



OSIRIS

Sistema de Administración de Juegos

Proyecto de fin de ciclo formativo

AUTORES

Jesús Gómez Villaboa
Ignacio García García
Rubén de las Heras

TUTORA DEL PROYECTO

Natalia Garcêa Giordano

CICLO FORMATIVO DE GRADO SUPERIOR

2º de Desarrollo de Aplicaciones Multiplataforma

FECHA

8 de Junio de 2025



ÍNDICE

Introducción y Justificación del Proyecto.....	4
Antecedentes.....	4
Objetivos.....	5
Diseño.....	6
Especificación de Requisitos.....	6
Análisis.....	7
Diagrama de pantallas.....	8
Diagrama de componentes.....	9
Diagrama Entidad-Relación.....	10
Diagramas de secuencia.....	11
Diagrama de casos de uso.....	12
Diagrama de clases.....	14
Diagrama de endpoints de la API.....	15
Endpoints de los juegos.....	15
Endpoints de la tienda.....	17
Endpoints de la biblioteca.....	19
Endpoints de Usuarios.....	21
Endpoints de perfiles de usuario.....	23
Endpoint de login.....	25
Endpoint de la lista de amigos.....	25
Endpoint de los chats.....	27
Arquitectura de servicio. API.....	30
Servidor.....	30
Paquete Raíz.....	30
Paquete API_Rest.....	30
Paquete BD.....	31
Paquete static.....	31
Paquete templates.....	31
Cliente.....	32
Paquete Raíz.....	32
Paquete Classes.....	32
• Usuario.cs.....	32
• LocalStorage.cs.....	32
• Config.cs/AppTheme.cs.....	32
• Notificacion.cs.....	33
Paquete Pages.....	33
• paginaTienda.....	33

Osiris - Índice

• paginaRecargarSaldo.....	34
• paginaPerfil.....	35
• paginaJuegoTienda.....	35
• paginaBiblioteca.....	37
• paginaAmigos.....	38
Paquete res.....	40
• imagenes.....	40
• library.....	40
• login.....	40
• store.....	40
Paquete UserControls.....	40
• Cabecera.....	41
• LoginControl.....	41
• RegistroControl.....	41
Paquete Windows.....	41
• ErrorWindow.....	41
• LoginWindow.....	41
• VEliminarJuego.....	42
• VentanaRecarga.....	42
• VentanaSinJuego.....	42
• VJuegoAgregadoCorrecto.....	42
• VNombreVacio.....	42
• VPerfilActualizado.....	42
• VRegistroExitoso.....	42
Tecnologías.....	42
Servidor.....	43
Python.....	43
Visual Studio Code.....	43
Flask.....	43
SQLAlchemy.....	43
HTML, CSS y JavaScript.....	44
Logging - Logging facility for Python.....	44
Cliente.....	44
C# con WPF (Windows Presentation Foundation).....	44
Visual Studio.....	45
HttpClient.....	45
NewtonsoftJson:.....	45
Aplicaciones y servicios externos.....	45
Github.....	45
Imgur.....	45
PostgreSQL.....	45
Postman.....	46
SocketIO.....	46
Adobe Photoshop.....	46
draw.io.....	46

Osiris - Índice

StarUML.....	47
dbDiagram.io.....	47
Cronograma.....	48
Desarrollo.....	49
Servidor.....	49
Flask.....	49
SQLAlchemy.....	52
Sistema de logs.....	53
Cliente.....	56
Estructura.....	56
Interfaz y Experiencia de Usuario.....	56
Desarrollo de la aplicación cliente.....	56
Interfaz gráfica de la aplicación.....	59
Tienda:.....	61
Juego de la tienda:.....	64
Amigos:.....	65
Manejo de logs.....	69
Experiencia y Problemas durante el desarrollo.....	69
Resultados.....	73
Servidor.....	73
Cliente.....	77
Discusión.....	88
Conclusión.....	89
Anexo.....	90
Hoja de pruebas.....	90
Bibliografía.....	90

Introducción y Justificación del Proyecto.

Actualmente, la distribución digital de videojuegos representa uno de los sectores más lucrativos de la industria del entretenimiento. Según reportes recientes, este mercado superó los 200 mil millones de dólares en 2023, debido a que cada vez más gente tiene acceso a internet, a la popularización de los juegos multiplataforma y al auge de modelos de negocio innovadores. Plataformas líderes como Steam, Epic Games Store y GOG han consolidado un ecosistema donde millones de usuarios compran, juegan, chatean y comparten contenido sobre sus videojuegos favoritos. Sin embargo, no todos los juegos se encuentran disponibles en todas las plataformas, por lo que los usuarios, acaban usando varios de estos servicios dependiendo de lo que jueguen en ese momento, lo que abre oportunidades para propuestas innovadoras que permitan competir en ese ámbito.

Este proyecto puede resultar una alternativa atractiva frente al resto de aplicaciones del mercado ya que tenemos un sistema con el que los clientes no tardarán mucho en familiarizarse puesto que sus funciones son similares a las de la competencia, es muy liviana, permite acceder a tu biblioteca de juegos en caso de no tener conexión, tiene una interfaz que proporciona una gran experiencia de usuario y permite al usuario agregar sus propios juegos para poder lanzarlos desde la biblioteca, abriendo la posibilidad de que agregue los juegos que tiene distribuidos en el resto de sistemas, para que el usuario pueda tener todos los juegos de su ordenador en una misma aplicación centralizada.

Antecedentes.

En 2003 Valve lanzó al mercado Steam, el gestor de videojuegos de ordenador más importante actualmente, hasta entonces. Cada videojuego se descargaba de su propia página web o se distribuía en físico. Durante muchos años, Steam fue ganando popularidad y cada vez más empresas publican sus videojuegos en ella.

En 2012 Ubisoft y EA lanzaron al mercado sus gestores de videojuegos propios, Uplay y Origin respectivamente, los cuales incluían juegos exclusivos de sus compañías. Por lo tanto, dividían a la comunidad, ya que si querías jugar un juego de Ubisoft (Assassin's Creed, Far Cry, etc) el usuario debía comprarlo y jugarlo en Uplay, y si querías un juego de EA (FIFA, Battlefield, etc.) debías comprarlo y jugarlo en Origin. Esto provocaba que los jugadores que habitualmente jugaban a videojuegos tuvieran que usar varios de estos sistemas, y abrir uno u otro en función del juego que querían jugar.

En 2018 esta situación empeoró, ya que apareció Epic Games, teniendo como juego exclusivo a Fortnite, un juego desarrollado por ellos, que se ha convertido en uno de los videojuegos más populares a día de hoy, la situación volvió a empeorar en 2019, cuando Microsoft decidió sacar un servicio llamado Game Pass, que volvió bastante atractiva la aplicación de Xbox en PC, la cual poca gente usaba.

A día de hoy, las plataformas más usadas son Steam y Epic Games, pero los usuarios siguen teniendo que verse obligados a usar una plataforma u otra dependiendo de sus necesidades, es por eso que hemos decidido desarrollar una aplicación similar, que permita centralizar las bibliotecas de estos sistemas en una sola, para poder ejecutar diferentes videojuegos pertenecientes a diferentes aplicaciones gestoras de videojuegos desde un mismo lugar.

Osiris - Antecedentes

Debido a que queremos trabajar en un gestor de videojuegos, es importante estudiar qué tecnologías han usado las aplicaciones más importantes de este sector, es decir, Steam y Epic Games.

Steam utiliza PostgreSQL principalmente para su base de datos y C++ para su servidor, y Epic Games, utiliza PostgreSQL para su base de datos y C++ y Python para su servidor. Viendo que ambos usaban PostgreSQL, decidimos utilizar la misma base de datos, en el caso del lenguaje de programación, preferimos evitar C++, ya que Epic también usaba Python, y nos llamaba más la atención aprender Python que aprender C++, y de cara al mundo laboral nos podía ser más útil aprender Python, ya que este es de los lenguajes más utilizados y demandados.

Objetivos

Este proyecto busca desarrollar una aplicación de gestión de videojuegos que tenga las funciones esenciales de la competencia y permita unificar las bibliotecas de juegos que tenga el usuario en el resto de plataformas, en una sola. El sistema propuesto integrará módulos clave:

- Tienda virtual con catálogo dinámico y simulación de compra.
- Biblioteca que permita al usuario ejecutar los juegos que tenga almacenados en ella, el usuario podrá agregar sus propios juegos, así como poder importar los juegos que tenga en otras plataformas.
- Sistema de usuarios con perfiles interactivos, y un sistema social que incluirá listas de amigos y chat integrado.

El objetivo final es ofrecer una alternativa que permita facilitar la portabilidad de bibliotecas entre plataformas para que el usuario no tenga la necesidad de iniciar una plataforma u otra dependiendo del juego que vaya a ejecutar.

Diseño

Especificación de Requisitos

- Requisitos Funcionales
 - Gestión de usuarios
 - **(REQ-001)** - Los usuarios podrán registrarse para posteriormente poder iniciar sesión en la aplicación con una cuenta propia.
 - **(REQ-002)** - Los usuarios podrán agregar a otros usuarios.
 - **(REQ-003)** - Los usuarios dispondrán de una lista de amigos donde ver a los usuarios agregados.
 - **(REQ-004)** - Los usuarios podrán chatear entre ellos.
 - **(REQ-005)** - Los usuarios podrán personalizar el perfil modificando su foto de perfil, su nombre de usuario o la descripción.
 - Gestión de bibliotecas
 - **(REQ-006)** - Los usuarios podrán acceder a sus juegos adquiridos de la tienda desde cualquier dispositivo que sea capaz de ejecutar la aplicación.
 - **(REQ-007)** - Los usuarios serán capaces de agregar el ejecutable de sus juegos a la su biblioteca.
 - **(REQ-008)** - Los usuarios podrán ejecutar cualquier juego que tengan disponible desde su biblioteca.
 - **(REQ-009)** - Los usuarios podrán eliminar las aplicaciones de sus bibliotecas.
 - Gestión de tienda
 - **(REQ-010)** - Los usuarios podrán buscar cualquier juego incluido en la base de datos.
 - **(REQ-011)** - Los usuarios podrán visualizar los datos relacionados con dichos juegos.
 - **(REQ-012)** - Los usuarios podrán simular el pago y obtención de dichos juegos.
- Requisitos No Funcionales
 - Rendimiento

Osiris - Diseño

- **(REQ-013)** - La aplicación deberá responder a las solicitudes de los usuarios en menos de 2 segundos en la mayoría de los casos.
- Seguridad
 - **(REQ-014)** - La aplicación deberá cifrar correctamente las contraseñas.
- Usabilidad
 - **(REQ-015)** - La aplicación no debe permitir al usuario dejar campos esenciales vacíos, como por ejemplo la contraseña.
 - **(REQ-016)** - La aplicación tendrá una interfaz intuitiva cuyo tiempo de aprendizaje será menor a 10 minutos para usuarios sin experiencia.

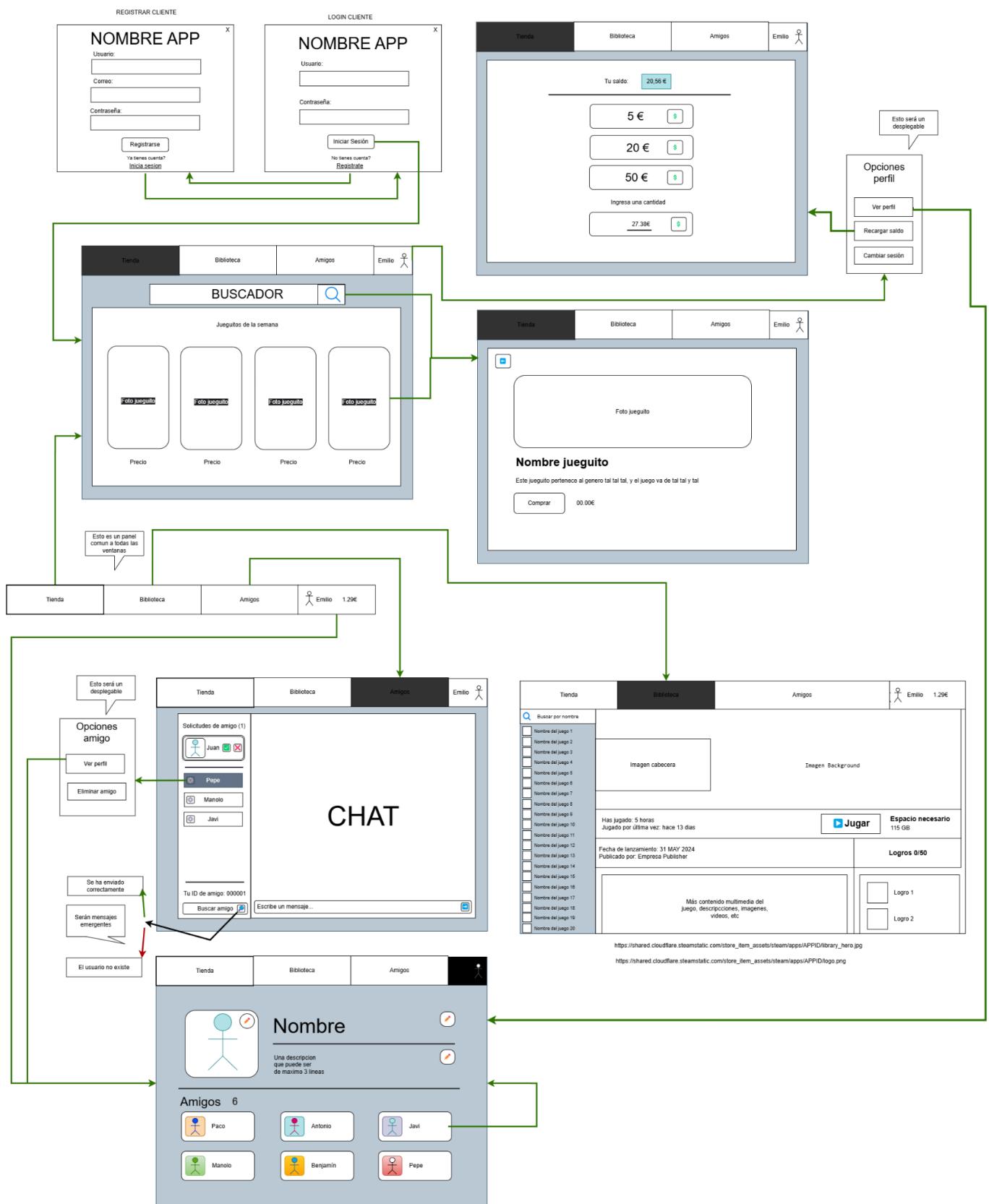
Análisis

Durante la fase de Análisis del proyecto, nos vimos obligados a realizar una gran cantidad de diagramas debido a que fué la primera vez que nos enfrentamos a una aplicación de esta magnitud, y en ese momento nos faltaba experiencia con muchas de las tecnologías que íbamos a utilizar para el proyecto, al no tener claro el hecho de cómo íbamos a realizar la aplicación, estuvimos 3 largas semanas estudiando las tecnologías a utilizar, creando diagramas y buscando cualquier tipo de asesoramiento en personas de nuestro entorno.

Los diagramas que llegamos a realizar fueron los siguientes:

