# PROYECTO FINAL

**ESGames: Desarrollo de una Plataforma Web para la Venta de Videojuegos UtilizandoDjango y Vue.js**

| Estudiantes | Escuela | Asignatura |
|---|---|---|
| Flores Nuñes, Rodrigo Francisco<br>Jimenez Paredes, Fabricio Gabriel<br>Leon Ramos, Mijael Paul<br>Paredes Miranda, Mauricio Antonio | Escuela Profesional de Ingeniería de Sistemas | Programación Web II<br>Semestre: III |

| Semestre académico | Fecha de inicio | Fecha de entrega |
|---|---|---|
| 2025 - A | Del 1 Julio 2025 | Al 27 Julio 2025 |

## 1. Tipo de Sistema

ESGames es una aplicación web de comercio digital que simula una tienda virtual de videojuegos, desarrollada con el framework Django 4 para el backend y Vue.js para el frontend. La arquitectura adoptada se basa en el modelo cliente-servidor, con una API RESTful que permite el intercambio eficiente de información entre ambos extremos de la aplicación. La interfaz de usuario (frontend) fue construida como una Single Page Application (SPA) y desplegada en Netlify, una plataforma especializada en alojamiento de sitios estáticos y aplicaciones web modernas. El backend fue desplegado en Render, un servicio en la nube que permite alojar aplicaciones web y APIs, ofreciendo soporte nativo para proyectos Django y PostgreSQL.

## 2. Modelo de Datos

El sistema **ESGames** implementa un modelo de datos relacional basado en una arquitectura cliente-servidor. Las entidades se definieron inicialmente en Django ORM y posteriormente se adaptaron para su implementación en Supabase. A continuación, se describen las entidades principales y sus relaciones.

### 2.1. Entidades Principales

- **UserProfile**: Representa a un usuario registrado. Se relaciona uno a uno con el modelo `User` de Django. Puede tener un carrito de compras, múltiples videojuegos adquiridos, favoritos y valoraciones.
- **Videogame**: Representa un videojuego en la tienda. Contiene información como título, descripción, precio, fecha de lanzamiento y pertenece a una categoría. Puede estar relacionado con muchos carritos, bibliotecas, favoritos y reseñas.
- **Cart**: Representa el carrito de compras de un usuario. Se relaciona uno a uno con un `UserProfile` y contiene múltiples `CartItem`.
- **CartItem**: Representa un ítem dentro del carrito de compras. Se asocia a un carrito y a un videojuego específicos.

- **Library**: Representa los videojuegos adquiridos por un usuario. Permite establecer una relación de muchos a muchos entre usuarios y videojuegos, almacenando además la fecha de adquisición.
- **Favorite**: Representa los videojuegos marcados como favoritos por un usuario. Establece una relación muchos a muchos entre `UserProfile` y `Videogame`.
- **Review**: Representa una reseña escrita por un usuario para un videojuego adquirido. Contiene un comentario, una calificación de 1 a 5 y la fecha de emisión.

### 2.2. Relaciones entre Entidades

- Un `UserProfile` tiene:
  - Un `Cart` (relación 1:1).
  - Muchos `CartItem` a través del `Cart`.
  - Muchos `Library`, `Favorite` y `Review`.
- Un `Videogame` puede:
  - Estar en múltiples `CartItem`, `Library`, `Favorite` y `Review`.
  - Pertenecer a una `Category`.
- Un `Cart` tiene muchos `CartItem`.
- Un `Library` y un `Favorite` representan relaciones N:M entre `UserProfile` y `Videogame`.

