.0rem;" title="Desmarcar probado"></i>
    <?php else: ?>
        <i id="botonProbado" class="fas fa-circle-xmark ponerProbado icono-no-probado"
            data-idVideojuego=<?= $videojuego->getId() ?>" style="cursor: pointer; font-size: 2.0rem;" title="Marcar como probado"></i>
    <?php endif; ?>
    <span class="texto-probado text-success<?= $marcadoProbado ? '' : ' oculto' ?>">
        <i class="fas fa-check-circle icono-confirmacion" style="font-size: 1.5rem;"></i>
    </span>
</div>
<?php endif; ?>
```

Búsqueda de videojuegos por título (barra de navegación). Cuando un usuario escribe en la barra de búsqueda del *header*, se realiza una consulta AJAX para sugerir o mostrar videojuegos que coincidan con el texto introducido. Esto permite un acceso rápido a los videojuegos deseados.

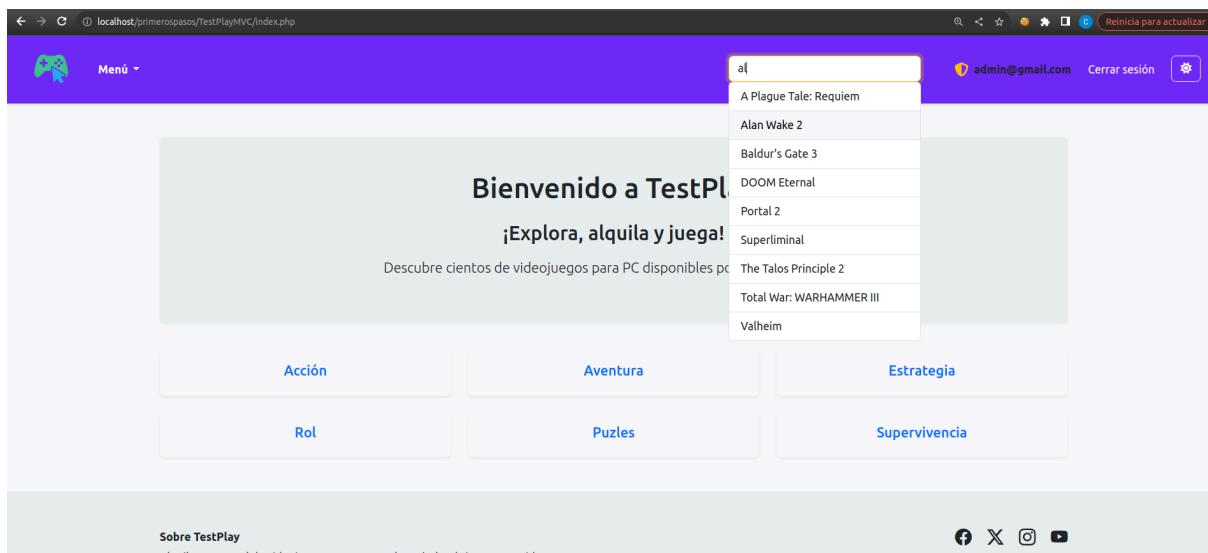
```
// Script para buscar videojuegos
document.addEventListener('DOMContentLoaded', () => {
    const buscador = document.getElementById('buscador-videojuegos');
    const sugerencias = document.getElementById('sugerencias-videojuegos');

    buscador.addEventListener('input', () => {
        const query = buscador.value.trim();
        if (query.length === 0) {
            sugerencias.innerHTML = '';
            return;
        }

        fetch(`index.php?accion=buscar_videojuegos_ajax&query=${encodeURIComponent(query)}`)
            .then(res => res.json())
            .then(data => {
                sugerencias.innerHTML = '';
                data.forEach(juego => {
                    const item = document.createElement('li');
                    item.classList.add('list-group-item', 'list-group-item-action');
                    item.textContent = juego.titulo;
                    item.style.cursor = 'pointer'; // Aplicamos cursor dinámicamente
                    item.addEventListener('click', () => {
                        window.location.href = `index.php?accion=ver_videojuego&id=${juego.id}`;
                    });
                    sugerencias.appendChild(item);
                });
            });
    });