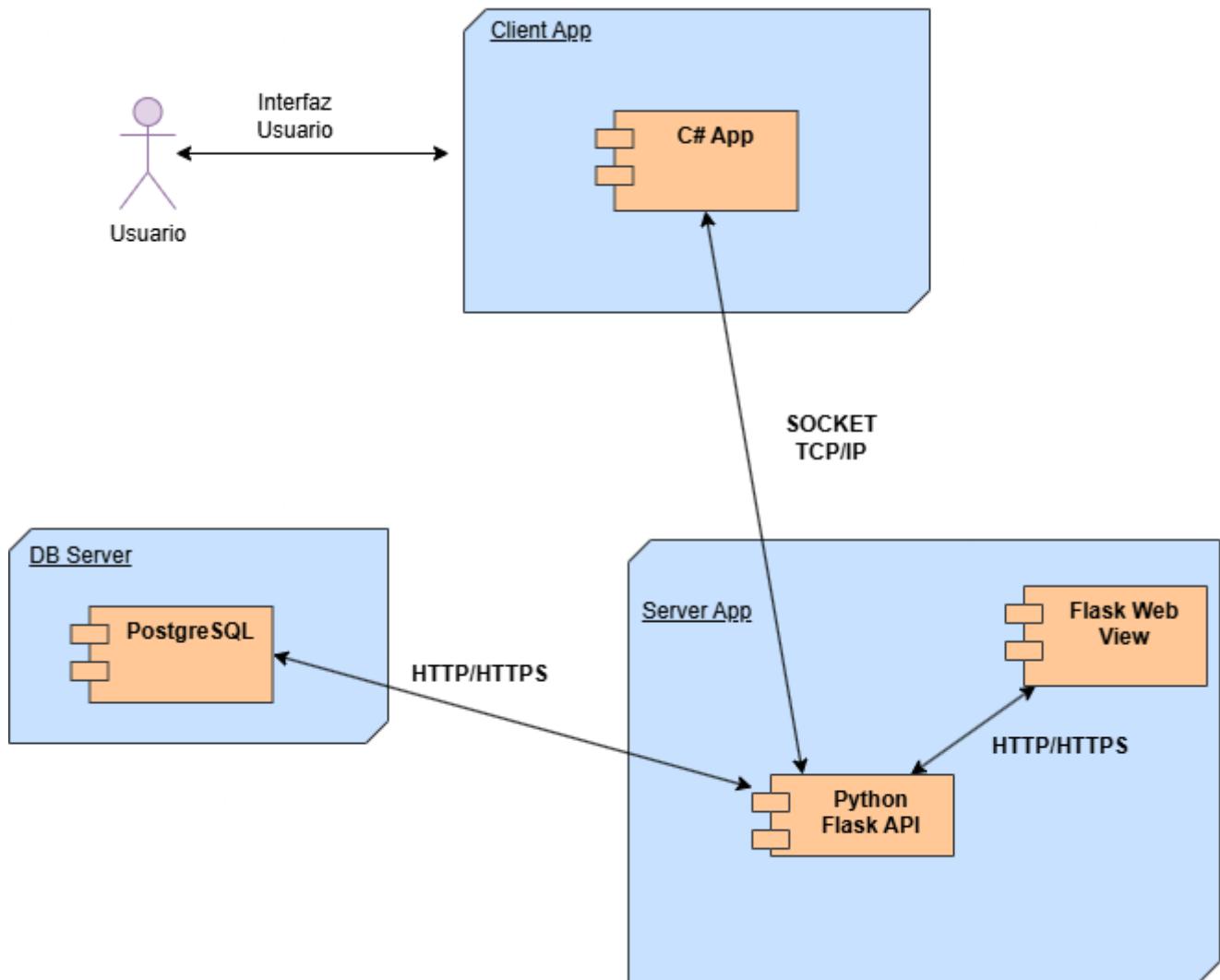
Osiris - Diseño

Diagrama de pantallas



Osiris - Diseño

Diagrama de componentes



Osiris - Diseño

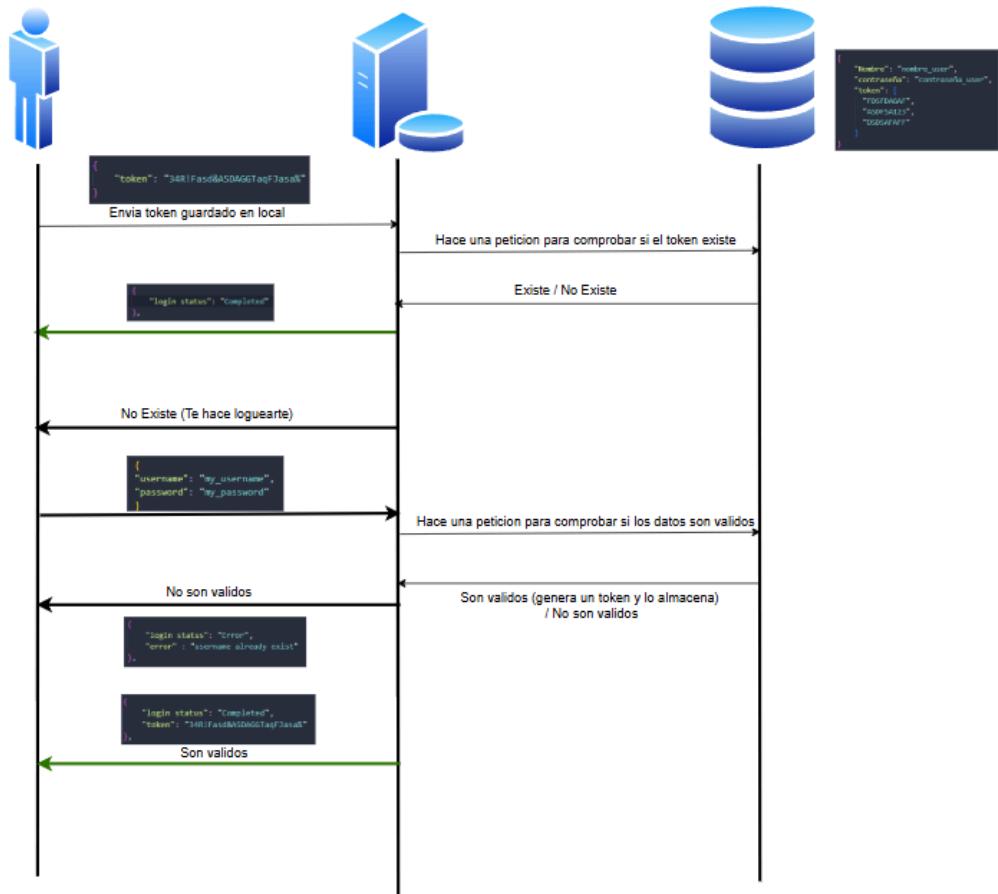
Diagrama Entidad-Relación



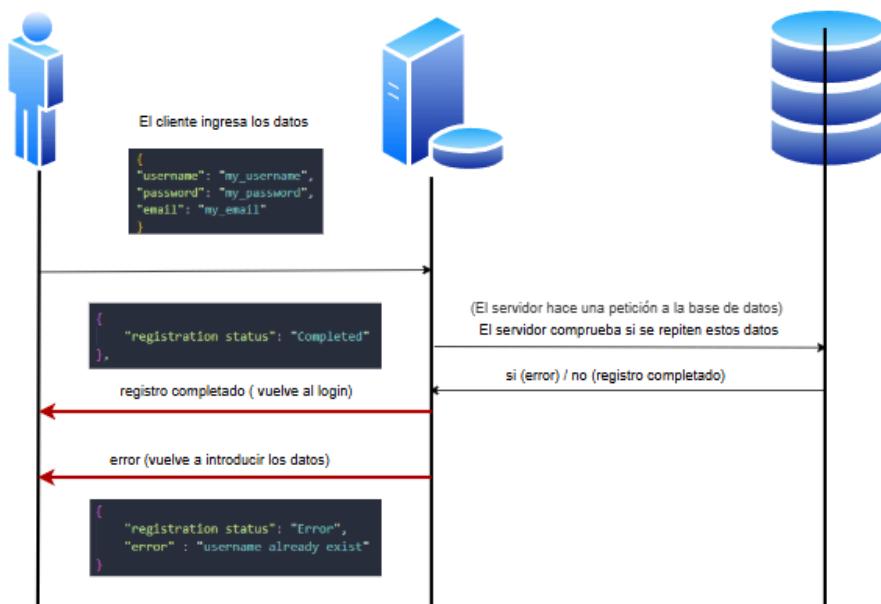
Osiris - Diseño

Diagramas de secuencia

Login

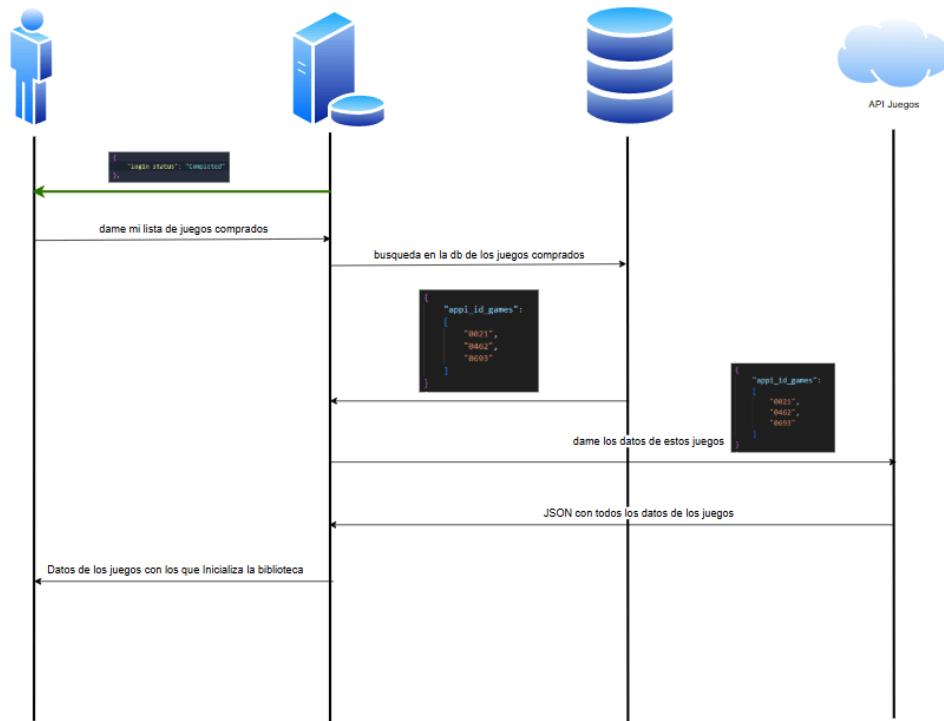


Registro

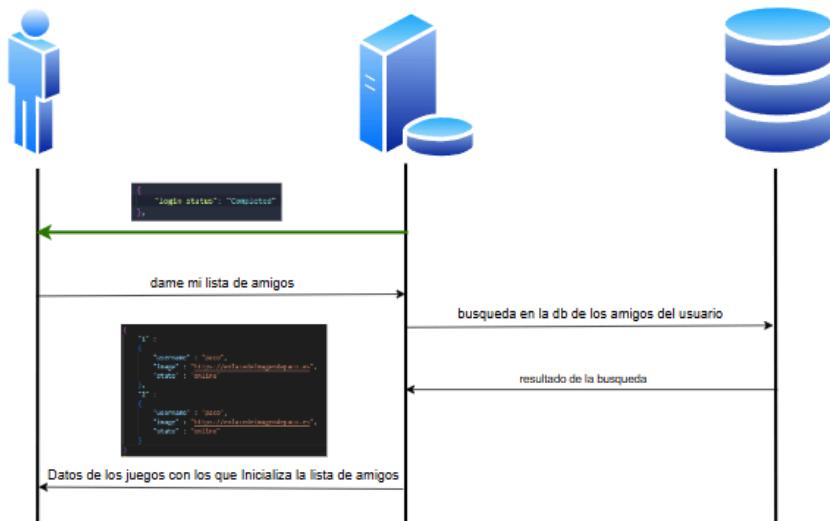


Osiris - Diseño

Biblioteca

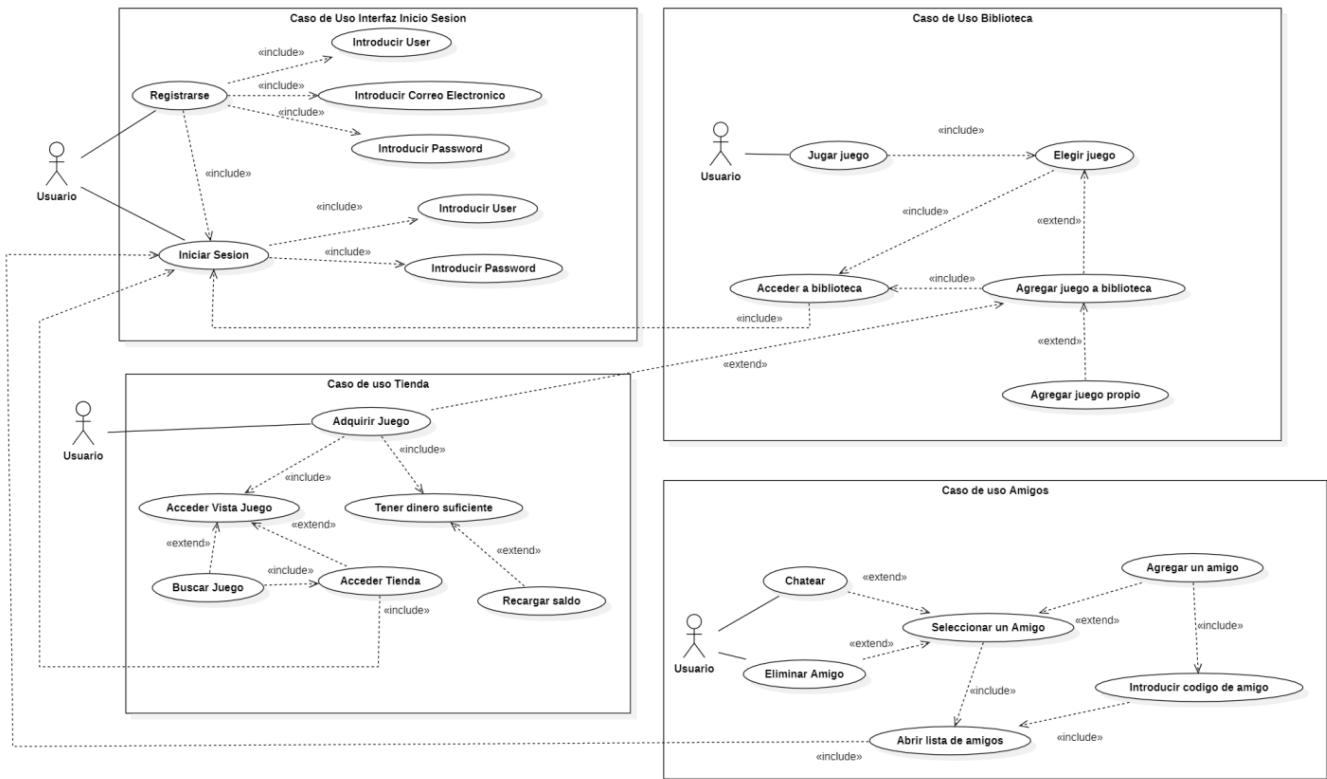


Amigos



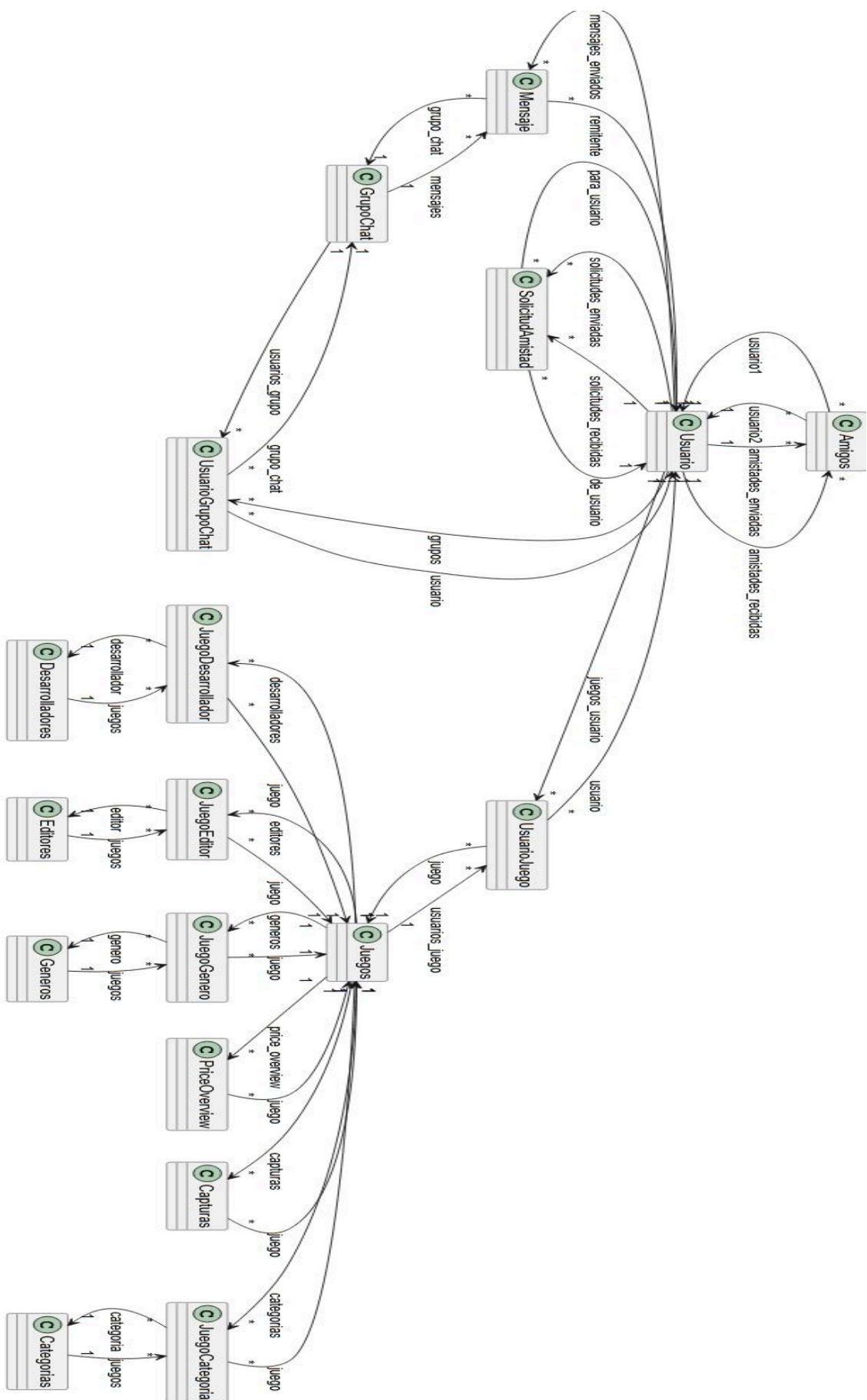
Osiris - Diseño

Diagrama de casos de uso



Osiris - Diseño

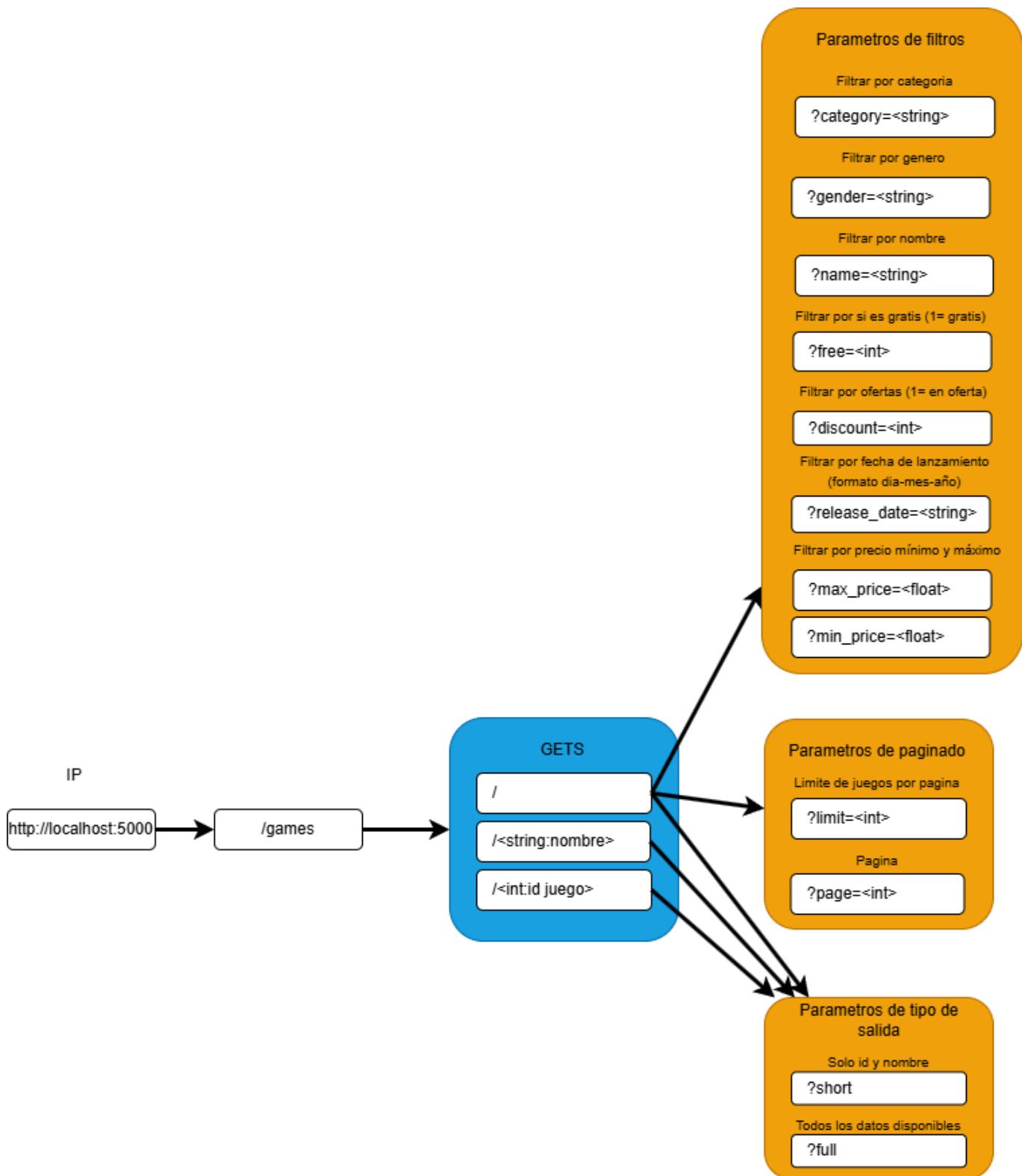
Diagrama de clases



Osiris - Diseño

Diagrama de endpoints de la API

Endpoints de los juegos

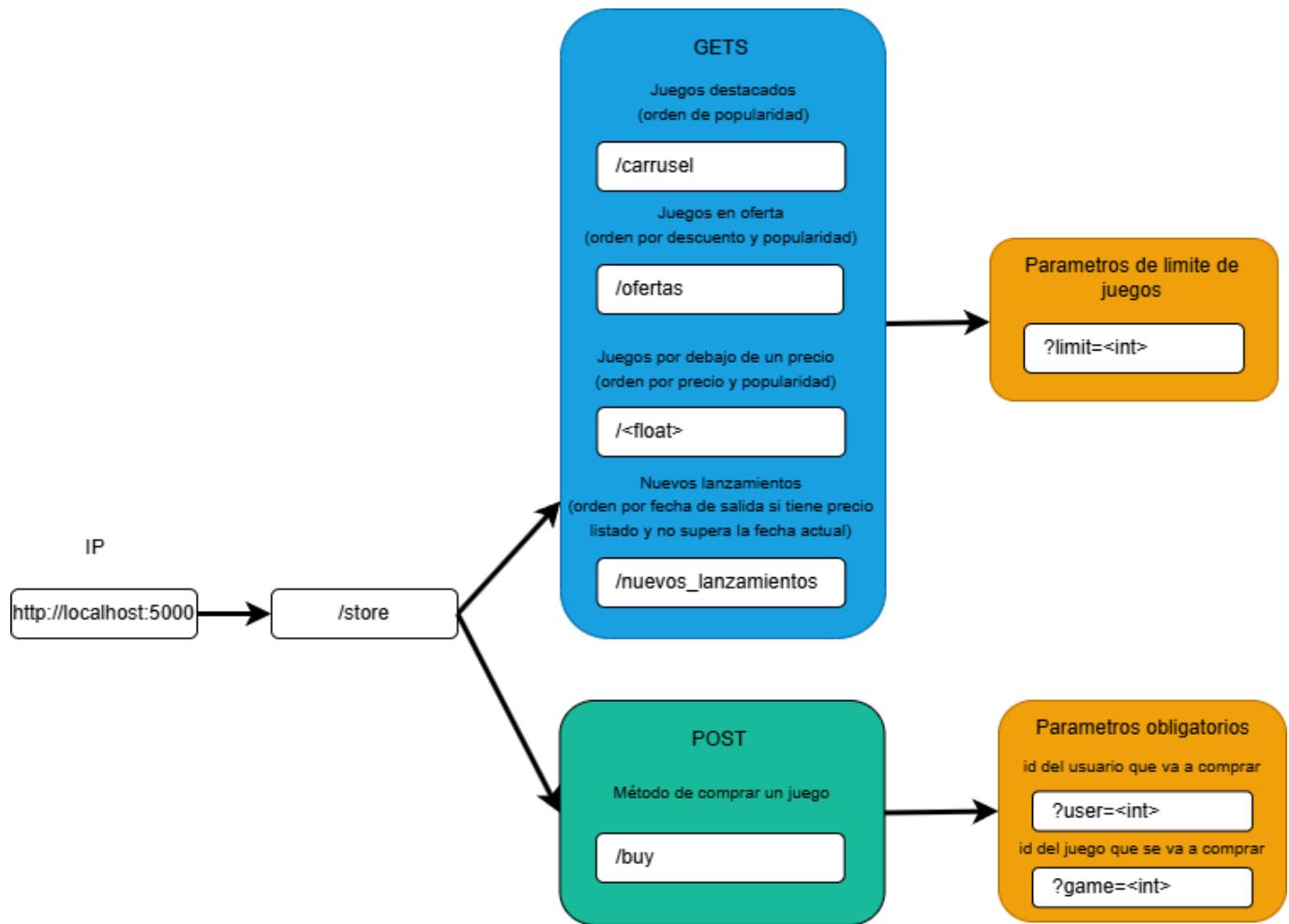


Osiris - Diseño

Método	Método	Descripción	Parámetros (query)
/games/	GET	Devuelve una lista de juegos, con soporte para filtros, paginación y modos	<ul style="list-style-type: none"> - <code>short</code> (bool): Devuelve solo <code>id</code> y <code>nombre</code>. - <code>full</code> (bool): Devuelve todos los datos posibles. - <code>page</code> (int): Número de página (default: 1). - <code>limit</code> (int): Cantidad de juegos por página (default: 50). Filtros: <ul style="list-style-type: none"> - <code>name</code> (str): Nombre parcial del juego. - <code>category</code> (str[]) (repetible): Filtra por categoría (e.g. <code>Multi-player</code>). - <code>gender</code> (str[]) (repetible): Filtra por género (e.g. <code>Action</code>). - <code>free</code> (int): 1 = gratis, 0 = de pago. - <code>discount</code> (int): 1 = con descuento. - <code>max_price</code> (float): Precio máximo. - <code>min_price</code> (float): Precio mínimo. - <code>release_date</code> (str): Fecha mínima de salida (dd-mm-YYYY).
/games/<string :game_name>	GET	Devuelve juegos cuyo nombre contenga <code>game_name</code>	<ul style="list-style-type: none"> - <code>short</code> (bool): Devuelve solo <code>id</code> y <code>nombre</code>. - <code>full</code> (bool): Devuelve todos los datos posibles.
/games/<int:id_game>	GET	Devuelve el juego con <code>id_game</code> exacto	<ul style="list-style-type: none"> - <code>short</code> (bool): Devuelve solo <code>id</code> y <code>nombre</code>. - <code>full</code> (bool): Devuelve todos los datos posibles.

Osiris - Diseño

Endpoints de la tienda



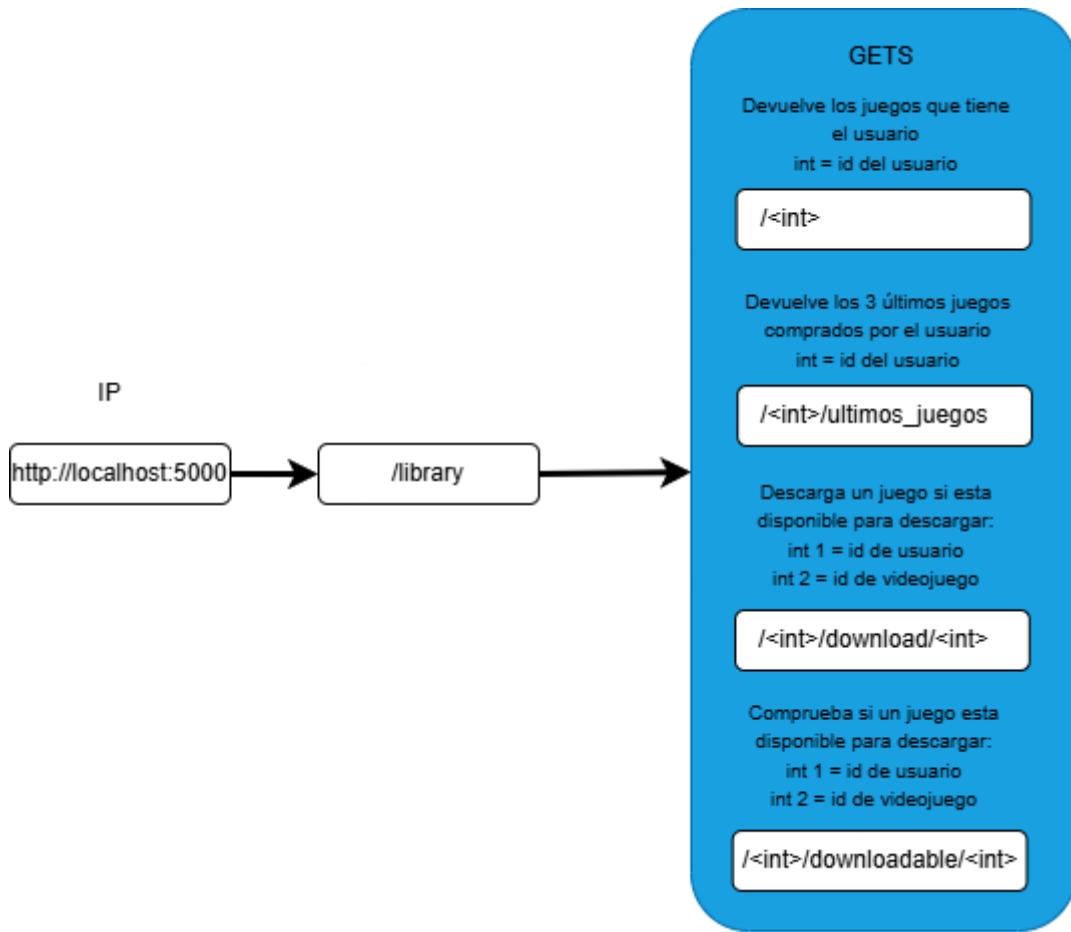
Endpoint	Método	Descripción	Parámetros de entrada	Respuesta esperada
/store/carrusel	GET	Devuelve una lista de juegos para el carrusel de la tienda, filtrados por popularidad y con al menos 4 capturas.	<code>limit</code> (int, opcional): número máximo de juegos a devolver (por defecto 10)	JSON con lista de juegos: <code>{"juegos": [{ ... }]}</code>

Osiris - Diseño

/store/ofertas	GET	Devuelve una lista de juegos en oferta, ordenados por popularidad y porcentaje de descuento.	<code>limit</code> (int, opcional): número máximo de juegos a devolver (por defecto 6)	JSON con lista de juegos: { "juegos": [{ ... }] }
/store/<float:price>	GET	Devuelve juegos con precio final (con descuento aplicado) por debajo de un valor dado.	<code>price</code> (float): precio máximo en euros <code>limit</code> (int, opcional): número máximo de juegos a devolver (por defecto 4)	JSON con lista de juegos: { "juegos": [{ ... }] }