### 2.3. Modelo Relacional

A continuación se muestra un resumen simplificado de las relaciones:

- `UserProfile` $\longrightarrow$ `Cart` (1:1)
- `Cart` $\longrightarrow$ `CartItem` (1:N)
- `CartItem` $\longrightarrow$ `Videogame` (N:1)
- `UserProfile` $\longrightarrow$ `Library` $\longleftarrow$ `Videogame` (N:M)
- `UserProfile` $\longrightarrow$ `Favorite` $\longleftarrow$ `Videogame` (N:M)
- `UserProfile` $\longrightarrow$ `Review` $\longleftarrow$ `Videogame` (1:N)
- `Videogame` $\longrightarrow$ `Category` (N:1)

## 3. Diccionario de Datos: `Videogame`

La tabla Videogame es una de las entidades centrales del sistema ESGames, ya que representa el producto principal que los usuarios pueden explorar, comprar, valorar o marcar como favorito. Su diseño busca ser lo suficientemente flexible para almacenar tanto los datos descriptivos como las referencias necesarias para integrarse con otras entidades del sistema.

| Atributo | Tipo | Nulo | Clave | Predeterminado | Descripción |
|---|---|---|---|---|---|
| id | Entero | No | Primaria | Autoincremental | Identificador único del videojuego. |
| title | Cadena (50) | No | No | Ninguno | Título del videojuego. |
| description | Texto | Sí | No | Vacío | Descripción del contenido del videojuego. |
| price | Decimal(10,2) | No | No | Ninguno | Precio del Videojuego. |
| stock | Entero | No | No | 0 | Cantidad disponible para comprar. |
| release_date | Fecha | Sí | No | NULL | Fecha de lanzamiento oficial del videojuego. |
| category_id | Entero (FK) | Sí | Foránea | NULL | Categoría a la que pertenece el videojuego. |
| image_url | Cadena (255) | Sí | No | Vacío | URL de la imagen del videojuego. |

Tabla 1: Diccionario de datos de la entidad `Videogame`.

## 4. Diagrama Entidad-Relación

El siguiente diagrama representa las entidades principales del sistema **ESGames** y sus relaciones. Cada entidad contiene atributos relevantes para el funcionamiento del sistema, y las relaciones reflejan cómo interactúan los usuarios con los videojuegos mediante compras, favoritos, valoraciones y carrito de compras.

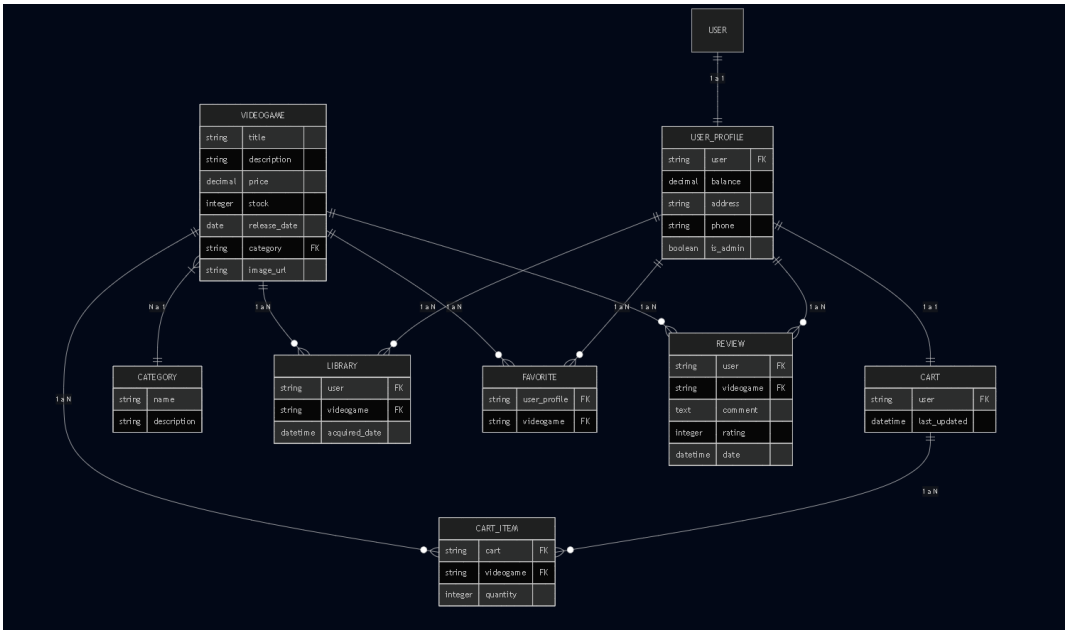


Figura 1: Diagrama Entidad-Relación del sistema `ESGames`.