    // Cerrar sugerencias si se hace clic fuera
    document.addEventListener('click', (e) => {
        if (!sugerencias.contains(e.target) && e.target !== buscador) {
            sugerencias.innerHTML = '';
        }
    });
});
```

Como se ve en la vista:



El resto de funcionalidades que hacen uso de AJAX siguen la misma filosofía: mejorar la experiencia del usuario mediante interacciones **dinámicas, sin recargas** y con respuesta inmediata desde el servidor.

Algunos otros casos donde se utiliza AJAX en el proyecto incluyen:

- Marcar y desmarcar videojuegos como **probados**.
- Publicar, editar o eliminar **comentarios** de forma instantánea.
- Votar videojuegos con el sistema de **puntuación por estrellas**.

Todas estas funciones están implementadas con peticiones `fetch()`, y se comunican con controladores específicos del servidor, que procesan los datos y devuelven respuestas en formato JSON o texto. La estructura del código JavaScript es **modular** y **reutilizable**, lo que facilita añadir nuevas funcionalidades interactivas en el futuro.

Gracias a este enfoque, TestPlay ofrece una interfaz más fluida, rápida y moderna, alineada con las buenas prácticas del desarrollo web actual.

b. Documentación de usuario

Se proporciona una guía básica de uso para los distintos perfiles:

- **Usuarios registrados:** pueden buscar videojuegos, reservarlos, marcarlos como probados, puntuarlos y dejar comentarios. También tienen acceso a su historial.

Para el registro en la barra de navegación del inicio aparece el botón “Registrarse” que si pulsamos nos llevará a esa vista donde nos podremos registrar con un usuario(email) y una password, después nos redirigirá al inicio de nuevo donde podremos entrar con nuestro usuario(email) y nuestra password para poder interactuar con la web. Cabe destacar que el rol por defecto al registrarse un usuario es el de un usuario común (“U” en la base de datos).

- **Administradores:** además de la mayoría de las funciones anteriores, pueden gestionar usuarios, modificar roles (excepto `admin@gmail.com`, que es el administrador principal), crear/editar/eliminar videojuegos, crear y devolver préstamos, quitar reservas y ver listados completos agrupados por usuario.

Si fuera necesario algún administrador más además del que hay creado que es “`admin@gmail.com`”, se crearía un usuario y luego un administrador desde configuraciones le daría el rol de administrador (“A” en la base de datos).

Las vistas y funcionalidades han sido diseñadas para ser intuitivas y accesibles, por lo que la mayoría de acciones no requieren formación previa.

Para más detalles visuales, capturas de pantalla explicadas y estructura de la interfaz, consulta el apartado [III.b. Diseño de la interfaz de usuario].

También se explica el funcionamiento de la **cabecera** (barra de navegación), incluyendo el **cambio de tema claro/oscuro** y el uso de la **barra de búsqueda** para acceder rápidamente a los títulos disponibles.

c. Manual de instalación y configuración

Requisitos previos

Antes de comenzar, asegúrate de tener los siguientes elementos instalados en tu sistema:

- **Servidor web local** (uno de los siguientes):
 - **XAMPP** (Windows, Linux, macOS)
 - **WAMP** (solo Windows)
 - **Laragon** (Windows)
 - **LAMP** (Linux)
- **PHP 8.x o superior**
- **MySQL** (integrado en XAMPP/WAMP o instalado aparte)
- **Visual Studio Code** (u otro editor de código)

- **Navegador web** actualizado (Google Chrome, Firefox, etc.)
 - **Git** (opcional, pero recomendable para clonar repositorio y mantener control de versiones)
-

1. Clonar el repositorio

Abre **Visual Studio Code** o tu terminal y ejecuta el siguiente comando:

“git clone https://github.com/tu-usuario/TestPlay.git”

Asegúrate de reemplazar “tu-usuario” por el nombre real de tu cuenta si el repositorio es privado o está en otra URL.

Una vez clonado, abre la carpeta del proyecto desde tu editor o colócalo en el directorio raíz de tu servidor local (por ejemplo, htdocs en XAMPP).

2. Configurar el entorno

1. Mover el proyecto a la carpeta adecuada:

- Si estás usando **XAMPP**, mueve la carpeta “TestPlay” al directorio “htdocs” (por ejemplo: “C:\xampp\htdocs\TestPlay”).
- En **LAMP (Linux)**, colócalo en “/var/www/html/”.

2. Crear la base de datos:

- Abre [phpMyAdmin](#) en tu navegador.
- Crea una nueva base de datos llamada “**TestPlayMVC**” (con cotejamiento utf8_general_ci o similar).

3. Importar la estructura de la base de datos:

- El proyecto incluye un archivo “.sql”, impórtalo desde la opción “Importar” en phpMyAdmin.
- Esto generará las tablas necesarias para que el sistema funcione correctamente.

4. Editar configuración de conexión:

- Abre el archivo “config.php” ubicado en la ruta “/app/config” del proyecto.
- Asegúrate de que los datos coincidan con tu entorno local:

```
define('MYSQL_USER', 'root');

define('MYSQL_PASS', '');

define(MYSQL_DB, 'TestPlayMVC');

define('MYSQL_HOST', 'localhost');
```

3. Ejecutar el proyecto

1. Abre tu navegador y escribe en la barra de direcciones:

“<http://localhost/TestPlay/index.php>”

2. Si todo está configurado correctamente, se cargará la página principal de **TestPlay**.

Y ya puedes empezar a utilizar **TestPlay**.

VI. Mantenimiento y evolución

a. Plan de mantenimiento y soporte

TestPlay está diseñado para facilitar su mantenimiento a lo largo del tiempo. Se han seguido buenas prácticas como la separación de lógica por capas (MVC), el uso de nombres claros y funciones reutilizables. El plan de mantenimiento contempla:

- Revisión periódica del funcionamiento general del sistema.
- Copias de seguridad de la base de datos para evitar pérdidas de información.
- Supervisión de errores reportados por los usuarios.
- Soporte técnico básico por parte del equipo desarrollador en caso de fallos o mejoras urgentes.

Además, el uso de AJAX facilita la actualización dinámica de partes del sistema sin afectar la estructura general, lo que simplifica futuras correcciones o añadidos.

b. Identificación de posibles mejoras y evolución del proyecto

A partir del uso y retroalimentación, se han identificado varias áreas con potencial de mejora:

- Incorporación de **notificaciones automáticas** (correo o mensajes internos) para informar sobre reservas próximas a vencer e incorporar un límite de tiempo que puede estar el videojuego prestado.
- Sistema de **filtros avanzados** en la búsqueda (por género, año, valoración...).
- Mejora del diseño responsive para adaptarse mejor a dispositivos móviles.
- Estadísticas visuales para usuarios (juegos más jugados, puntuaciones dadas, etc.).

- Panel de control más completo para administradores con gráficas y métricas.

Estas mejoras podrían aumentar la interacción y la utilidad del sistema tanto para usuarios como para administradores.

c. Actualizaciones y mejoras futuras

El proyecto está preparado para futuras ampliaciones sin comprometer la estabilidad del sistema. Algunas ideas previstas son:

- Añadir la posibilidad de subir **portadas personalizadas** al crear o editar videojuegos.
- Incluir una **API REST** para permitir integración con otras plataformas o apps móviles.
- Implementación de un sistema de **sanciones o alertas** en caso de uso indebido (por ejemplo, reservas no recogidas).
- Posibilidad de gestionar periodos de alquiler y devoluciones automáticas con fecha límite.

Estas actualizaciones serán consideradas según el crecimiento y las necesidades futuras de la plataforma y sus usuarios.

VII. Conclusiones

a. Evaluación del proyecto

El desarrollo del sistema **TestPlay** ha resultado ser una experiencia satisfactoria tanto a nivel técnico como funcional. Se ha logrado construir una plataforma para la gestión de alquileres temporales de videojuegos, permitiendo a usuarios interactuar de forma intuitiva con el catálogo y a los administradores mantener un control completo del sistema.

El uso del patrón MVC, la implementación de AJAX para mejorar la experiencia de usuario y el enfoque modular han contribuido a una estructura organizada, escalable y fácil de mantener.

b. Cumplimiento de objetivos y requisitos

Los objetivos planteados al inicio se han cumplido en su totalidad:

- Se ha desarrollado un sistema funcional que permite el registro, reserva, préstamo, puntuación y comentario de videojuegos.
- Se han respetado las diferencias entre los roles de usuario y administrador.
- La interfaz es clara, con diseño adaptable y funcionalidades dinámicas.
- La documentación técnica y de usuario facilita la comprensión y mantenimiento del sistema.

Además, se han incorporado funcionalidades adicionales no previstas inicialmente, como el modo oscuro/claro, los historiales agrupados y la edición de comentarios, lo cual ha enriquecido el resultado final.

c. Lecciones aprendidas y recomendaciones para futuros proyectos

A lo largo del desarrollo del proyecto he aprendido importantes lecciones:

- La planificación anticipada y el uso de estructuras claras (como MVC) evitan problemas a largo plazo.
- La implementación progresiva y el testeo constante permiten detectar errores de forma temprana.
- La experiencia de usuario es fundamental: pequeños detalles como AJAX o la visibilidad según rol hacen la diferencia.

Para futuros proyectos recomiendo:

- Definir de forma más detallada el alcance y prioridades desde el inicio.
- Documentar desde las primeras etapas para evitar acumulación al final.
- Considerar desde el comienzo aspectos de accesibilidad y diseño responsive más profundo.

VIII. Bibliografía y referencias

a. Fuentes utilizadas en el proyecto

<https://www.w3schools.com/>

<https://www.php.net/docs.php>

https://developer.mozilla.org/es/docs/Learn_web_development/Core/Scripting/Network_requests

<https://www.w3schools.com/jquery/default.asp>

<http://developer.mozilla.org/es/docs/Web/CSS>

<https://getbootstrap.com/docs/5.3/getting-started/introduction/>

<https://fontawesome.com/>

<https://github.com/>

<https://sweetalert2.github.io/>

b. Referencias y enlaces de interés

Proyectos realizados en clase.

Referencias en webs como:

<https://store.steampowered.com/>

<https://www.instant-gaming.com/en/>