/store/nuevos_lanzamientos	GET	Devuelve una lista de juegos no gratuitos ordenados por fecha de lanzamiento descendente.	<code>limit</code> (int, opcional): número máximo de juegos a devolver (por defecto 10)	JSON con lista de juegos: { "juegos": [{ ... }] }
/store/buy	POST	Permite a un usuario comprar un juego, si no lo posee ya.	<code>user</code> (int): ID del usuario <code>game</code> (int): ID del juego	JSON con resultado de la operación: { "success": true/false, "message": "..." }

Osiris - Diseño

Endpoints de la biblioteca



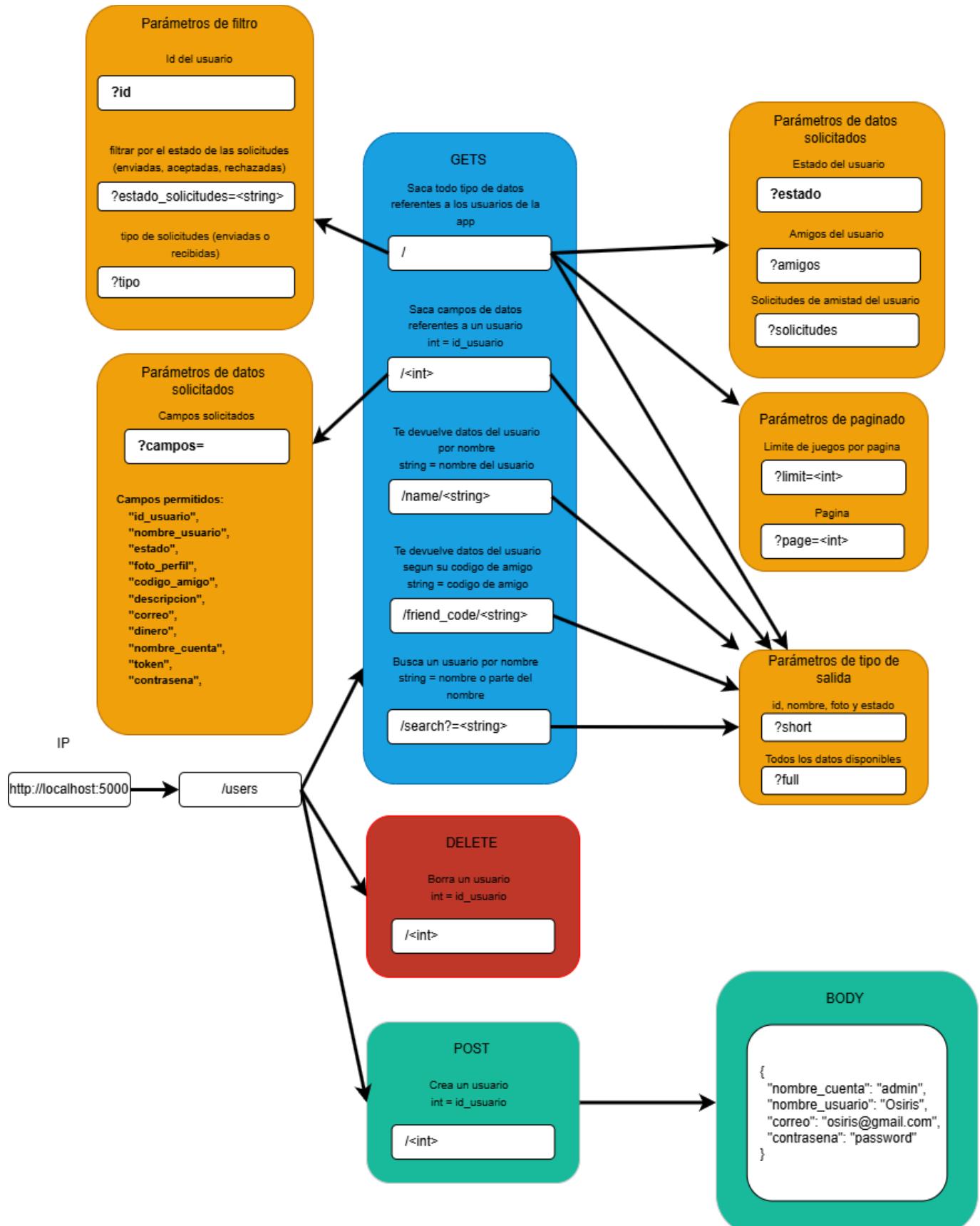
Endpoint	Método	Descripción	Parámetros de entrada	Respuesta esperada
<code>/library/<id_user></code>	GET	Devuelve la biblioteca de juegos del usuario con ID <code><id_user></code> .	<code>id_user</code> (int): ID del usuario	JSON con una lista de juegos: <code>{ "juegos": [{ "app_id", "nombre", "captura", "header" }, ...] }</code>

Osiris - Diseño

/library/<id_user>/ultimos_juegos	GET	Devuelve los 3 juegos más recientes comprados por el usuario.	<code>id_user</code> (int): ID del usuario	JSON con lista de los últimos juegos: <pre>{ "ultimas_compras": [{ "app_id", "nombre", "captura", "header", "fecha_compra" }, ...] }</pre>
/library/<id_user>/download/<app_id>	GET	Permite la descarga de un juego específico si el usuario lo posee y el archivo existe en el servidor.	<code>id_user</code> (int): ID del usuario, <code>app_id</code> (int): ID del juego	Archivo .zip del juego si es válido; JSON con error en caso de fallo: { "error": "mensaje" } (403 si no posee el juego, 404 si el archivo no existe, 500 si hay error en el servidor)
/library/<id_user>/downloadable/<app_id>	GET	Comprueba si un juego específico es descargable (si el archivo existe en el servidor).	<code>id_user</code> (int): ID del usuario, <code>app_id</code> (int): ID del juego	JSON con estado: { "descargable": true/false } o { "error": "mensaje", "detalle": "descripción" } en caso de error (500)

Osiris - Diseño

Endpoints de Usuarios

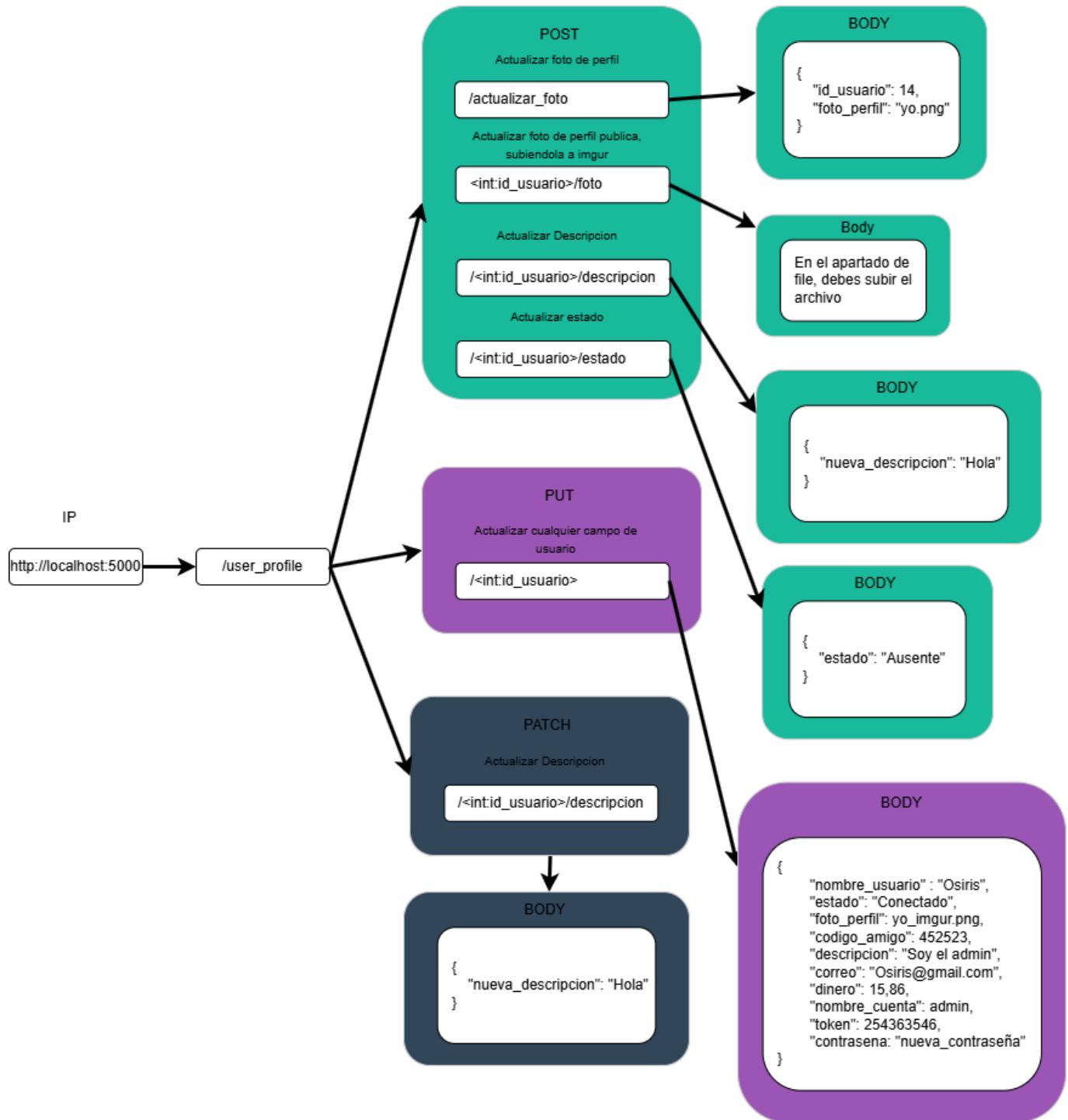


Osiris - Diseño

Endpoint	Método	Descripción	Parámetros / Query
/users/	GET	Obtener lista de usuarios. Soporta paginación y filtros.	limit, page, short, full, estado, id, amigos, solicitudes, estado_solicitudes, tipo
/users/	POST	Crear un nuevo usuario con validaciones.	JSON: nombre_cuenta, correo, contrasena (obligatorios), foto_perfil, descripcion, nombre_usuario (opcionales)
/users/<int:id_usuario>	GET	Obtener datos de un usuario específico.	short, campos (lista CSV de campos permitidos)
/users/<int:id_usuario>	DELETE	Eliminar usuario por ID.	
/users/<int:user_id>/recargar	PUT	Recargar saldo del usuario.	Query param: cantidad
/users/search	GET	Buscar usuarios por nombre (busca primero los que empiezan y luego los que contienen la cadena).	nombre (obligatorio), short
/users/name/<string:nombre>	GET	Obtener usuario por nombre de usuario exacto.	short
/users/friend_code/<string:friend_code>	GET	Obtener usuario por código de amigo único.	short

Osiris - Diseño

Endpoints de perfiles de usuario



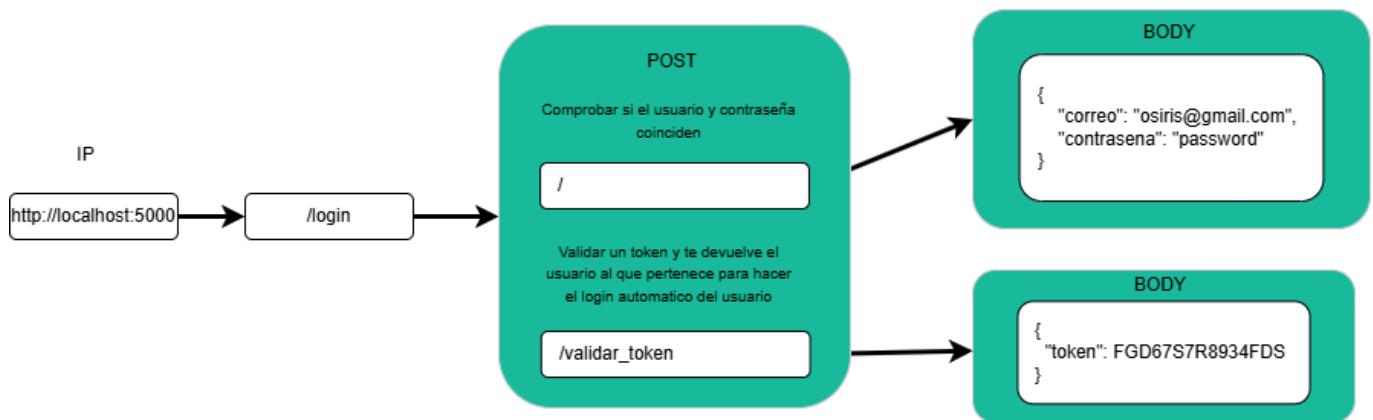
Endpoint	Método	Descripción	Datos / Parámetros
----------	--------	-------------	--------------------