## 5. Administración con Django

Para la implementación del sistema **ESGames**, se utilizó el framework `Django 4`, el cual proporciona un sistema administrativo robusto para gestionar modelos y datos de forma eficiente. A continuación, se describen los pasos realizados para configurar la administración y gestión del sistema:

## 5.1. Creación del Proyecto y la Aplicación

Se inició la construcción del sistema mediante los siguientes comandos:

```
django-admin startproject gameStore
cd gameStore
python manage.py startapp ESGames
```

El proyecto principal `gameStore` contiene la configuración global del sistema, mientras que la aplicación `ESGames` encapsula los modelos, vistas, serializadores y lógica de negocio principal.

## 5.2. Definición de Modelos

Los modelos del sistema, como `Videogame`, `UserProfile`, `Cart`, `CartItem`, entre otros, fueron definidos en la carpeta `models/.py`, utilizando el ORM de Django. Cada modelo representa una tabla en la base de datos relacional.

## 5.3. Generación y Aplicación de Migraciones

Una vez definidos los modelos, se generaron las migraciones necesarias para sincronizar la base de datos mediante:

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

Esto permitió crear las estructuras de tablas en Supabase, la base de datos relacional utilizada por el sistema.

## 5.4. Habilitación del Panel de Administración

Django proporciona una interfaz administrativa integrada en la URL `/admin`. Para habilitar el acceso, se registraron los modelos en el archivo `ESGames/admin.py`:

```
from django.contrib import admin
from .models import Videogame, UserProfile, Cart, CartItem, Review, Favorite, Library

admin.site.register(Videogame)
admin.site.register(UserProfile)
admin.site.register(Cart)
admin.site.register(CartItem)
admin.site.register(Review)
admin.site.register(Favorite)
admin.site.register(Library)
```

## 5.5. Creación de Superusuario

Para acceder al panel de administración, se creó un superusuario ejecutando:

```
python manage.py createsuperuser
```

Una vez autenticado, se pudo gestionar videojuegos, usuarios, reseñas y otros datos del sistema directamente desde el panel administrativo web.

### 5.6. Ventajas del Administrador de Django

El sistema administrativo permitió:

- Visualización y edición de registros sin necesidad de crear interfaces personalizadas.
- Creación, modificación y eliminación de datos relacionados con usuarios y productos.
- Validación automática de formularios.
- Control de acceso mediante credenciales y permisos.

Este entorno aceleró significativamente el proceso de pruebas y carga de información durante el desarrollo.

## 6. Estructura del Frontend

El frontend del sistema **ESGames** fue desarrollado utilizando el framework `Vue.js 3` en conjunto con `Vite` como sistema de empaquetado. El despliegue del cliente se realizó en la plataforma `Netlify`, lo cual permite una integración continua y despliegue automático tras cada cambio en el repositorio.

### 6.1. Organización del Proyecto

La estructura del proyecto está organizada de forma modular en la carpeta `src`, la cual contiene los siguientes elementos principales:

- **views/**: Contiene las vistas principales del sistema como Home, Login, Register, Store, Cart y GameDetails.
- **components/**: Aquí se encuentran los componentes reutilizables como tarjetas de videojuegos, encabezados, formularios, etc.
- **api/**: Funciones centralizadas que gestionan la comunicación con el backend desplegado en Render.
- **router/**: Configuración de rutas mediante `vue-router`, definiendo navegación entre vistas.
- **store/**: Gestión de estado global usando `Pinia`, permitiendo mantener sincronizados los datos del usuario, carrito y biblioteca.
- **style.css**: Archivo global de estilos con diseño personalizado inspirado en estética moderna de videojuegos.