Osiris - Diseño

/actualizar_foto	POST	Actualiza la foto de perfil de un usuario (recibe <code>id_usuario</code> y <code>foto_perfil</code> como form-data).	Form data: <code>id_usuario</code> , <code>foto_perfil</code> (URL nueva)
/<int:id_usuario>/foto	POST	Cambia la foto de perfil subiendo una imagen a Imgur y actualiza la URL en el usuario.	Archivo: <code>foto</code> (imagen)
/<int:user_id>	PATCH	Actualiza campos permitidos de un usuario (como <code>nombre_usuario</code> , <code>correo</code> , <code>contrasena</code> , <code>descripcion</code> , <code>dinero</code> , etc).	JSON con campos a actualizar (ej. <code>"nombre_usuario": "Nuevo nombre"</code> , <code>"correo": "nuevo@mail.com"</code>)
/<int:id_usuario>/descripcion	PUT	Cambia la descripción del usuario.	JSON: { <code>"nueva_descripcion": "texto nuevo"</code> }
/<int:id_usuario>/foto	GET	Muestra un formulario HTML para cambiar la foto (uso para pruebas en navegador).	Parámetro URL: <code>id_usuario</code>

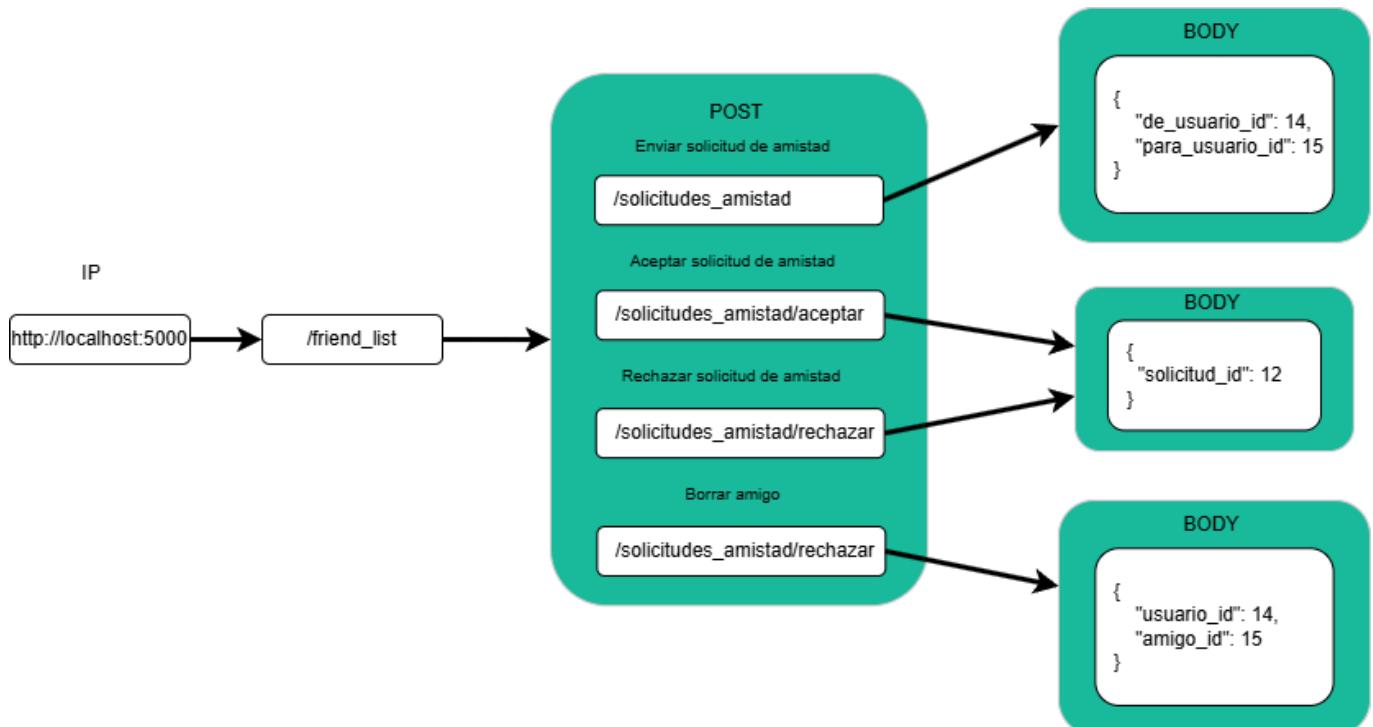
Osiris - Diseño

Endpoint de login



Endpoint	Método	Descripción	Datos / Parámetros
/login/	POST	Login de usuario verificando correo y contraseña.	JSON con correo y contraseña .

Endpoint de la lista de amigos

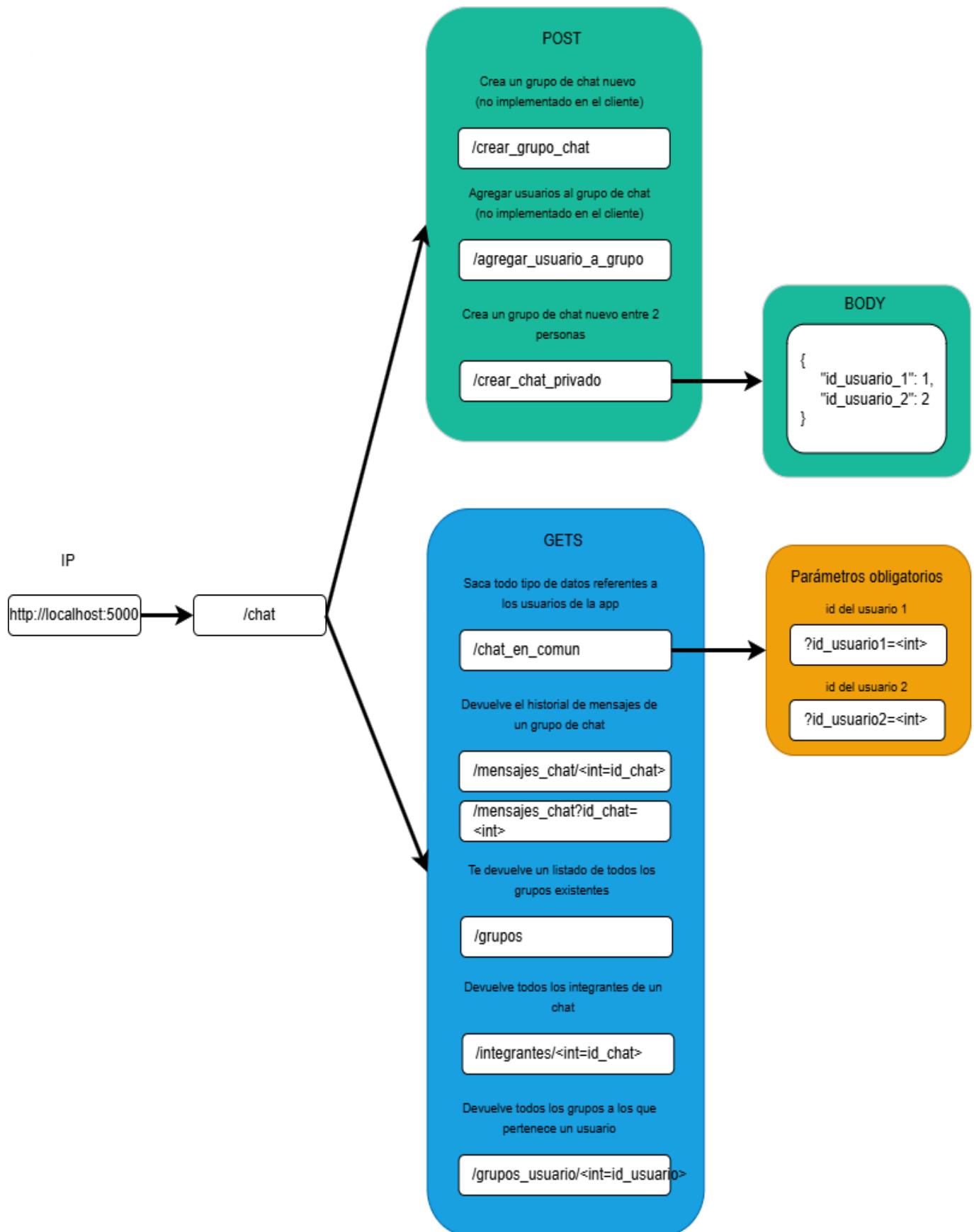


Osiris - Diseño

Endpoint	Método	Descripción	Datos JSON / Parámetros
/friend_list/solicitudes_amistad	POST	Envía una solicitud de amistad desde un usuario a otro.	JSON con de_usuario_id y para_usuario_id
/friend_list/solicitudes_amistad/aceptar	POST	Acepta una solicitud de amistad pendiente y crea la relación.	JSON con solicitud_id
/friend_list/solicitudes_amistad/rechazar	POST	Rechaza una solicitud de amistad pendiente.	JSON con solicitud_id
/friend_list/amigos/borrar	POST	Elimina una amistad existente entre dos usuarios y sus solicitudes.	JSON con usuario_id y amigo_id

Osiris - Diseño

Endpoint de los chats



Osiris - Diseño

Endpoint / Evento Socket	Método / Tipo	Descripción	Datos JSON / Parámetros
/crear_grupo_chat	POST	Crea un nuevo grupo de chat vacío.	Ninguno (vacío)
/agregar_usuario_a_grupo	POST	Agrega un usuario a un grupo de chat si no está ya en él.	JSON con <code>id_usuario</code> (int), <code>id_chat</code> (int)
/grupos	GET	Lista todos los grupos de chat existentes.	Ninguno
/mensajes/<id_chat>	GET	Obtiene todos los mensajes del grupo de chat indicado.	Parámetro URL: <code>id_chat</code> (int)
/integrantes/<id_chat>	GET	Lista todos los usuarios integrantes del grupo de chat.	Parámetro URL: <code>id_chat</code> (int)
/grupos_usuario/<id_usuario>	GET	Lista los grupos de chat a los que pertenece un usuario.	Parámetro URL: <code>id_usuario</code> (int)
join_chat	Socket.IO event	Permite a un usuario unirse a un grupo (sala) de chat.	JSON con <code>id_chat</code> (int), <code>id_usuario</code> (int)
leave_chat	Socket.IO event	Permite a un usuario salir de un grupo (sala) de chat.	JSON con <code>id_chat</code> (int), <code>id_usuario</code> (int)
enviar_mensaje	Socket.IO event	Envía un mensaje de un usuario a un grupo, y lo guarda.	JSON con <code>id_usuario_remitente</code> (int), <code>id_chat</code> (int), <code>mensaje</code> (str)

Osiris - Diseño

connect	Socket.IO event	Evento de conexión del cliente al servidor Socket.IO.	Ninguno
disconnect	Socket.IO event	Evento de desconexión del cliente del servidor Socket.IO.	Ninguno

Arquitectura de servicio. API

La API de Osiris es el núcleo de comunicación de la aplicación, diseñada para proporcionar un acceso eficiente y seguro a los datos. La API actúa como una capa intermedia entre el frontend (la aplicación del cliente) y la base de datos de la aplicación. Permitiendo la gestión de información en tiempo real entre ambas partes.

La arquitectura de la API sigue un diseño RESTful, priorizando la simplicidad, modularidad y escalabilidad. La API tiene unos tiempos de carga por solicitud muy bajos, optimizando la experiencia del usuario en la exploración del contenido, interacción social y personalización dentro de la aplicación Osiris.

La API de Osiris se basa en los siguientes principios:

- RESTful: Utiliza los métodos HTTP estándar (GET, POST, PUT, DELETE) para operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
- Seguridad: Implementa un sistema de autenticación utilizando funciones hash sobre las contraseñas y utilizando un sistema de tokens para proteger datos sensibles y garantizar el acceso autorizado.
- Modularidad: Divide la funcionalidad en blueprints o endpoints específicos para facilitar el mantenimiento y la evolución del sistema.

Servidor

El servidor usa la librería de Flask para montar una API, usa la librería de SQLAlchemy para la base de datos, tiene un sistema de logs y una vista web administrativa, en esta sección revisaremos la estructura y diseño del servidor, el cual tiene un diseño modular que separa cada módulo en paquetes según su función a excepción de las clases y archivos principales del proyecto.

Paquete Raíz

- **main.py**: Es el archivo principal, registra los blueprints, configura los logs, y arranca la aplicación.
- **extensions.py**: Es un archivo que sirve para configurar los sockets.
- **requirements.txt**: Es una lista de las dependencias necesarias para usar la aplicación, para poder instalarlas, se introduce en el terminal de python el comando: `pip install -r ./src/requirements.txt`
- **server.log**: Es un archivo donde se generan los logs del sistema.
- **config.json**: Es un archivo donde el usuario podrá cambiar los datos necesarios para establecer la conexión con la base de datos.

Paquete API_Rest

Hemos diseñado el servidor haciendo una estructura que sea modular utilizando “blueprints”. Los blueprints son una forma que tiene Flask para organizar los endpoints en módulos lógicos, cada cual

Osiris - Diseño

con un prefijo distinto, promoviendo así la separación de responsabilidades y la escalabilidad. Los principales módulos son:

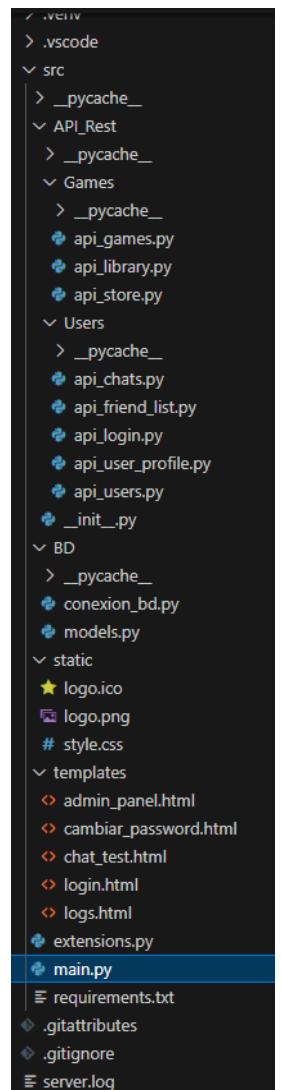
- Games

- api_games.py: Proporciona datos de los videojuegos.
- api_store.py: Administra las compras de los juegos así como las secciones destacadas de la tienda.
- api_library.py: Permite gestionar la biblioteca de juegos de los usuarios.

- Users

- api_users.py: Gestiona datos generales de los usuarios.
- api_friend_list.py: Gestiona las listas de amigos y solicitudes de amistad de los usuarios.
- api_login.py: Gestiona la autenticación y registro de los usuarios.
- api_user_profile.py: Gestiona datos del perfil de los usuarios, como los cambios de estado, descripción, nombre, foto etc.
- api_chat.py: Gestiona mensajes y grupos de chat de los usuarios.

Cada módulo tiene sus propias rutas y métodos REST, lo que simplifica la gestión del código y permite agregar nuevas características con facilidad.



Paquete BD

Hemos creado otro paquete para la conexión a la base de datos y otro para el modelo de datos el cual necesita SQLAlchemy para funcionar.

El paquete de conexión de la base de datos intentará acceder al archivo config.json para acceder a la configuración de la base de datos.

Paquete static

En este paquete van las imágenes y los estilos CSS necesarios para las páginas de la vista web.

Paquete templates

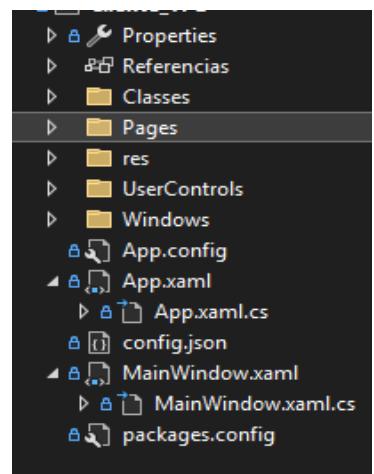
En este paquete están ubicados los archivos HTML que se utilizan para mostrar la vista web en el servidor.

Osiris - Diseño

Cliente

El cliente está desarrollado en C# y XAML y su funcionamiento principal consiste en solicitar los datos de la API del servidor montado con Flask y deserializar los datos JSON extraídos para mostrar la información al cliente. En esta sección se analizan y detallan la estructura y el diseño, la cual ha seguido un principio de separación de responsabilidades para su correcto desarrollo e implementación.

En la imagen se detalla la jerarquía de manera simple, únicamente indicando los paquetes desarrollados y a lo largo de este apartado de diseño profundizaremos en el contenido de estos mismos.



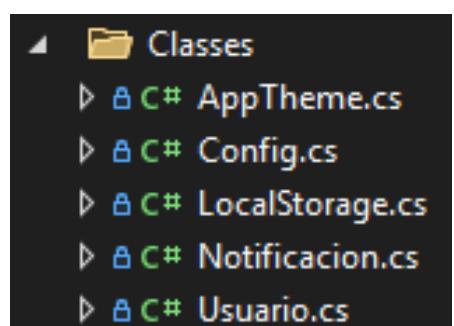
Paquete Raíz

- **App.xaml.cs**
 - La clase App es el cerebro de inicio de la aplicación, gestionando la autenticación automática del usuario mediante un token guardado o redirigiendo al inicio de sesión, con un modo offline de respaldo si no hay conexión.
- **config.json**
 - Se utiliza fuera de los paquetes de código para facilitar la configuración de la aplicación sin modificar el código fuente. Permite cambiar parámetros como la **IP** o el **puerto** del servidor de forma externa, lo que es crucial para la flexibilidad, el despliegue y la gestión de diferentes entornos (desarrollo, pruebas, producción) sin necesidad de recompilar la aplicación.
- **MainWindow.xaml**
 - La clase MainWindow define la estructura principal de la aplicación, incluyendo una cabecera (Cabeza UserControl) y un área dinámica (Frame) donde se cargan las diferentes páginas.
- **MainWindow.xaml.cs**
 - La clase MainWindow es la ventana principal de la aplicación que gestiona la carga y el estado del usuario (online/offline), carga una temática visual, y controla la navegación entre las diferentes páginas de la interfaz (**biblioteca, tienda, amigos, perfil, recarga de saldo**) a través de un Frame y eventos de su cabecera.

Paquete Classes

Este paquete contiene los **datos de usuario** (entidades o modelos), y maneja los **servicios y la lógica de negocio para el cliente**. A continuación, se especifica la utilidad de cada clase:

- **Usuario.cs**
 - Define los datos del usuario (nombre, email, etc.).
- **LocalStorage.cs**
 - Gestiona la carga y el guardado de datos de usuario en el almacenamiento local.
- **Config.cs/AppTheme.cs**
 - Almacenan las preferencias del usuario, como el modo claro/oscuro o la configuración de conectividad con el servidor.



Osiris - Diseño

- **Notificacion.cs**

- Se encarga de gestionar las notificaciones que se muestran al usuario.

Paquete Pages

El paquete Pages, agrupa todas las páginas visuales de la aplicación cliente del proyecto TFG. Cada clase dentro de este paquete hereda de *Page* (WPF) y representa una sección específica de la interfaz de usuario, como la tienda, los detalles de un juego, la biblioteca del usuario, o el perfil, entre otras. A continuación, se especifica la utilidad de cada clase:

- **paginaTienda**

- **Diseño visual (.xaml):** Muestra los juegos disponibles y destacados de la tienda, implementando estrategias visuales como hover o animaciones. Se define la estructura visual con:

- **Barra de búsqueda:**

- Estilo minimalista con animaciones.
 - Muestra sugerencias en un menú dropdown al escribir.

- **Carrusel de juegos destacados:**

- Muestra un juego principal con una imagen grande.
 - Pequeñas capturas del juego interactivas y navegación con botones.

- **Ofertas especiales:**

- Grid con juegos en descuento (imagen + precio rebajado).

- **Juegos económicos:**

- Muestra títulos con precios inferiores a 10€.

- **Nuevos lanzamientos:**

- Lista con imágenes, nombres, géneros y fechas de lanzamiento.

- **Funcionalidad (.cs):** Simula una tienda de videojuegos, conectándose al servidor para obtener datos de los juegos, implementando las siguientes funcionalidades:

- **Interfaz principal** con:

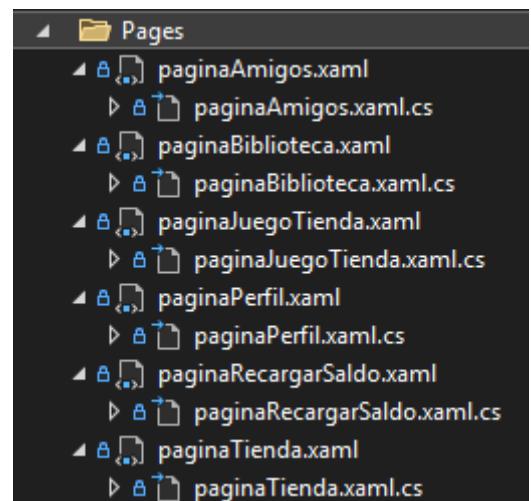
- Carrusel de juegos destacados con imágenes, precios y miniaturas.
 - Sección de ofertas especiales con descuentos.
 - Juegos por precio específico.
 - Nuevos lanzamientos con detalles como género y fecha.

- **Funcionalidades clave:**

- Animaciones fluidas.
 - Búsqueda en tiempo real con autocompletado.
 - Navegación a las páginas en las que se detallan los juegos de la tienda.

- **Tecnologías utilizadas:**

- **HTTP requests** (`HttpClient`) para obtener datos JSON de `api_store` mencionada anteriormente.
 - Binding y plantillas WPF para la UI.
 - Programación asíncrona (`async/await`).



Osiris - Diseño

- **páginaRecargarSaldo**

- **Diseño visual (.xaml)**: Define la interfaz de **recarga de saldo**. A continuación se detalla la estructura:
 - **Recarga Rápida:**
 - Botones con cantidades predefinidas (5€, 10€, 25€, etc.).
 - Diseño con estilo personalizado (**BotonRecargaRapidaStyle**).
 - **Cantidad Personalizada:**
 - Campo de texto para ingresar una cantidad de dinero manual.
 - Validación dinámica y mensajes informativos.
 - **Confirmación de Recarga:** Panel flotante que muestra:
 - Cantidad seleccionada.
 - Selección de método de pago.
 - Botones para confirmar o cancelar.
 - **Elementos XAML vinculados a la funcionalidad:**
 - Validar cantidades de dinero a recargar.
 - Procesar pagos.
 - Mostrar/ocultar el panel de confirmación.
- **Funcionalidad (.cs)**: Lógica para implementar la recarga de saldo que complementa el XAML. Proporciona la funcionalidad para:
 - **Recarga Rápida**
 - Maneja clics en botones predefinidos (5€, 10€, etc.) mediante (**RecargaRapida_Click**).
 - Muestra un panel de confirmación con la cantidad seleccionada (**MostrarConfirmacionRecarga**).
 - **Recarga Personalizada**
 - Valida el input del usuario en tiempo real (**txtMontoPersonalizado_TextChanged**):
 - Sólo permite números y un punto decimal.
 - Verifica rangos (1€ - 10,000€).
 - Muestra feedback visual (mensajes de error/exito).
 - **Procesamiento de Pago**
 - Envía la recarga al servidor mediante *HTTP PUT* (**RecargarSaldoAsync**) enviándola con **api_users**.
 - Parsea la respuesta JSON para actualizar el saldo del usuario.
 - Muestra una ventana de confirmación (**VentanaRecarga**).
 - **Gestión de Métodos de Pago**
 - Carga opciones predefinidas (tarjeta, PayPal, etc.) en un ComboBox. (**CargarMetodosPago**).
 - **Animaciones:**
 - Efectos hover en botones (**AplicarEfectoHover**).
 - Transiciones suaves al mostrar/ocultar paneles (**AplicarFadeIn**, **AplicarFadeOut**).
 - Animación de entrada al cargar la página (**AplicarAnimacionEntrada**).

Osiris - Diseño

- **páginaPerfil**
 - **Diseño visual (.xaml):** Define la interfaz del perfil del usuario, agregando un toque moderno y animado. A continuación, se detallan los puntos clave:
 - **Cabecera del Perfil:**
 - **Foto de perfil:** Con botón para cambiar la imagen.
 - **Nombre de usuario:** Editable.
 - **Nombre de cuenta:** No editable.
 - **Biografía:** Área de texto con fondo oscuro.
 - **Amigos:** Muestra contador de amigos.
 - **Últimos Juegos Comprados:**
 - Muestra los 3 últimos juegos comprados como tarjetas interactivas con:
 - Carátula animada (efecto hover con zoom y borde rojo).
 - Nombre y fecha de compra.
 - **Funcionalidad (.cs):** Es la parte de Backend de la página de perfil de usuario, que complementa el XAML anterior. Proporciona toda la funcionalidad dinámica y la lógica de negocio. A continuación, se proporciona un desglose detallado:
 - **Carga de Datos del Usuario:**
 - Obtiene los datos de **api_users** y muestra: nombre, foto de perfil, biografía y número de amigos.
 - Usa **cargarDatosUser()** para mostrar la información del objeto Usuario almacenado en MainWindow.
 - **Gestión de Amigos:**
 - **ObtenerNúmeroAmigos():** Consulta al servidor y actualiza el contador de amigos.
 - **Subida de Foto de Perfil:**
 - **btnSubirImagen_Click_1:** Permite seleccionar una imagen local y subirla a Imgur mediante su API.
 - **ActualizarFotoUsuarioConUrl():** Guarda la URL de la nueva foto en la base de datos.
 - **Edición de Perfil**
 - **btnEditar_Click:** Habilita campos editables (nombre y biografía).
 - **btnGuardar_Click:** Envía los cambios al servidor mediante una petición HTTP PATCH.
 - **Últimos Juegos Comprados**
 - **CargarÚltimosJuegos():** Muestra hasta 3 juegos recientes con sus carátulas y fechas de compra.
 - Efecto hover en carátulas: Cambia el fondo de la página al pasar el ratón.
 - **Sincronización con el Servidor**
 - **RecargarUsuarioDesdeServidor():** Actualiza los datos locales del usuario desde la base de datos.

● **páginaJuegoTienda**

- **Diseño visual (.xaml):** Está diseñada para mostrar los detalles de un juego dentro de una tienda digital, combinando aspectos visuales, interactividad y usabilidad:

Osiris - Diseño

- **Layout Principal:** Utiliza un Grid como contenedor base con dos columnas principales:
 - **Izquierda:** Carrusel de imágenes, miniaturas, descripción extendida y categorías.
 - **Derecha:** Imagen principal, precio, botón de compra y detalle clave de juego.
- **Estilos personalizados:**
 - **PurchaseButtonStyle** controla estados visuales del botón de compra mediante **triggers** y **multitriggers**.
 - **ScrollBars** vertical y horizontal estilizadas para una apariencia minimalista y coherente.
 - **ScrollView** personalizado para integrar barras de scroll estilizadas.
- **Componentes Visuales:**
 - Carrusel con miniaturas para navegación entre imágenes del juego.
 - Paneles con bordes y espaciados para organizar texto descriptivo y categorías.
 - Panel derecho con información crítica (precio, lanzamiento, desarrolladores) y botón de compra con animación de aparición controlada por triggers.
- **Interactividad y Usabilidad:**
 - Los estados del botón reflejan disponibilidad y condición de compra mediante propiedad **Tag**.
 - Uso de **Cursor="Hand"** para elementos clicables.
 - Diseño pensado para 1280x720, con tipografía y colores que garantizan legibilidad y contraste.
- **Funcionalidad (.cs):** Maneja la lógica de los detalles de la página de un videojuego en específico. Trabaja junto a XAML para mostrar información dinámica. Aquí está su estructura clave:
 - **Carga de Datos del Juego:**
 - Obtiene los detalles del juego desde la **api_games**, y muestra las carátulas, descripciones, precio, categorías, desarrolladores, etc.
 - Usa modelos como *CarruselResponse* y *Juego* para deserializar JSON.
 - **Carrusel de Imágenes:**
 - Muestra capturas de pantalla en un **carrusel automático** con:
 - Miniaturas interactivas.
 - Temporizador para cambio automático (7 segundos).
 - Efectos hover y animaciones de fade-in.
 - **Sección de Compra:**
 - Botón dinámico ("COMPRAR"/"EN BIBLIOTECA"/"Free To Play").
 - Lógica de compra con petición HTTP POST hacia la **api_store** (**ComprarJuegoAsync()**).
 - Actualización del saldo del usuario tras la compra.
 - **Visualización de Contenido**
 - **Descripción HTML:** Parsea y muestra texto con formatos (negritas, listas, encabezados).
 - **Metadata:** Fecha de lanzamiento, desarrolladores, géneros.
 - **Precio:** Calcula descuentos y muestra precio final.
 - **Integración con el Sistema**

Osiris - Diseño

- Sincroniza con la biblioteca del usuario.
- Actualiza la UI al comprar (notificaciones, cambio de botón).

• paginaBiblioteca

- **Diseño visual (.xaml):** Define una interfaz de usuario para una página de biblioteca de juegos. A continuación, detallaremos los componentes principales:
 - **Sección Superior (Datos del Juego):**
 - **Imagen de fondo:** Con efecto de degradado para difuminarse hacia abajo.
 - Logo del juego: *ImagenLogo* (con alternativa *txtFalloLogo* si falla la imagen).
 - **Barra de búsqueda:** Estilo personalizado (**DarkMinimalistSearchBox**) con icono de lupa.
 - **Botón de acción:** *BotonJugar* que puede cambiar entre estados (*Play*, *Install*, *Unavailable*).
 - **Sección Inferior (Lista de juegos):**
 - **ScrollViewer:** Para navegar por la lista de juegos.
 - **WrapPanel:** *panelJuegosBiblioteca* donde se mostrarán los juegos en formato de cuadrícula.
 - **Botón "Agregar Juego":** Estilo premium en la esquina inferior izquierda.
- **Funcionalidad (.cs):** Maneja la lógica de la Biblioteca, la cual permite a los usuarios gestionar y explorar su colección de juegos. Su estructura y lógica se centra en **la carga eficiente de datos, la interacción dinámica con el usuario y la resistencia frente a la ausencia de recursos visuales o de conexión.**
 - **Carga y Gestión de Datos:**
 - **Limpieza de datos:** Se garantiza que la interfaz esté limpia al borrar los datos de sesiones anteriores, evitando duplicados y asegurando una presentación fresca.
 - **Carga Local de Biblioteca:** Primero, se intenta cargar la lista de juegos (*appids* y *nombres*) desde el almacenamiento local. Esto permite que la aplicación funcione incluso si no hay conexión a Internet, mostrando al menos los juegos que el usuario ya tenía registrados.
 - **Obtención de Biblioteca desde la API:** Si hay conexión a Internet, la aplicación se conecta a un endpoint de la API(*api_library*) configurado en **Config.IP** y **Config.Puerto** para obtener la biblioteca de juegos del usuario. Esta llamada asíncrona asegura que la interfaz de usuario no se bloquee mientras se esperan los datos del servidor. Se deserializa la respuesta JSON para obtener una lista de objetos *Juego* que contienen *app_id*, *captura*, *nombre* y *header*.
 - **Almacenamiento Local de Juegos:** Los juegos se gestionan localmente mediante la clase *JuegoInfo*, que almacena el *AppId*, el *Nombre* y la *RutaEjecutable*. Esto permite persistir las rutas de instalación de los juegos incluso si el usuario cierra la aplicación o cambia de sesión. Los métodos **GuardarJuegosEnJson()** y **CargarJuegosDesdeJson()** se encargan de serializar y deserializar esta información en un archivo JSON ubicado en el directorio de datos de la aplicación.

Osiris - Diseño

- **Fondo Dinámico y Logos de Juegos:** Se descargan y cargan de forma asíncrona imágenes de fondo, logos y pósteres verticales para cada juego.
- **Manejo de Fallos en Imágenes:** En caso de que una imagen principal no se pueda cargar, el sistema intenta cargar una imagen de respaldo local. Si no existe, se recurre a otras URLs proporcionadas por la API. Además, las funciones `GenerarImagenFallback` son capaces de crear dinámicamente pósteres verticales con un efecto de desenfoque de fondo si la imagen principal no está disponible, utilizando otras imágenes del juego para mantener la coherencia visual.
- **Visualización de Juegos:** Los juegos de la biblioteca se presentan como miniaturas de imágenes interactivas. Cada miniatura tiene las siguientes funcionalidades:
 - **Animaciones al pasar el ratón:** Las animaciones de escala al pasar el ratón por encima proporcionan una **respuesta visual inmediata**, indicando que el elemento es interactivo.
 - **Menú Contextual:** Al hacer clic con el botón derecho, se despliega un menú contextual que ofrece opciones esenciales: "Jugar", "Cambiar ejecutable", "Ver detalles" y "Desinstalar". Este diseño es **familiar para los usuarios** y agrupa las acciones relevantes de forma lógica.
 - **Interactividad con el Botón Principal:** Al hacer clic izquierdo en una miniatura, el juego se convierte en el "actual", actualizando el fondo principal y el botón de acción principal para reflejar su estado (*Jugar/Instalar*).

- **paginaAmigos**

- **Diseño visual (.xaml):** El diseño está estructurado para ofrecer una experiencia al usuario **centralizada e intuitiva** para la gestión de contactos y la comunicación en tiempo real:
 - **Panel Izquierdo:** Dedicado a la **gestión de amigos**, incluyendo solicitudes pendientes y la lista de amigos.
 - **Solicitudes de Amistad:**
 - Un **TextBlock** muestra un título dinámico que incluye el número actual de solicitudes pendientes, manteniendo al usuario informado de un vistazo.
 - Las solicitudes se listan dentro de un **ScrollView** que contiene un **StackPanel**, permitiendo la adición dinámica de elementos y el desplazamiento si la lista es extensa.
 - Un **Border** actúa como separador visual, dividiendo esta sección de la lista de amigos.
- **Lista de Amigos:**
 - Un **TextBlock** claro (Lista de amigos) encabeza esta sección, indicando su propósito.
 - Un **ScrollView** envuelve un **StackPanel** (listaAmigos), que será poblado dinámicamente con los perfiles de los amigos del usuario. Este diseño facilita la exploración de listas largas de amigos.

Osiris - Diseño

- **Funcionalidad de Adición de Amigos:**
 - **Tu Código de Amigo:** Se muestra el ID único del usuario (txtIdAmigoUsuarioActual) junto a un botón (btnCopiarIdAmigo) que permite copiar el ID al portapapeles con un solo clic. El botón utiliza un estilo personalizado con animaciones sutiles (ScaleTransform) al interactuar, mejorando la respuesta visual.
 - **Buscar Amigos:** Un TextBox con un diseño minimalista y un ícono de búsqueda integrado permite al usuario introducir un *ID* de amigo para enviar una solicitud. Un botón con un ícono de "enviar" complementa la funcionalidad de búsqueda, también con el estilo animado. Los **TextBlock** de placeholder dentro del **TextBox** ofrecen una guía clara al usuario.
- **Panel Derecho:** Asignado al **área de chat**, donde los usuarios pueden interactuar con sus amigos.
 - **Encabezado del Chat:**
 - Un TextBlock con gran tamaño y negrita se destina a mostrar el nombre del amigo con el que se está chateando, proporcionando un contexto claro. Inicialmente está oculto (Visibility="Collapsed").
 - Un TextBlock adicional sirve como mensaje de bienvenida o instrucción inicial ("Haz click en un amigo para ver su chat"), siendo visible hasta que un amigo es seleccionado.
 - **Área de Mensajes:**
 - Un ScrollViewer contiene un StackPanel donde se muestran los mensajes de la conversación. El ScrollViewer asegura que, independientemente de la longitud del historial de chat, el usuario pueda desplazarse para ver todos los mensajes. Los mensajes se agregarán dinámicamente a este panel.
 - **Entrada de mensajes:**
 - Un Grid, inicialmente oculto, alberga el TextBox para escribir mensajes y un botón de "enviar".
 - El TextBox utiliza un estilo similar al de búsqueda, proporcionando una interfaz uniforme y reconocible. Incluye un TextBlock como placeholder ("Escribe un mensaje...") que se oculta automáticamente cuando el usuario empieza a escribir.
 - El botón de envío usa el mismo BotonCopiarStyle para mantener la consistencia visual y ofrecer una retroalimentación interactiva.
- **Funcionalidad (.cs):** El código implementa la funcionalidad para gestionar los siguientes elementos:
 - **Inicialización**
 - Configura elementos UI, tema y temporizador para actualizaciones automáticas.
 - Llama al servidor para cargar los datos, haciendo una solicitud GET hacia **api_users** para cargar todos los amigos de ese usuario en concreto.
 - **Conexión con SocketIO**
 - Establece conexión con el servidor Socket.IO utilizando **api_chats**.
 - El cliente **Socket.IO** se configura para escuchar eventos clave como *nuevo_mensaje*, *status* y *error*, permitiendo la recepción y procesamiento de mensajes en tiempo real.

Osiris - Diseño

■ Gestión de Solicitudes de Amistad

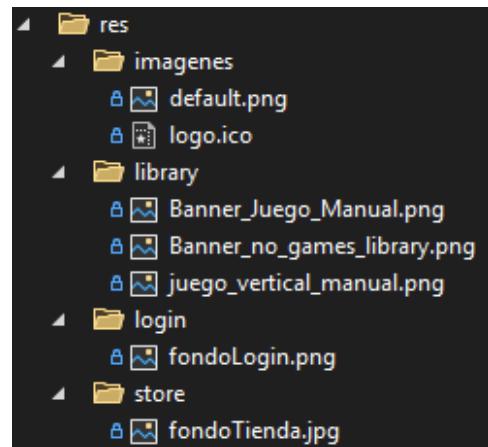
- El código consulta la `api_users` mediante un método **GET** para verificar las solicitudes de amistad pendientes. Una vez obtenidas, actualiza dinámicamente la interfaz de usuario para reflejar el estado actual de las solicitudes. También gestiona las acciones de aceptar y rechazar solicitudes, interactuando con el servidor mediante solicitudes **POST**.

■ Chat en Tiempo Real

- Permite el envío de mensajes a través de la conexión **Socket.IO**, utilizando el evento `enviar_mensaje` para comunicar el contenido del chat al servidor. Los mensajes recibidos del servidor se procesan y se añaden al historial de chat en la UI.

Paquete res

Este apartado detalla la organización de los recursos gráficos (imágenes e iconos) utilizados en la aplicación. Estos elementos visuales se encuentran centralizados en el directorio `/res`, facilitando su gestión y acceso. A continuación se detalla la utilización de cada imagen en la aplicación:



• imágenes

- **default.png**: Imagen utilizada para agregar de manera por defecto a un usuario cuando no tuviese foto de perfil.
- **logo.ico**: Logo de la aplicación.

• library

- **Banner_Juego_Manual.png**: Banner utilizado para cuando el usuario agrega de manera manual un juego de su dispositivo.
- **Banner_no_games_library.png**: Banner utilizado para cuando el usuario no tiene ningún juego agregado en la biblioteca.
- **juego_vertical_manual.png**: Banner vertical utilizado para cuando el usuario agrega un juego en su biblioteca.

• login

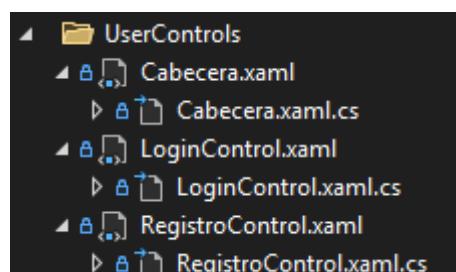
- **fondoLogin.png**: Banner utilizado en el inicio de sesión

• store

- **fondoTienda.jpg**: Banner utilizado en la tienda.

Paquete UserControls

Es un componente nativo de la UI de WPF que nos proporciona recursos para construir interfaces de usuario modulares, reutilizables y bien organizadas. A continuación, se explica el contenido de los recursos:



Osiris - Diseño

- **Cabecera**

- Representa la barra superior de la aplicación. Su función es proporcionar navegación global y mostrar la información del perfil logueado.