### 6.2. Diseño Visual y Experiencia de Usuario

El diseño visual fue desarrollado de forma personalizada mediante hojas de estilo CSS y aprovechando clases utilitarias. El estilo general busca una interfaz amigable, clara y responsiva, con una paleta de colores oscura y elementos destacados mediante efectos sutiles como sombras y bordes redondeados.

### 6.3. Navegación y Responsividad

Se implementó un sistema de rutas dinámicas que permite cargar detalles de cada videojuego desde la API del backend. El sistema es responsivo, adaptándose a dispositivos móviles y pantallas de escritorio gracias al uso de Flexbox y media queries.

### 6.4. Ventajas del Enfoque Modular

El uso de Vue.js y la separación de vistas y componentes favorece la mantenibilidad, la escalabilidad y la reutilización de código. Esto permite futuras mejoras en el diseño o integración de nuevas funcionalidades sin alterar el núcleo del sistema.

### 6.5. Despliegue

El proyecto fue desplegado en `Netlify`, aprovechando su integración con GitHub y su soporte para proyectos Vite. Cada vez que se hace un `push` a la rama principal, el sistema se vuelve a compilar y actualizar automáticamente.

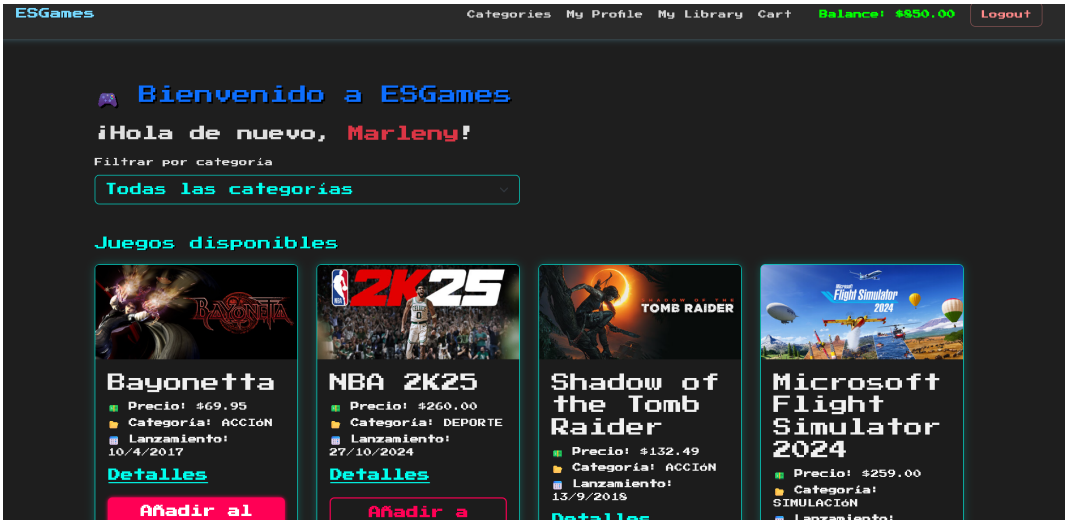


Figura 2: Interfas Principal con usuario registrado

## 7. Servicios mediante una API RESTful

El sistema **ESGames** expone su funcionalidad principal mediante una arquitectura RESTful, permitiendo la comunicación entre el frontend desarrollado en `Vue.js` y el backend en `Django`. Para la creación de esta API se utilizó el `Django REST Framework` (DRF), el cual facilita la serialización de datos, gestión de rutas, autenticación y validación de permisos.

### 7.1. Estructura de la API

Los recursos principales del sistema (usuarios, videojuegos, carrito, biblioteca, favoritos y reseñas) están expuestos a través de endpoints organizados bajo la estructura:

- `/api/videogames/`
  Listado general de videojuegos disponibles. Soporta búsqueda y filtrado.
- `/api/cart/`
  Gestión del carrito de compras del usuario autenticado: agregar, actualizar o eliminar juegos.
- `/api/library/`
  Consulta de los videojuegos adquiridos por el usuario.
- `/api/reviews/`
  Lectura y creación de reseñas vinculadas a videojuegos comprados.