- **LoginControl**

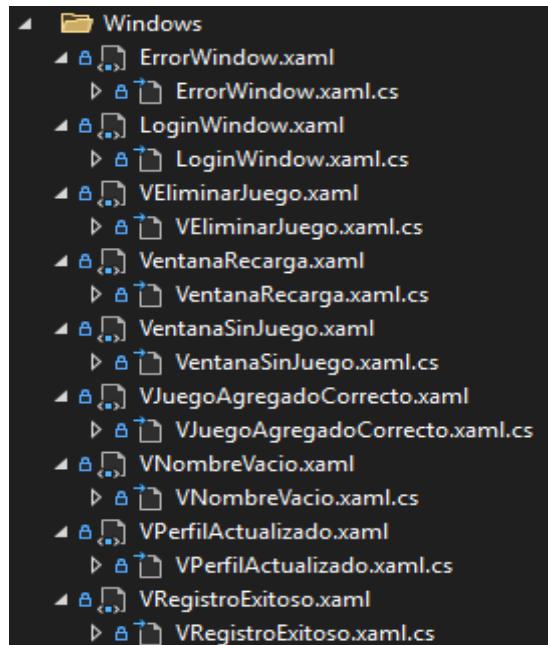
- Gestiona la autenticación de usuario. Su objetivo es permitir a los usuarios iniciar sesión en la aplicación proporcionando su correo electrónico y contraseña, además de ofrecer un enlace para registrarse si no tienen cuenta. La lógica de autenticación y la interacción con el backend se hace utilizando la **api_login** del servidor.

- **RegistroControl**

- Es un componente de interfaz de usuario que permite a los usuarios registrarse en la aplicación. Recopila la información necesaria como nombre de cuenta, correo electrónico y contraseña, realiza validaciones básicas en el cliente y se comunica con un servicio de backend para completar el proceso de registro. También proporciona una opción para volver a la pantalla de inicio de sesión. El backend es responsable de la lógica, las validaciones de entrada y la comunicación con el servicio de registro.

Paquete Windows

El paquete Windows contiene todas las definiciones de ventanas (**.xaml**) y su lógica asociada (**.xaml.cs**) utilizadas en la aplicación. Estas ventanas están diseñadas para guiar al usuario a través de los diversos flujos de interacción, como el inicio de sesión (**LoginWindow.xaml**) o el registro. El diseño de estas ventanas es altamente personalizado, a menudo implementando estilos sin bordes o barras de título predeterminadas para ofrecer una apariencia más integrada y moderna. Además de la interacción principal, estas ventanas también cumplen la función vital de informar al usuario sobre el resultado de sus acciones, mostrando mensajes de éxito (por ejemplo, **VRegistroExitoso.xaml** o **VJuegoAgregadoCorrecto.xaml**) o de error (como **ErrorWindow.xaml** o **VNombreVacio.xaml**).



- **ErrorWindow**

- La clase ErrorWindow es una ventana emergente personalizada que muestra mensajes de error específicos, aplicando una temática dinámica y ofreciendo un botón interactivo para cerrar la notificación.

- **LoginWindow**

- La clase LoginWindow gestiona la navegación entre las vistas de inicio de sesión y registro mediante transiciones animadas de desvanecimiento, permitiendo al usuario cerrar y mover la ventana personalizada.

Osiris - Diseño

- **VEliminarJuego**
 - La clase VEliminarJuego es una ventana de diálogo personalizada que solicita confirmación al usuario para eliminar un juego, estableciendo una propiedad booleana según la respuesta y cerrándose automáticamente.
- **VentanaRecarga**
 - La clase VentanaRecarga es una ventana de confirmación personalizada que informa al usuario sobre una recarga de saldo exitosa, mostrando detalles como el monto recargado, el método de pago y el saldo actual, y se cierra al presionar un botón de aceptar.
- **VentanaSinJuego**
 - La clase VentanaSinJuego es una ventana de notificación personalizada que informa al usuario que tiene menos de tres juegos comprados y le ofrece la opción de cerrar la ventana o navegar directamente a la página de la tienda.
- **VJuegoAgregadoCorrecto**
 - La clase VJuegoAgregadoCorrecto es una ventana de notificación personalizada que informa al usuario sobre el éxito al agregar un juego y se cierra al presionar un botón de aceptar.
- **VNombreVacío**
 - La clase VNombreVacío es una ventana de notificación personalizada que informa al usuario que el campo de nombre no puede estar vacío y se cierra al presionar el botón "Aceptar".
- **VPerfilActualizado**
 - La clase VPerfilActualizado es una ventana de notificación personalizada que informa al usuario que su perfil ha sido actualizado correctamente y se cierra al presionar el botón "Aceptar".
- **VRegistroExitoso**
 - La clase VRegistroExitoso es una ventana de notificación personalizada que informa al usuario que su registro fue exitoso y que ahora puede iniciar sesión, cerrándose al presionar el botón "Aceptar".

Tecnologías

En este proyecto se han utilizado una gran variedad de tecnologías, tantas que hemos decidido organizarlas según si se han utilizado dentro de las aplicaciones (como librerías, frameworks, lenguajes de programación, entornos de desarrollo, etc.) o si se han usado fuera (aplicaciones para la creación de diagramas, control de versiones etc.)

Osiris - Diseño

Servidor

Python



Para hacer el servidor, hemos necesitado escoger un lenguaje de programación, decidimos usar Python ya que nos interesaba aprenderlo y a su vez es un lenguaje de los más utilizados en el mundo, y de cara al mundo laboral, nos podría beneficiar mucho aprenderlo.

Visual Studio Code



Visual Studio Code es el entorno de desarrollo que hemos elegido para poder realizar la aplicación del servidor, la hemos elegido por ser el entorno de desarrollo más popular del mercado y por ser el más recomendado para Python.

Flask



Para realizar el servidor hemos tenido que investigar cómo funciona un API y cómo crear una API Rest. Tras ver las opciones que teníamos hemos decidido usar Flask.

Flask es un Framework escrito en Python muy utilizado para facilitar el desarrollo de aplicaciones Web bajo el patrón MVC (Modelo Vista Controlador). Este Framework nos abre un puerto donde poder alojar un HTML de forma muy cómoda, para tener una vista web de la aplicación, y poder acceder a los métodos de la API.

Hemos investigado que tiene una forma de modularizar los métodos de la API usando “blueprints” que son objetos que permiten definir rutas, templates y recursos estáticos de forma modular. Lo cual es algo que nos ha convencido bastante y hemos decidido implementarlo.

SQLAlchemy



Es una librería que te permite trabajar con bases de datos SQL sin tener que utilizar sentencias SQL, trabajando con los datos como si fuesen objetos, ha sido una herramienta muy útil a la hora de realizar las peticiones a la base de datos.

Osiris - Diseño

HTML, CSS y JavaScript



Ya que Flask nos ha brindado la oportunidad, hemos aprovechado el servidor web para algo más que solo hacer endpoints de la api, hemos hecho una vista web para que el administrador pueda ver los logs del servidor, así como una gran variedad de datos en tiempo real, por ejemplo: la cantidad de juegos y usuarios que tiene la aplicación.

Logging - Logging facility for Python



Hemos investigado cómo usar un sistema de logs, para en lugar de depurar en el terminal usando prints, utilizar nuestro sistema de logs. Para ello hemos encontrado una librería llamada '*logging*' que permite depurar forma modular.

Cada *log* tiene una categoría, y funciona con niveles, pudiendo limitar cuales son los logs que salen y cuales no. Estos logs aparecen en la consola y también en un fichero, este fichero, se sube periódicamente a un endpoint, el cual se lee cada N segundos y se agrega a una lista en la página web que monitoriza el servidor.

Cliente

C# con WPF (Windows Presentation Foundation)



Para la parte del cliente, hemos optado por desarrollar una aplicación de escritorio utilizando C# con WPF. Una tecnología potente y versátil para crear interfaces gráficas modernas en Windows. Gracias a XAML hemos podido integrar elementos visuales con gran personalización y mantener un rendimiento fluido en equipos actuales.

Osiris - Diseño

Visual Studio



Es el entorno de desarrollo que hemos utilizado para desarrollar la aplicación del cliente. El cual nos ha permitido implementar las siguientes librerías:

HttpClient

Se ha utilizado la herramienta HttpClient para consumir servicios REST y recuperar datos en formato JSON.

NewtonsoftJson:

Biblioteca de serialización utilizada para deserializar los objetos JSON del cliente.

Aplicaciones y servicios externos

Github



Para hacer la aplicación entera, hemos usado GitHub como sistema de control de versiones, hemos hecho un repositorio para el cliente y otro para el servidor, esto nos ha permitido trabajar cómodamente en equipo y también puede servirnos en un futuro si decidimos dejarlo público a la hora de buscar trabajo.

Imgur



Para subir las imágenes de los usuarios hemos utilizado el api de imgur, que permitía subir las imágenes a internet mientras nos guardamos esa url

PostgreSQL



Cuando ya teníamos la base de datos funcionando, tuvimos que cambiarla por una base de datos relacional, después de un trabajo de investigación, descubrimos que tanto Steam como Epic Games, utilizan PostgreSQL para sus bases de datos, ya que manejan un gran volumen de información.

Osiris - Diseño

Una vez decidido, tuvimos que pasar los datos que teníamos en MongoDB, también era la primera vez que lo usamos, así que tuvimos muchas dificultades con ello, todo esto está más detallado en el apartado de Experiencia durante el desarrollo.

Postman



En medio del desarrollo, tuvimos que investigar cómo realizar peticiones, POST , PUT, PATCH y DELETE. Lo cual nos llevó a descubrir Postman, una app que te permite enviar peticiones de cualquier tipo a tu API.

SocketIO



Para realizar el chat, teníamos claro que debía de ser en tiempo real. Por lo que decidimos utilizar Sockets. Por suerte, Flask y C# permiten trabajar con SocketIO. Así que nos pusimos manos a la obra y realizamos los métodos oportunos para comunicar los dos clientes y realizar el chat.

Adobe Photoshop



Hemos utilizado photoshop para la creación del logo de la aplicación, ya que era una herramienta que teníamos a mano y ya estábamos familiarizados con ella.

draw.io



Hemos utilizado [Draw.io](#) para la creacion de la mayoría de los diagramas, el cual es un programa que hemos usado mucho en clase y estamos familiarizados con él, y en su versión web, tiene una forma de trabajar varias personas a la vez en el mismo diagrama, lo cual ha sido muy cómodo para organizarnos en la fase de análisis.

Osiris - Diseño

StarUML



Hemos utilizado StarUML, una potente aplicación que permite hacer diagramas UML con gran facilidad, para realizar el diagrama de casos de uso y el diagrama de clases, ya que nos resultaba más fácil realizarlo con esta herramienta.

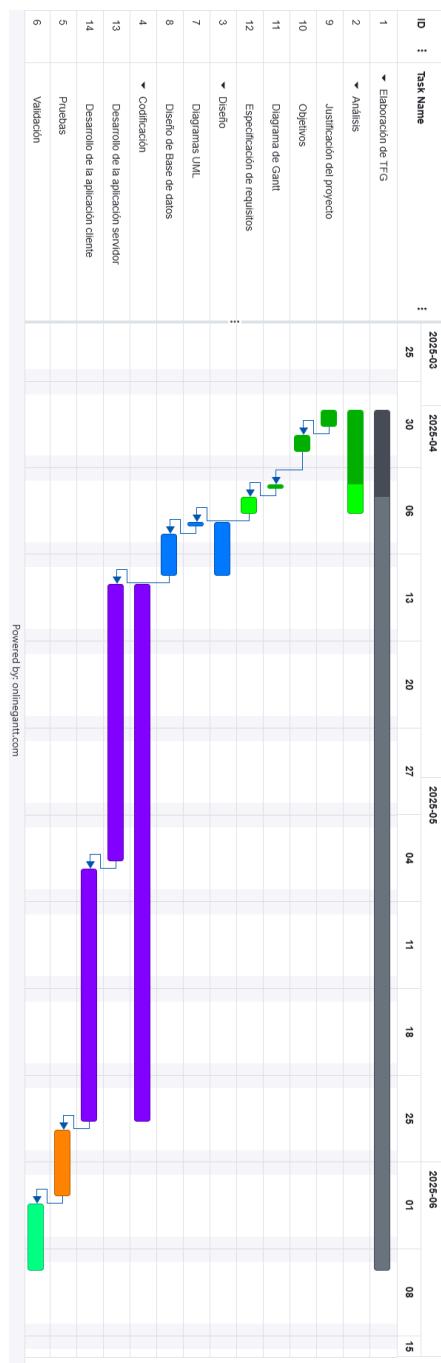
dbDiagram.io



[dbDiagram.io](#) es una aplicación web que permite realizar el diseño de bases de datos relacionales y exportar su estructura, nos ha facilitado el diseño de la propia base de datos, así como posteriormente, su traspaso a PostgreSQL.

Osiris - Diseño

Cronograma



Desarrollo

Servidor

Flask

Para poder iniciar el servidor es tan fácil como poner las siguientes líneas:

```
app = Flask(__name__)
app.run()
```

El programa ya te da un enlace, por defecto la IP de localhost y el puerto 50000 que es donde puedes poner cosas en la página web, de esa forma enviaremos la información necesaria para el cliente.

Para ello mostraremos uno de los métodos más sencillos de la API, buscar un juego según su id.

```
@blueprint_games.route('/<int:id_game>', methods=['GET'])
def get_game_by_id(id_game):
    try:
        short = request.args.get('short') is not None # esto devuelve true si no hay que poner parametro
        full = request.args.get('full') is not None
        with get_session() as session:
            games_query = session.query(Juegos).order_by(Juegos.steam_appid).filter(Juegos.steam_appid == id_game)

            if short:
                json_string = [game_to_dict_short(juego) for juego in games_query]
                logger.debug("La petición se ha convertido en diccionario correctamente")
            elif full:
                json_string = [game_to_dict_full(session, juego) for juego in games_query]
                logger.debug("La petición se ha convertido en diccionario de formato entero correctamente")
            else:
                json_string = [game_to_dict(session, juego) for juego in games_query]
                logger.debug("La petición se ha convertido en diccionario de formato corto correctamente")

            return jsonify({
                'juegos': json_string
            })
    except Exception as e:
        logger.error(f"Error inesperado en el endpoint /games/<int:id_game>: {e}")
        return jsonify({'error': 'Error inesperado en el servidor', 'detalle': str(e)}), 500
```

Como se puede observar en las primeras líneas, estos métodos pueden recibir parámetros, en la url, permitiendo que hagamos cosas como mostrar más o menos información o incluso aplicar filtros.

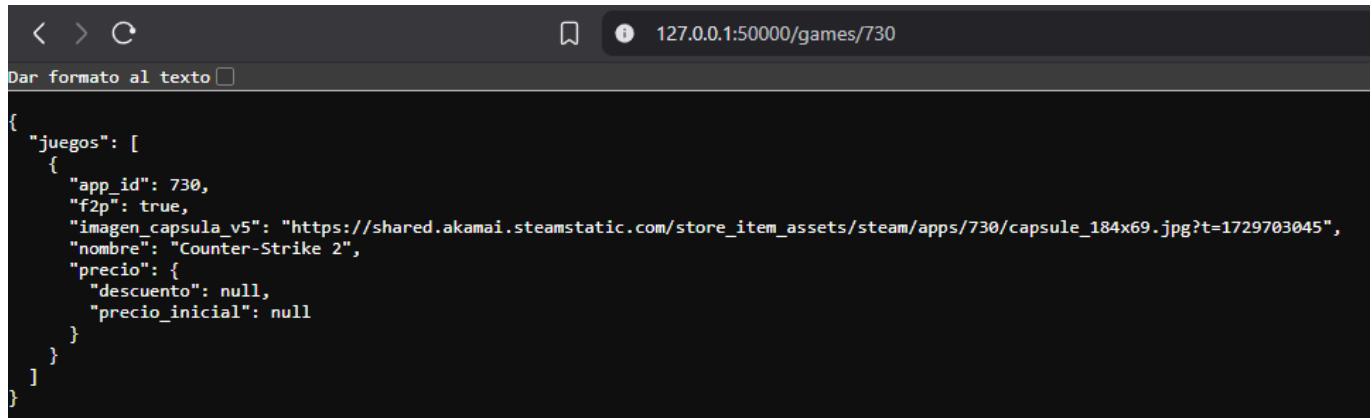
Este método usa ‘blueprint_games’, el cual hemos registrado en el main con el prefijo games/

```
# Registra los blueprints, los cuales son necesarios para modularizar la API
def register_flask_blueprints():
    app.register_blueprint(blueprint_friend_list, url_prefix='/friend_list')
    app.register_blueprint(blueprint_games, url_prefix='/games')
    app.register_blueprint(blueprint_library, url_prefix='/library')
    app.register_blueprint(blueprint_store, url_prefix='/store')
    app.register_blueprint(blueprint_users, url_prefix='/users')
    app.register_blueprint(blueprint_user_profile, url_prefix='/user_profile')
    app.register_blueprint(blueprint_login, url_prefix='/login')
    app.register_blueprint(blueprint_chats, url_prefix='/chats')
    logger.debug('Se han registrado los blueprints de flask')
```

Osiris - Desarrollo

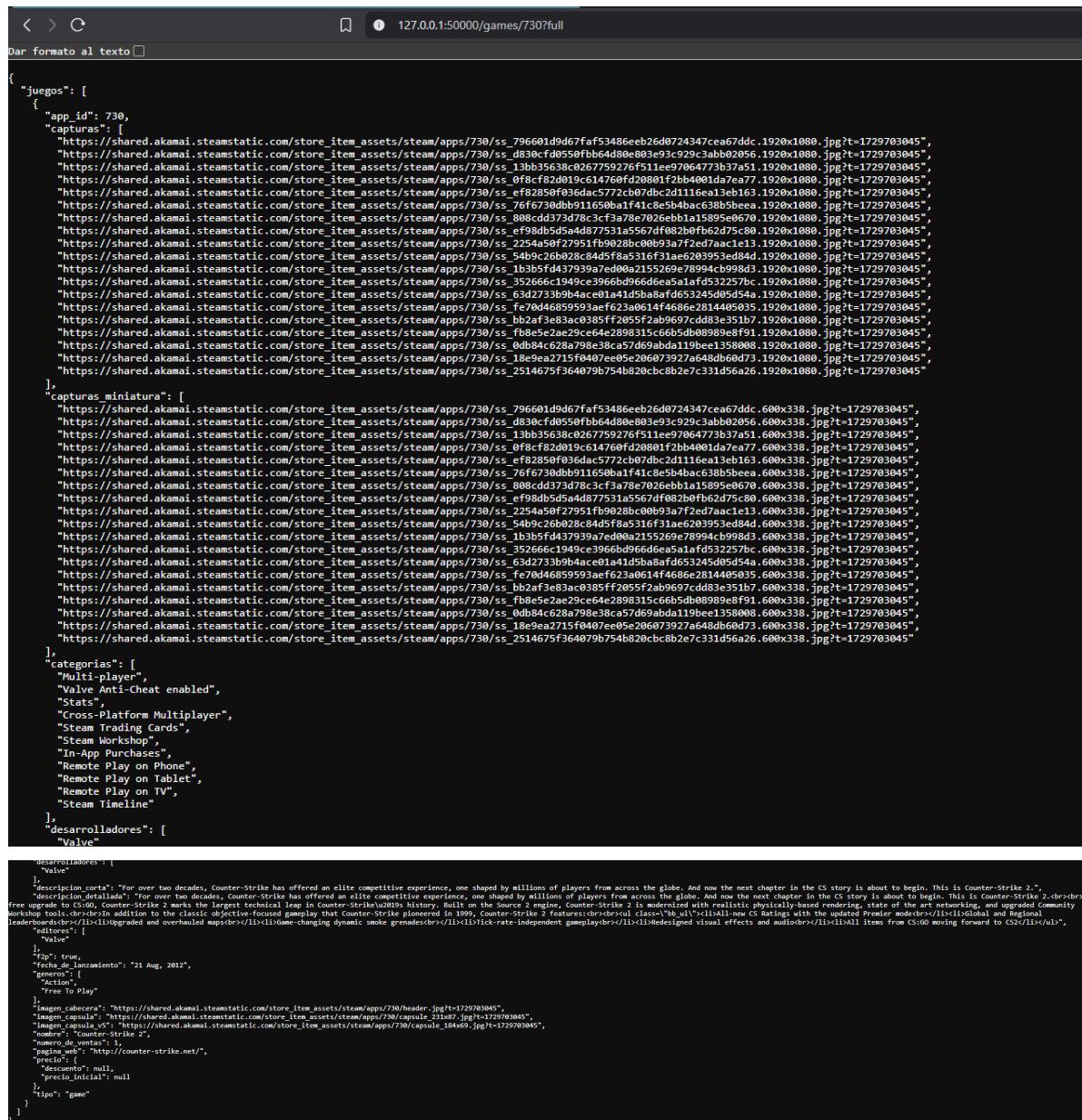
para poner la url para probar este método tendríamos que buscar la id de un juego, por ejemplo la id 730, y ponerla en el enlace de esta forma <http://127.0.0.1:50000/games/730>

Esto nos llevará a una página que nos da la información en formato json.



```
{  
  "juegos": [  
    {  
      "app_id": 730,  
      "f2p": true,  
      "imagen_capsula_v5": "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/capsule_184x69.jpg?t=1729703045",  
      "nombre": "Counter-Strike 2",  
      "precio": {  
        "descuento": null,  
        "precio_inicial": null  
      }  
    }  
  ]  
}
```

Para pasarle un argumento habría que poner el símbolo '?' antes del argumento, de esta forma



```
{  
  "juegos": [  
    {  
      "app_id": 730,  
      "capturas": [  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_796601d9d67faf5348eeb26d0724347cea67ddc.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_d830cfcd0550fbbe4d8e0803e93c929c3abb02056.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_13bb35638c0267759276511e64773b37a51.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_0f8cf82d019c614760fd20801fbba4001da7e77.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_ef82850f036dac5772cb02d1116ea13eb163.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_f6f6730dbb911650ba1fa1f5b4ba638b5beea.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_888cd373d78c3cf3a7e87026ebba15895e0670.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_e98bd5d5a4d7751a556df082b0fb62d75c80.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_2254a50f27951f9028b00b93a7f2ed7aac1e13.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_54b9c26b028c64d5f8a5316f1ae6208953ed84d.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_ef82850f036dac5772cb02d1116ea13eb163.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_3526661949cc3966bd966feaa5a1afad532257bc.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_63d2733b9b4ace01a41d5ba8afdf653245d05d54a.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_f7046859593aefc23a0614f4686e2814405035.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_bb2af3e83a0385ff2055f2ab9697cd83e351b7.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_fb8e5e2a29c64e2898315c6b5d8e0898e8f91.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_0db84c628a798e3ca5769abda119beel358008.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_18e9aea2715f0407ee05e206073927a648db60d73.1920x1888.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_2514675f364079b754b820cbc8b2e7c331d56a26.1920x1888.jpg?t=1729703045"  
      ],  
      "capturas_minitura": [  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_796601d9d67faf5348eeb26d0724347cea67ddc.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_d830cfcd0550fbbe4d8e0803e93c929c3abb02056.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_13bb35638c0267759276511e64773b37a51.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_0f8cf82d019c614760fd20801fbba4001da7e77.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_ef82850f036dac5772cb02d1116ea13eb163.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_76f6730dbb911650ba1fa1f5b4ba638b5beea.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_888cd373d78c3cf3a7e87026ebba15895e0670.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_ef82850f036dac5772cb02d1116ea13eb163.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_2254a50f27951f9028b00b93a7f2ed7aac1e13.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_54b9c26b028c64d5f8a5316f1ae6208953ed84d.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_2254a50f27951f9028b00b93a7f2ed7aac1e13.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_3526661949cc3966bd966feaa5a1afad532257bc.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_63d2733b9b4ace01a41d5ba8afdf653245d05d54a.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_f7046859593aefc23a0614f4686e2814405035.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_bb2af3e83a0385ff2055f2ab9697cd83e351b7.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_fb8e5e2a29c64e2898315c6b5d8e0898e8f91.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_0db84c628a798e3ca5769abda119beel358008.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_18e9aea2715f0407ee05e206073927a648db60d73.600x338.jpg?t=1729703045",  
        "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/ss_2514675f364079b754b820cbc8b2e7c331d56a26.600x338.jpg?t=1729703045"  
      ],  
      "categorias": [  
        "Multi-player",  
        "Value Anti-Cheat enabled",  
        "Stats",  
        "Cross-Platform Multiplayer",  
        "Steam Trading Cards",  
        "Steam Workshop",  
        "In-App Purchases",  
        "Remote Play on Phone",  
        "Remote Play on Tablet",  
        "Remote Play on TV",  
        "Steam Timeline"  
      ],  
      "desarrolladores": [  
        "Value"  
      ],  
      "editores": [  
        "Value"  
      ],  
      "f2p": true,  
      "lanzamiento": "21 Aug, 2012",  
      "generos": [  
        "Action",  
        "Free to Play"  
      ],  
      "imagen_cabeza": "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/header.jpg?t=1729703045",  
      "imagen_cabeza_v5": "https://shared.akamai.steamstatic.com/store_item_assets/steam/apps/730/capsule_184x69.jpg?t=1729703045",  
      "nombre": "Counter-Strike 2",  
      "pagina_web": "https://counter-strike.net/",  
      "precio": {  
        "descuento": null,  
        "precio_inicial": null  
      },  
      "tipo": "game"  
    ]  
  ]  
}
```

Osiris - Desarrollo

Los argumentos no tienen límite, y funciona así para todos los métodos GET que hemos necesitado del servidor, teniendo métodos con una cantidad exagerada de ellos, como estos casos:

```
# Ejemplos de uso:  
# http://127.0.0.1:5000/games  
# http://127.0.0.1:5000/games/?short # Te da solo nombre e id  
# http://127.0.0.1:5000/games/?page=4 # Te da la pagina 4  
# http://127.0.0.1:5000/games/?page=20&limit=100 # Te da la pagina 20 y salen 100 juegos por pagina  
# http://127.0.0.1:5000/games/?short&limit=100000 # Te salen todos los juegos en una pagina con solo el nombre e id  
# http://127.0.0.1:5000/games/?gender=Action&gender=RPG&category=Multi-player&category=Co-op&short # Filtras por generos y categorias  
# http://127.0.0.1:5000/games/?short&category=Multi-player&gender=Action&min_price=10&max_price=20&discount=1&name=Mortal  
# http://127.0.0.1:5000/games/?release_date=06-10-2024&short  
@blueprint_games.route('/', methods=['GET'])  
def get_games():  
    try:  
        logger.info("Se ha recibido una solicitud para obtener juegos")  
  
        # Info paginada  
        limit = request.args.get('limit', default=50, type=int)  
        page = request.args.get('page', default=1, type=int)  
  
        # Cantidad de datos  
        short = request.args.get('short') is not None  
        full = request.args.get('full') is not None  
  
        # Filtros  
        category_names = request.args.getlist('category', type=str)  
        gender_names = request.args.getlist('gender', type=str)  
        game_name = request.args.get('name', type=str)  
        free = request.args.get('free', type=int) # 0= de pago y 1 = gratis  
        discounted = request.args.get('discount', type=int) # 0= sin descuento 1 = en descuento  
        max_price = request.args.get('max_price', type=float)  
        min_price = request.args.get('min_price', type=float)  
        release_date = request.args.get('release_date', type=str)  
  
        offset = (page - 1) * limit  
  
        # Registrar argumentos para el logger
```

```
# Ejemplos de ENDPOINTS:  
# http://127.0.0.1:5000/users -> Saca todos los usuarios  
# http://127.0.0.1:5000/users/?short -> Saca todos los usuarios de manera reducida  
# http://127.0.0.1:5000/users/?page=1 -> Saca la primera página  
# http://127.0.0.1:5000/users/?limit=2 -> Limita las consultas empezando por el ID 0  
# http://127.0.0.1:5000/users/?full -> Devuelve todos los campos  
# http://127.0.0.1:5000/users/?estado=activo&estado=inactivo&short&limit=2 -> Ejemplo combinado con '&'  
  
# Este método es para sacar la información de usuarios de manera general. Realiza consultas de manera global de usuarios, filtrando por campos  
# como número de amigos de un usuario, de solicitudes o estados. Tiene otros métodos anidados para mejorar la legibilidad.  
#  
@blueprint_users.route('/', methods=['GET'])  
def get_users():  
    limit = request.args.get('limit', default=50, type=int)  
    page = request.args.get('page', default=1, type=int)  
    short = request.args.get('short') is not None  
    full = request.args.get('full') is not None  
  
    estado = request.args.getlist('estado', type=str)  
    id_usuario = request.args.get('id', type=int)  
    solo_amigos = request.args.get('amigos') is not None  
    solo_solicitudes = request.args.get('solicitudes') is not None  
    estado_solicitudes = request.args.get('estado_solicitudes')  
    tipo_solicitudes = request.args.get('tipo')
```

En los únicos casos en los que cambia esta dinámica es cuando usamos un método PUT, o POST lo cual no podemos hacer desde el navegador, nos obliga a usar postman para ello, por ejemplo en este método que sirve para mandar solicitudes de amistad de un usuario a otro, es necesario pasarle información en el body, ya que es más seguro que en el propio enlace:

Osiris - Desarrollo

```
# Método para enviar solicitud de amistad
# Endpoint: POST http://127.0.0.1:5000/friend_list/solicitudes_amistad
# Ejemplo de uso con postman
# {
#     "de_usuario_id": 14,
#     "para_usuario_id": 15
# }
@blueprint_friend_list.route('/solicitudes_amistad', methods=['POST'])
def enviar_solicitud_amistad():
    data = request.json
    de_usuario_id = data.get('de_usuario_id')
    para_usuario_id = data.get('para_usuario_id')
```

The screenshot shows the Postman interface. On the left, there's a sidebar with a tree view of 'Metodos API' containing various endpoints like 'Comprar', 'Recargar saldo', etc. The main area shows a 'POST' request to 'http://127.0.0.1:5000/friend_list/solicitudes_amistad'. The 'Body' tab is selected, showing raw JSON input:

```
1 {
2     "de_usuario_id": 11,
3     "para_usuario_id": 1
4 }
```

After sending the request, the response is shown in the 'Body' tab:

```
{ } JSON > Preview > Visualize | 201 CREATED | 13 ms | 189 B | Save Response | ...
```

The response body is:

```
1 {
2     "mensaje": "Solicitud enviada con éxito"
3 }
```

SQLAlchemy

Para usar SQLAlchemy, debemos declarar el modelo de datos, para ello las clases se declaran haciendo referencia directamente a las columnas de la base de datos, de esta forma

```
# Esta es la forma de declarar el modelo de datos de la base de datos
# para que sqlalchemy pueda usarlo como si fuesen objetos
Base = declarative_base()

# === JUEGOS ===
class Juegos(Base):
    __tablename__ = 'juegos'

    steam_appid = Column(Integer, primary_key=True)
    nombre = Column(Text)
    tipo = Column(Text)
    required_age = Column(Integer)
    is_free = Column(Boolean)
    detailed_description = Column(Text)
    short_description = Column(Text)
    fecha_salida = Column(Text)
    header_image = Column(Text)
    capsule_image = Column(Text)
    capsule_image5 = Column(Text)
    website = Column(Text)
    player_count = Column(Integer)

    price_overview = relationship("PriceOverview", back_populates="juego")
    generos = relationship("JuegoGenero", back_populates="juego")
    editores = relationship("JuegoEditor", back_populates="juego")
    desarrolladores = relationship("JuegoDesarrollador", back_populates="juego")
    categorias = relationship("JuegoCategoria", back_populates="juego")
    capturas = relationship("Capturas", back_populates="juego")
    usuarios_juego = relationship("UsuarioJuego", back_populates="juego")
```

Osiris - Desarrollo

una vez declarado el modelo de datos la sintaxis para hacer las consultas a la base de datos sería la siguiente:

```
# FILTRO: Nombre del juego
if game_name:
    query = query.filter(Juegos.nombre.ilike(f"%{game_name}%"))
    logger.debug("Se ha aplicado el filtro correctamente por el nombre", str(game_name))

# FILTRO: Gratis
if free == 1:
    query = query.filter(Juegos.is_free == True)
    logger.debug("Se ha aplicado el filtro '¿Es Gratis?' correctamente")

# FILTRO: En descuento
if discounted == 1:
    query = query.join(Juegos.price_overview).filter(
        PriceOverview.descuento.isnot(None),
        PriceOverview.descuento > "0"
    )
    logger.debug("Se ha aplicado el filtro 'En descuento' correctamente")

try:
    # FILTRO: Precio máximo
    if max_price is not None:
        query = query.join(Juegos.price_overview).filter(
            cast(PriceOverview.precio_inicial, Float) / 100 <= max_price
        )
    logger.debug("Se ha aplicado correctamente el filtro 'Precio máximo' en", str(max_price))
```

Sistema de logs

En todos los métodos, vamos poniendo logs para saber qué está haciendo el servidor internamente. Para ello simplemente importamos la librería logging, creamos el objeto indicando en qué parte estamos

```
logger = logging.getLogger('Server.friendlist')
```

y mandamos los logs de esta manera

```
logger.info(f"Se ha enviado una solicitud amistad de el user {de_usuario_id} al user {para_usuario_id}")
```

y en el main seleccionamos a donde se van a enviar los logs, y hasta que nivel se va a mostrar, en este caso nosotros los sacamos a la consola y al archivo server.log

Osiris - Desarrollo

```
def config_log_system():
    logger.setLevel(logging.DEBUG)

    # Crea el file handler y selecciona el nivel
    # El nivel elige que logs van a salir y cuales no
    fh = logging.FileHandler('server.log',encoding='utf-8')
    fh.setLevel(logging.DEBUG)

    # Crea el console handler y selecciona el nivel
    # El nivel elige que logs van a salir y cuales no
    ch = logging.StreamHandler()
    ch.setLevel(logging.DEBUG)

    # Crea un formato y lo añade al console handler
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    ch.setFormatter(formatter)
    fh.setFormatter(formatter)

    # Agrega el console handler al logger
    logger.addHandler(ch)
    logger.addHandler(fh)

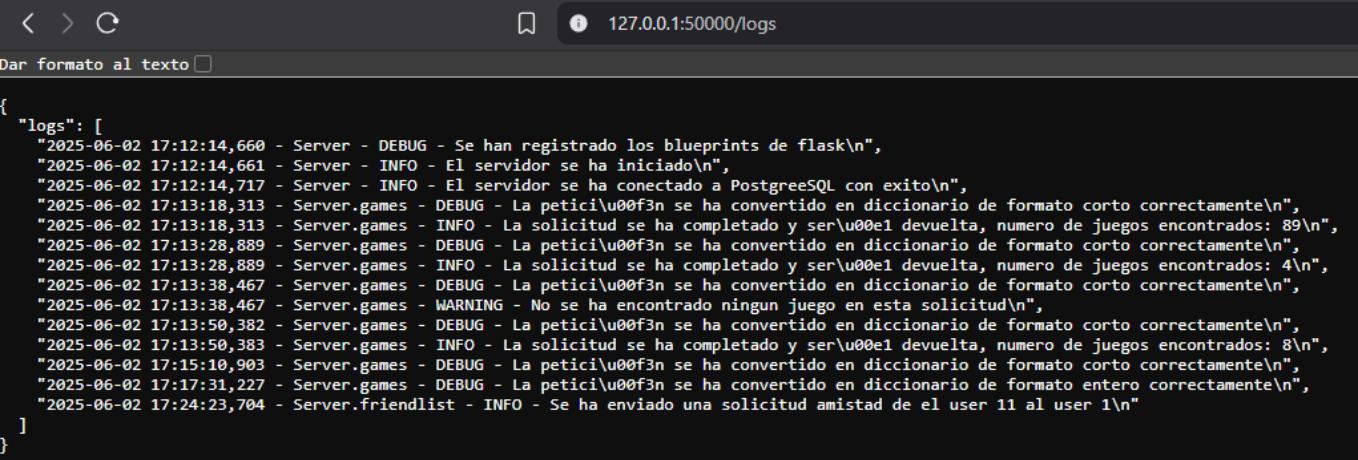
    # Estos son los comandos que deben usarse para usar los logs
    # logger.debug('debug message')
    # logger.info('info message')
    # logger.warning('warn message')
    # logger.error('error message')
    # logger.critical('critical message')
```

este fichero se ve de la siguiente forma:

```
1 2025-06-02 17:12:14,660 - Server - DEBUG - Se han registrado los blueprints de flask
2 2025-06-02 17:12:14,661 - Server - INFO - El servidor se ha iniciado
3 2025-06-02 17:12:14,717 - Server - INFO - El servidor se ha conectado a PostgreSQL con exito
4 2025-06-02 17:13:18,313 - Server.games - DEBUG - La petición se ha convertido en diccionario de formato corto correctamente
5 2025-06-02 17:13:18,313 - Server.games - INFO - La solicitud se ha completado y será devuelta, numero de juegos encontrados: 89
6 2025-06-02 17:13:28,889 - Server.games - DEBUG - La petición se ha convertido en diccionario de formato corto correctamente
7 2025-06-02 17:13:28,889 - Server.games - INFO - La solicitud se ha completado y será devuelta, numero de juegos encontrados: 4
8 2025-06-02 17:13:38,467 - Server.games - DEBUG - La petición se ha convertido en diccionario de formato corto correctamente
9 2025-06-02 17:13:38,467 - Server.games - WARNING - No se ha encontrado ningun juego en esta solicitud
10 2025-06-02 17:13:50,382 - Server.games - DEBUG - La petición se ha convertido en diccionario de formato corto correctamente
11 2025-06-02 17:13:50,383 - Server.games - INFO - La solicitud se ha completado y será devuelta, numero de juegos encontrados: 8
12 2025-06-02 17:15:10,903 - Server.games - DEBUG - La petición se ha convertido en diccionario de formato corto correctamente
13 2025-06-02 17:17:31,227 - Server.games - DEBUG - La petición se ha convertido en diccionario de formato entero correctamente
14 2025-06-02 17:24:23,704 - Server.friendlist - INFO - Se ha enviado una solicitud amistad de el user 11 al user 1
15
```

y lo generamos cada vez que se enciende el servidor, pero eso no es todo, también lo subimos a un endpoint de la api

Osiris - Desarrollo



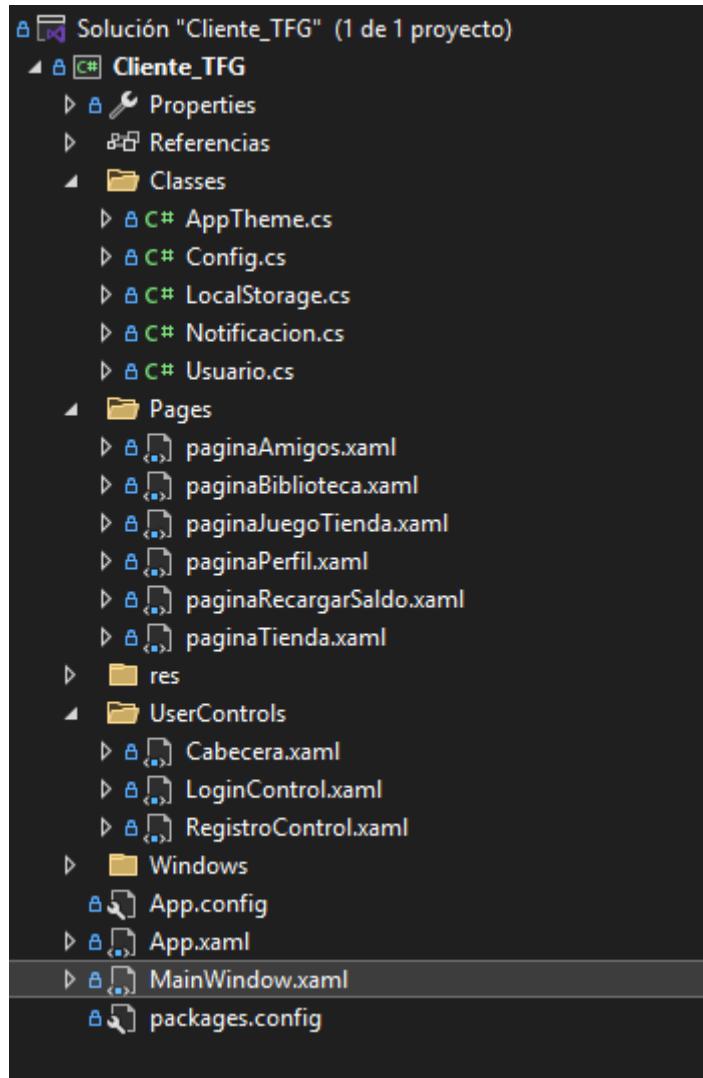
```
{ "logs": [ "2025-06-02 17:12:14,660 - Server - DEBUG - Se han registrado los blueprints de flask\n", "2025-06-02 17:12:14,661 - Server - INFO - El servidor se ha iniciado\n", "2025-06-02 17:12:14,717 - Server - INFO - El servidor se ha conectado a PostgreSQL con exito\n", "2025-06-02 17:13:18,313 - Server.games - DEBUG - La petici\u00f3n se ha convertido en diccionario de formato corto correctamente\n", "2025-06-02 17:13:18,313 - Server.games - INFO - La solicitud se ha completado y se ha devuelto, numero de juegos encontrados: 89\n", "2025-06-02 17:13:28,889 - Server.games - DEBUG - La petici\u00f3n se ha convertido en diccionario de formato corto correctamente\n", "2025-06-02 17:13:28,889 - Server.games - INFO - La solicitud se ha completado y se ha devuelto, numero de juegos encontrados: 4\n", "2025-06-02 17:13:38,467 - Server.games - DEBUG - La petici\u00f3n se ha convertido en diccionario de formato corto correctamente\n", "2025-06-02 17:13:38,467 - Server.games - WARNING - No se ha encontrado ningun juego en esta solicitud\n", "2025-06-02 17:13:50,382 - Server.games - DEBUG - La petici\u00f3n se ha convertido en diccionario de formato corto correctamente\n", "2025-06-02 17:13:50,383 - Server.games - INFO - La solicitud se ha completado y se ha devuelto, numero de juegos encontrados: 8\n", "2025-06-02 17:15:10,903 - Server.games - DEBUG - La petici\u00f3n se ha convertido en diccionario de formato corto correctamente\n", "2025-06-02 17:17:31,227 - Server.games - DEBUG - La petici\u00f3n se ha convertido en diccionario de formato entero correctamente\n", "2025-06-02 17:24:23,704 - Server.friendlist - INFO - Se ha enviado una solicitud amistad de el user 11 al user 1\n" ] }
```

de esta forma permitimos que se puedan ver los logs del servidor en tiempo real, lo que nos permite hacer cosas como que haciendo un html con un pequeño script, creamos una vista web mas profesional para que el admin pueda leer los logs en tiempo real.