- `/api/favorites/`
  Agregado y eliminación de videojuegos marcados como favoritos.
- `/api/auth/`
  Módulo de autenticación: registro, login, obtención de token JWT.

# 8. Seguridad y Autenticación con Tokens

El sistema **ESGames** implementa un mecanismo de autenticación basado en *tokens* para garantizar la seguridad de los recursos expuestos a través de su API RESTful. Esta técnica es fundamental para proteger las operaciones críticas, como la compra de videojuegos, la gestión del carrito, la visualización de bibliotecas personales y la publicación de reseñas.

## 8.1. Autenticación Token

El sistema utiliza el módulo `TokenAuthentication` de Django REST Framework para generar y validar tokens de sesión. Al momento del inicio de sesión, el servidor genera un token único para el usuario autenticado, el cual debe ser incluido en todas las solicitudes posteriores a endpoints protegidos.

```
POST /api/auth/login/
{
  "username": "usuario",
  "password": "clave123"
}
```

La respuesta del servidor incluye el token:

```
{
  "token": "123abc456def..."
}
```

Este token se almacena en el cliente (por ejemplo, en el `localStorage`) y se envía en el encabezado `Authorization` con cada solicitud autenticada:

```
Authorization: Token 123abc456def...
```

## 8.2. Protección de Endpoints

Todos los endpoints sensibles (como la gestión de carrito, biblioteca, favoritos y reseñas) están protegidos mediante decoradores o clases que exigen autenticación previa. Por ejemplo:

```
from rest_framework.permissions import IsAuthenticated

class CartView(APIView):
    permission_classes = [IsAuthenticated]
    ...
```

Esto garantiza que solo usuarios autenticados puedan acceder a recursos propios y realizar operaciones.

### 8.3. Ventajas del Enfoque

- **Seguridad**: Los datos del usuario solo se exponen si el token válido es presentado.
- **Escalabilidad**: La autenticación por token permite integrar fácilmente otros clientes .
- **Independencia de Sesión**: A diferencia de las cookies, los tokens no dependen de sesiones almacenadas en el servidor.

### 8.4. Buenas Prácticas Aplicadas

- Los tokens no se exponen en rutas públicas ni se almacenan en variables inseguras.
- Se recomienda usar HTTPS en producción para evitar ataques de tipo *man-in-the-middle*.

Este sistema de autenticación proporciona una capa de seguridad robusta que protege los datos del usuario final y garantiza que las operaciones del sistema se ejecuten de manera segura.

## 9. Servicios mediante una API RESTful

El sistema **ESGames** expone su funcionalidad principal mediante una arquitectura RESTful, permitiendo la comunicación entre el frontend desarrollado en `Vue.js` y el backend en `Django`. Para la creación de esta API se utilizó el `Django REST Framework` (DRF), el cual facilita la serialización de datos, gestión de rutas, autenticación y validación de permisos.

### 9.1. Estructura de la API

Los recursos principales del sistema (usuarios, videojuegos, carrito, biblioteca, favoritos y reseñas) están expuestos a través de endpoints organizados bajo la estructura:

- `/api/videogames/`
  Listado general de videojuegos disponibles. Soporta búsqueda y filtrado.
- `/api/cart/`
  Gestión del carrito de compras del usuario autenticado: agregar, actualizar o eliminar juegos.
- `/api/library/`
  Consulta de los videojuegos adquiridos por el usuario.
- `/api/reviews/`
  Lectura y creación de reseñas vinculadas a videojuegos comprados.
- `/api/favorites/`
  Agregado y eliminación de videojuegos marcados como favoritos.
- `/api/auth/`
  Módulo de autenticación: registro, login, obtención de token JWT.

### 9.2. Serialización de Datos

Cada modelo del sistema cuenta con su respectivo `serializer`, el cual transforma las instancias de objetos Python en datos JSON aptos para ser enviados al cliente. Por ejemplo, el modelo `Videogame` se serializa mediante:

```
class VideogameSerializer(serializers.ModelSerializer):
    class Meta:
        model = Videogame
        fields = '__all__'
```

Estos serializadores también se encargan de validar datos de entrada antes de ser persistidos.

### 9.3. Métodos HTTP y Operaciones

La API expone los métodos HTTP estándares para operar sobre los recursos:

- `GET`: Consultar información (ej. lista de juegos, detalles de un carrito, biblioteca del usuario).
- `POST`: Crear nuevos recursos (ej. agregar juegos al carrito, registrar un usuario, enviar una reseña).
- `PUT/PATCH`: Actualizar información existente (ej. cantidad de un juego en el carrito).
- `DELETE`: Eliminar elementos (ej. remover un juego de favoritos o del carrito).

### 9.4. Consumo desde el Frontend

Desde el cliente Vue.js, estas rutas se consumen mediante peticiones asincrónicas usando `fetch()` o librerías como `axios`. Por ejemplo:

```
const response = await fetch("https://api.esgames.com/api/videogames/");
const data = await response.json();
```

Los datos se presentan dinámicamente en las vistas del sistema, con validaciones y retroalimentación para el usuario.

### 9.5. Pruebas de Consumo

Para verificar el correcto funcionamiento de la API, se realizaron pruebas desde distintas interfaces cliente:

- Frontend oficial en Netlify.
- Herramientas de prueba como SoapUI.

Estas pruebas permitieron validar la consistencia de la API, la correcta autenticación y el cumplimiento del protocolo REST.

## 10. Generación y Descarga de Boletas en PDF

El sistema **ESGames** permite al usuario generar una boleta o comprobante digital al completar una compra. Esta funcionalidad tiene como propósito proporcionar al usuario un registro formal de su adquisición, incluyendo detalles como los videojuegos comprados, precios, fecha de transacción y datos personales del cliente.

### 10.1. Renderización en HTML

Inicialmente, la boleta es construida como una plantilla HTML estilizada, la cual se visualiza directamente desde el navegador luego de una compra exitosa. Esta plantilla utiliza etiquetas semánticas y estilos CSS definidos para garantizar una presentación clara, estructurada y adaptada al diseño general del sistema.

## 10.2. Conversión a PDF

Para permitir la descarga del comprobante, el HTML generado se convierte a formato PDF utilizando bibliotecas del entorno Python. En este caso, se empleó la librería `WeasyPrint`, la cual permite transformar contenido HTML + CSS a un documento PDF completamente formateado.

La conversión se realiza a través de una vista del backend, accesible mediante un endpoint autenticado:

```
from weasyprint import HTML
from django.http import HttpResponse

def generate_invoice_pdf(request, order_id):
    html_string = render_to_string("invoice_template.html", context)
    html = HTML(string=html_string)
    pdf = html.write_pdf()

    response = HttpResponse(pdf, content_type="application/pdf")
    response['Content-Disposition'] = 'attachment; filename="invoice.pdf"'
    return response
```

## 10.3. Descarga desde el Frontend

En el frontend, el usuario puede descargar el comprobante haciendo clic en un botón disponible al finalizar su compra. Esta acción realiza una solicitud al endpoint correspondiente y fuerza la descarga del archivo:

```
<a :href="`/api/invoice/${orderId}/`" target="_blank">
    Descargar Boleta PDF
</a>
```

## 10.4. Contenido de la Boleta

El PDF generado incluye la siguiente información:

- Datos del cliente
- Lista de videojuegos adquiridos
- Precios unitarios y subtotal
- Fecha y hora de la transacción
- Identificador de la operación

## 10.5. Beneficios del Formato PDF

- **Portabilidad:** Los comprobantes pueden ser almacenados localmente por el usuario.
- **Impresión:** El formato PDF es apto para impresión sin pérdida de formato.
- **Validez documental:** Sirve como evidencia de compra para el usuario ante futuras consultas.

Esta funcionalidad añade una capa de profesionalismo y confianza en el proceso de compra, mejorando la experiencia del cliente final.

# 11. Recursos en Línea

A continuación, se presentan los enlaces oficiales relacionados con el proyecto **ESGames**, tanto del código fuente como del sistema desplegado para uso público.

## 11.1. Repositorio del Proyecto

El código fuente completo del sistema se encuentra disponible en la plataforma GitHub. Este repositorio contiene el backend desarrollado con Django, el frontend construido con Vue.js y Vite, así como documentación técnica complementaria.

- **Repositorio GitHub:**
  `https://github.com/RodrigoFFN/Proyecto_Final_gameStore.git`

## 11.2. Sitio Web en Producción

El sistema se encuentra desplegado en línea para uso y evaluación a través del servicio `Netlify`, permitiendo a los usuarios interactuar con la tienda, registrarse, explorar videojuegos y simular compras reales.

- **Enlace de acceso:**
  `https://esgamesproject.netlify.app/`

Ambos recursos permiten la validación funcional del sistema.

# 12. Adicionales

## 12.1. Docentes Responsables del Curso

Durante el desarrollo de este proyecto, el curso fue llevado en modalidad teórico–práctica, bajo la orientación de los siguientes docentes:

- **Teoría:** Mg. Richart Smith Escobedo Quispe
- **Laboratorio:** Mg. Carlo José Luis Corrales Delgado

Ambos docentes brindaron las bases conceptuales y técnicas necesarias para el desarrollo de aplicaciones web modernas utilizando tecnologías actuales del backend y frontend.

## 12.2. Autoevaluación

A continuación, se presenta la imagen correspondiente a la autoevaluación del equipo con relación a su desempeño en el proyecto:

| | 3 - Excelente | 2 - Bueno | 1.5 - Regular | 0.7 - Deficiente | Autocalificación Estudiante | Calificación Profesor |
|---|---|---|---|---|---|---|
| Producción de videos | Muy claro, coherente y bien estructurado. Alta calidad (audio, video, edición). Comunica totalmente el propósito del video. | Claro en general. Buena calidad con detalles menores. Comunica la mayoría. | Poco claro o con errores leves. Calidad media, edición básica. Comunica parcialmente. | Confuso o con errores graves. Baja calidad o sin edición. No comunica el objetivo. | 2 | |
| Backend en Producción | 100% funcional y sin errores. Desplegado correctamente y accesible. Seguridad y validación sólidas. | Funciona con pequeños errores. Desplegado con leves fallos. Buen manejo general. | Funciona parcialmente. Despliegue incompleto o poco estable. Validaciones básicas. | No funciona. No desplegado. Sin validaciones ni manejo de errores. | 3 | |
| Frontend en Producción | Interfaz atractiva, intuitiva y responsiva. Todas las funcionalidades operativas. Totalmente integrado y funcional. | Buena usabilidad con pocos detalles. Algunas fallas menores. Integración parcial, funciona bien. | Interfaz básica y poco intuitiva. Funciones básicas, faltan otras. Integración básica o incompleta. | Interfaz confusa o sin diseño. Fallos importantes o no funcional. No hay integración. | 2 | |
| Backup de BD | Backup completo y verificado. Documentado claramente. Proceso automatizado y funcional. | Backup completo sin verificación. Documentado con pocos detalles. Proceso semiautomatizado. | Backup incompleto. Documentación mínima. Manual pero funcional. | No se generó backup. Sin documentación. No funcional. | 3 | |
| Github Backend | Muy clara y profesional. Commits frecuentes, claros y con mensajes descriptivos. Uso correcto de ramas y merges. | Bien organizada. Mensajes adecuados. Uso parcial de ramas. | Algo desorganizada. Pocos commits o mensajes vagos. Uso mínimo de ramas. | Confusa o caótica. Sin estructura ni control. Todo en main/master sin control. | 1.5 | |
| Github Frontend | Muy bien estructurado. Detallado y constante. Buen uso de versiones y ramas. | Estructura adecuada. Constante pero sin detalle. Uso limitado de control. | Organización básica. Escaso o mal documentado. Versión única y cambios directos. | Desorganizado. Desordenado o ausente. Sin control de versiones. | 2 | |
| Informe Final | Muy completo, análisis crítico y detallado. Excelente redacción, sin errores. Incluye capturas, enlaces, resultados, etc. Se autocalifica. | Completo con buen análisis. Pocos errores menores. Incluye algunas evidencias. | Superficial pero cumple. Algunos errores importantes. Pocas evidencias. | Incompleto o sin análisis. Muchos errores. No incluye evidencias. | 3 | |
| | | | | TOTAL | 14.5 | |

Figura 3: Imagen de autoevaluación del estudiante

# 13. Conclusiones

El desarrollo del sistema **ESGames** ha permitido consolidar una solución web funcional e integral para la gestión de una tienda digital de videojuegos. A lo largo del proyecto, se logró implementar satisfactoriamente tanto el frontend como el backend, integrando múltiples tecnologías modernas que permiten una experiencia fluida, segura y modular para el usuario final.

Entre los principales logros del sistema se destacan:

- La creación de una interfaz de usuario dinámica y responsiva utilizando `Vue.js`, desplegada eficientemente en `Netlify`.
- La implementación de una API RESTful robusta mediante `Django REST Framework`, con autenticación, control de acceso, validación de reglas de negocio y persistencia de datos.
- La integración con `Supabase` como sistema de base de datos relacional, y la utilización de `Render` para el despliegue del backend.
- La generación de boletas de compra en formato PDF, ofreciendo un respaldo formal a los usuarios por cada transacción realizada.

Asimismo, se destaca el valor que la aplicación entrega al usuario, permitiéndole explorar un catálogo de videojuegos, realizar compras simuladas, gestionar su biblioteca personal y dejar valoraciones útiles para otros compradores. Esto hace de ESGames una plataforma versátil y con potencial de crecimiento.

Como trabajo futuro se plantea:

- Integrar pasarelas de pago reales.
- Añadir notificaciones por correo electrónico tras cada compra.
- Incorporar un sistema de roles para la administración y gestión del inventario.
- Ampliar el catálogo con funcionalidades de búsqueda avanzada y recomendaciones.

En conclusión, el proyecto ha cumplido los objetivos propuestos, ofreciendo una base sólida para una plataforma de e-commerce especializada en videojuegos, con miras a seguir evolucionando en futuras versiones.

# Referencias

[1] Django Software Foundation. (2024). *Django Documentation (v4.x)*. Recuperado de `https://docs.djangoproject.com/`

[2] Encode. (2024). *Django REST Framework*. Recuperado de `https://www.django-rest-framework.org/`

[3] Vue.js. (2024). *Vue 3 Documentation*. Recuperado de `https://vuejs.org/`

[4] Vite. (2024). *Vite – Next Generation Frontend Tooling*. Recuperado de `https://vitejs.dev/`

[5] Netlify. (2024). *Netlify Documentation*. Recuperado de `https://docs.netlify.com/`

[6] Render. (2024). *Render Docs: Django Deployment*. Recuperado de `https://render.com/docs`

[7] Supabase. (2024). *Supabase Documentation*. Recuperado de `https://supabase.com/docs`

[8] WeasyPrint. (2024). *WeasyPrint Documentation*. Recuperado de `https://weasyprint.readthedocs.io/`

[9] Pinia. (2024). *Pinia – Vue Store*. Recuperado de `https://pinia.vuejs.org/`

[10] Mozilla Developer Network (MDN). (2024). *Fetch API*. Recuperado de `https://developer.mozilla.org/es/docs/Web/API/Fetch_API`