Osiris - Desarrollo

Cliente

Estructura



Interfaz y Experiencia de Usuario

WPF nos ha ofrecido total libertad a la hora de diseñar la UI. Hemos implementado:

- Navegación por ventanas con Frame y Page.
- Estilos personalizados y plantillas para mantener una estética coherente.
- Notificaciones visuales (toasts) para avisos y errores.
- Vinculación de datos (Data Binding) para mantener sincronizados los datos con los elementos de la interfaz.

Desarrollo de la aplicación cliente

El desarrollo del cliente ha sido de forma escalada y modular, escalada en el sentido de que se ha

Osiris - Desarrollo

empezado desde lo más básico e ir aumentando progresivamente el desarrollo de la misma y modular en el sentido de que cada página, ventana, clase, hemos decidido hacerla independiente o lo menos dependiente posible de las demás para así, cada miembro pueda trabajar cómodamente en una parte sin interferir en el trabajo del otro.

La aplicación comienza en la clase “App.xaml.cs”. Esta clase contiene el método principal encargado del inicio de la app, llamado “Application_Startup”, este método incluye las verificaciones necesarias para:

1. Mirar si el usuario tiene guardado un token de inicio de sesión (para el inicio sin contraseña), en caso positivo, iniciar sesión automáticamente e iniciar la ventana principal.

```
if (File.Exists(rutaToken))
{
    string tokenGuardado = File.ReadAllText(rutaToken);

    using (HttpClient client = new HttpClient())
    {
        var datos = new { token = tokenGuardado };
        var contenido = new StringContent(JsonConvert.SerializeObject(datos), Encoding.UTF8, "application/json");

        try
        {
            HttpResponseMessage resp = await client.PostAsync($"http://:{Config.Puerto}/login/validar_token", contenido);

            if (resp.IsSuccessStatusCode)
            {
                string json = await resp.Content.ReadAsStringAsync();
                dynamic data = JsonConvert.DeserializeObject(json);
                int idUsuario = data.id_usuario;

                MainWindow mainWindow = new MainWindow(idUsuario);
                mainWindow.Show();
                return;
            }
        }
    }
}
```

2. En caso de que la respuesta sea negativa, es decir, no tiene token de inicio de sesión, abrimos automáticamente la ventana de login para que el usuario pueda iniciar sesión.

```
else
{
    //Token inválido o expirado - abrir login
    LoginWindow loginWindow = new LoginWindow();
    loginWindow.Show();
    return;
}
```

3. En caso de que suceda un error en el proceso de verificación de token con el servidor, querrá decir que el usuario, no dispone de conexión, en este caso especial, buscamos si el usuario tiene datos locales en su ordenador, y de ser afirmativo, cargamos esos datos para lograr mostrarle su biblioteca y que pueda seguir ejecutando sus juegos

```
catch (Exception)
{
    //FALLÓ LA CONEXIÓN - INTENTAR MODO OFFLINE
    int idUsuarioOffline = LocalStorage.CargarIdUsuarioLocal();

    if (idUsuarioOffline != -1)
    {
        MainWindow mainWindow = new MainWindow(idUsuarioOffline);
        mainWindow.Show();
        MessageBox.Show("No hay conexión, se abrió la aplicación en modo offline.");
    }
    else
    {
        MessageBox.Show("No hay conexión y no hay datos guardados. Debes iniciar sesión cuando tengas conexión.");
        LoginWindow loginWindow = new LoginWindow();
        loginWindow.Show();
    }
    return;
}
```

Una vez que el usuario ha iniciado sesión, “MainWindow” es la encargada de los siguientes pasos, comenzando con la carga de la ventana como tal, la ventana principal, el userControl de la cabecera, etc. Después de eso, la clase “MainWindow” se convierte en el núcleo de interacción visual del usuario con la aplicación. Es la ventana principal que contiene el sistema de navegación entre páginas (usando framePrincipal) y el UserControl de la cabecera, que actúa como menú global para acceder a las distintas secciones del cliente: Biblioteca, Tienda, Amigos, Perfil, y Recarga de saldo.

Osiris - Desarrollo

Cada una de estas secciones está encapsulada en su propia página (Page) dentro del proyecto, lo que facilita el mantenimiento y la escalabilidad del código. Gracias a los eventos configurados en el UserControl de la cabecera “Cabecera_top”, la navegación entre las diferentes secciones es totalmente modular y controlada centralmente desde la clase MainWindow.

```
this.DataContext = AppTheme.Actual;
AppTheme.SetDark();
cargarTema();

framePrincipal.Navigated += FramePrincipal_Navigated;

Cabecera_top.IdUser = this.idUser;
Cabecera_top.AtrasPresionado += boton_atras_presionado;
Cabecera_top.AvanzarPresionado += boton_avanzar_presionado;
Cabecera_top.BibliotecaPresionado += boton_biblioteca_presionado;
Cabecera_top.TiendaPresionado += boton_tienda_presionado;
Cabecera_top.AmigosPresionado += boton_amigos_presionado;

Cabecera_top.VerPerfilPresionado += boton_verPerfil_presionado;
Cabecera_top.RecargarSaldoPresionado += boton_recargarSaldo_presionado;
```

Una parte clave de esta ventana es la gestión del modo online y offline. En caso de que se pueda establecer conexión con el servidor, se cargan los datos del usuario en tiempo real y se actualiza la interfaz con la información obtenida (nombre, saldo, estado, etc.). Si por el contrario no hay conexión, se accede a los datos guardados previamente en local mediante la clase LocalStorage, lo cual permite que el usuario pueda seguir disfrutando de su biblioteca y ejecutar sus juegos, aunque con funciones limitadas.

El método “GuardarDatosLoca” de la clase “LocalStorage” se encarga de persistir toda la información relevante en disco tras una sesión online exitosa, para que esté disponible en el futuro sin necesidad de conexión.

```
if (online)
{
    try
    {
        user = new Usuario(this);
        user.CargarDatos(idUser); //SOLO CARGAMOS LOS DATOS SI ESTA EN MODO ONLINE
        //MessageBox.Show(user.estado);
        Cabecera_top.CambiarEstado(user.estado);
        //GUARDAMOS TODO EL LOCAL DESPUES DE LA CARGA CORRECTA
        GuardarDatosLocal();

    }
    catch
    {
        //SI FALLA LA CARGA, PASAMOS A MODO OFFLINE AUTOMATICAMENTE Y CARGAMOS LOS DATOS GUARDADOS
        online = false;
        user = LocalStorage.CargarUsuario() ?? new Usuario(this);
        user.bibliotecaJuegos = LocalStorage.CargarBiblioteca();

        MessageBox.Show("No se pudo conectar al servidor. Modo offline activado.");
    }
}
```

El método “CargarPrimeraVentana()” es el encargado de establecer qué página se muestra inicialmente al usuario tras el login. En este caso, por defecto, se carga la página de la biblioteca “paginaBiblioteca”, pero el código está preparado para ser modificado fácilmente si se quisiera cambiar esa lógica, por ejemplo, para abrir la tienda directamente o continuar en una página que el usuario visitó por última vez.

Osiris - Desarrollo

La navegación del cliente se realiza a través del “framePrincipal”, que es un contenedor dinámico donde se cargan las distintas páginas. Se ha implementado soporte para navegación hacia atrás y adelante, similar a un navegador web, lo que mejora la experiencia del usuario. Además, ciertos eventos como la navegación a la tienda, restauran visualmente ciertos estados de opacidad para asegurar que la UI se comporte correctamente tras cambios de vista y animaciones.

```
1 referencia
private void boton_avanzar_presionado(object sender, RoutedEventArgs e)
{
    if (framePrincipal.CanGoForward)
        framePrincipal.GoForward();
}

1 referencia
private void boton_biblioteca_presionado(object sender, RoutedEventArgs e)
{
    framePrincipal.Navigate(new paginaBiblioteca(this));
}

1 referencia
private void boton_tienda_presionado(object sender, RoutedEventArgs e)
{
    framePrincipal.Navigate(new paginaTienda(this));
}

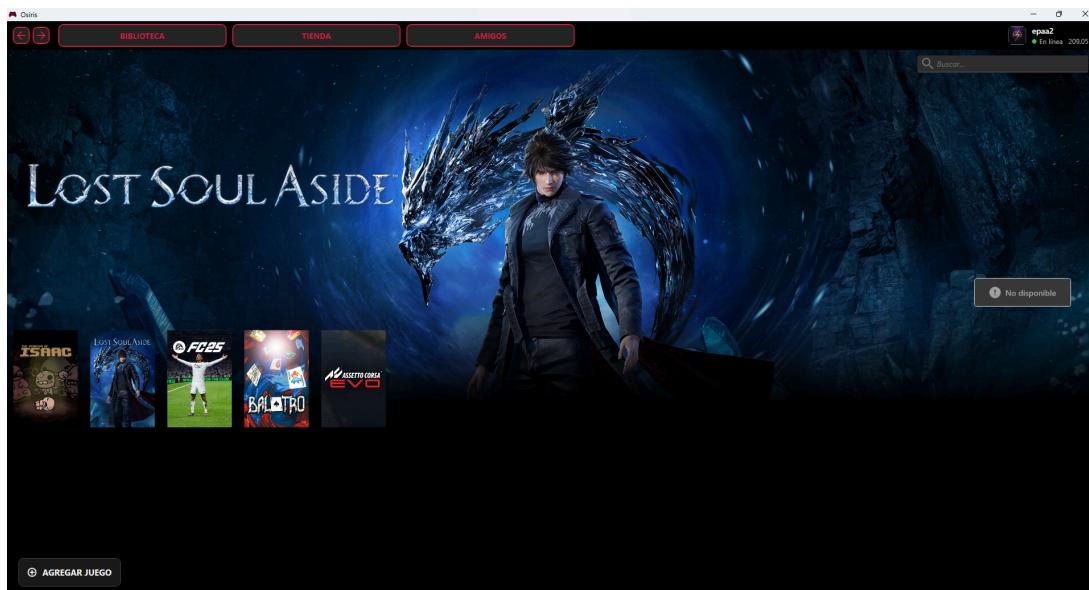
1 referencia
private void boton_amigos_presionado(object sender, RoutedEventArgs e)
{
    framePrincipal.Navigate(new paginaAmigos(this));
}

1 referencia
private void boton_verPerfil_presionado(object sender, RoutedEventArgs e)
{
    framePrincipal.Navigate(new paginaPerfil(this));
}
```

Interfaz gráfica de la aplicación

Para la interfaz gráfica de esta aplicación hemos optado por ventanas o páginas muy intuitivas que permitan al usuario final acceder al contenido que busca con los mínimos clicks posibles, para ello, hemos dividido la app en 4 grandes páginas:

Biblioteca:

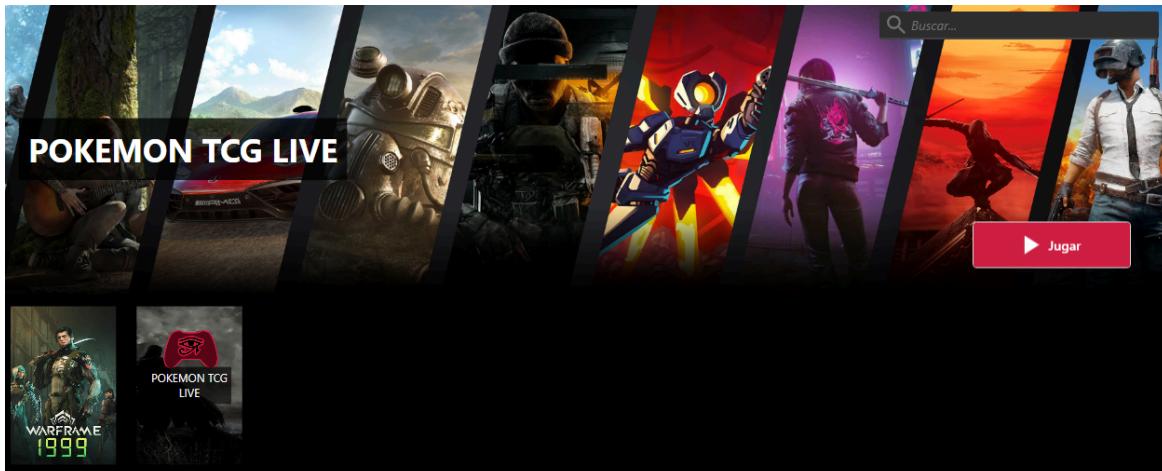


Osiris - Desarrollo

Esta página o ventana, muestra el contenido de nuestra librería de videojuegos, nada más crear una cuenta, la biblioteca estará vacía por lo que esta ventana no cambiará hasta que se realice la primera compra. Una vez tengamos uno o más videojuegos adquiridos con nuestra cuenta, la ventana muestra todos y cada uno de los juegos que tiene nuestra cuenta. La ventana se divide en varios bloques:

- **Imagen de fondo:** Esta es la imagen que aparece como cabecera dentro de nuestra biblioteca convirtiendo esta interfaz en una mucho más estética e intuitiva ya que la imagen tiene referencia al juego seleccionado, es decir, esta imagen cambiará dependiendo del juego seleccionado para que el usuario sepa en todo momento qué es lo que está viendo.
- **Logo del juego:** Otra imagen que representa el logo con el nombre del juego para, nuevamente, el usuario sepa en todo momento qué juego está viendo, en caso de no existir esta imagen, se mostrará el nombre del juego en grande.
- **Contenedor de videojuegos:** Lo hemos llamado contenedor porque es el lugar donde se visualizarán todos los juegos que el usuario posee. Para visualizar este contenido hemos optado de nuevo por una forma visual en la que podamos quitarnos de títulos que embarren nuestra interfaz, cada juego es una imagen vertical que al seleccionarla, nos mostrará los datos del juego en los dos apartados mencionados anteriormente
- **Barra de búsqueda:** Para poder buscar dentro de nuestras bibliotecas, hemos implementado una barra de búsqueda sencilla y fácil de usar, cada imagen del contenedor mencionado en el punto anteriormente, está vinculado a un nombre (el nombre del juego) por lo que al realizar una búsqueda, sólo nos mostrará el o los juegos que coincidan con el patrón descrito en este cuadro de búsqueda, si la búsqueda no coincide con ningún juego, se informa al usuario de que no dispone de este título en su biblioteca.
- **Botón para agregar juegos:** Para poder agregar juegos a nuestra biblioteca que no pertenecen o existen en nuestra tienda, hemos desarrollado este botón. Gracias a él, vamos a poder agregar cualquier juego, aplicación, programa, etc, que el usuario quiera y disponga en su dispositivo. Tan solo hay que hacer clic en el botón de “AGREGAR JUEGO”, seleccionar el ejecutable o exe y listo, el juego se añadirá a tu biblioteca y podrás ejecutarlo desde la misma app. Mediante un contextMenu, también podemos eliminar esta clase de juegos agregados manualmente por si el usuario no desea tenerlo más en la biblioteca. Tan solo hay que hacer clic derecho en la carátula y seleccionar eliminar.

Un juego agregado de esta forma se vería así:



- **Menú de juegos:** Y hablando de este contextmenu, cada carátula de cada juego, dispone de un menú, independientemente de si el juego es de la tienda o agregado manualmente. Los juegos de la tienda disponen de tres opciones, “Jugar”, “Cambiar ejecutable” y “Ver detalles”. En

Osiris - Desarrollo

cambio, los juegos agregados manualmente, solo disponen de las opciones en el menú, "Jugar" y "Eliminar de la biblioteca" el cual ya explicamos en el punto anterior.

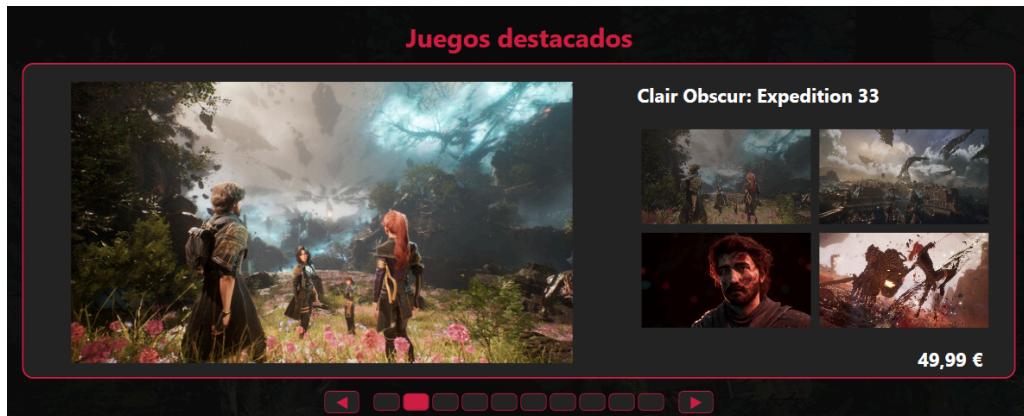
Tienda:



La página de la tienda está diseñada para ofrecer una experiencia de compra intuitiva y visualmente atractiva, dividida en varias secciones que permiten al usuario descubrir fácilmente nuevos juegos, ofertas y lanzamientos destacados. La estructura principal consta de:

- **Carrusel de juegos destacados:** La función principal de este carrusel es mostrar los juegos destacados en base a la popularidad mediante controles que hacen que los juegos se vayan desplazando poco a poco. Este carrusel está compuesto por:
 - **Imagen principal:** Una imagen grande (616x353px) que representa el juego destacado actual.
 - **Título y precio:** Texto superpuesto que muestra el nombre del juego y su precio (o "Free To Play" si es gratuito).
 - **Miniaturas:** Pequeñas imágenes debajo del carrusel que muestran capturas del juego. Al pasar el ratón, se visualiza la miniatura en grande.
 - **Controles de navegación:** Botones para avanzar/retroceder manualmente y puntos indicadores para saltar a un juego específico.
 - **Animaciones:** Transiciones suaves (fade-in/fade-out) al cambiar entre juegos.

Osiris - Desarrollo



- **Ofertas especiales:** Este apartado nos muestra juegos en oferta, ordenados por su popularidad en la app o el porcentaje de descuento, consta de seis juegos y cada juego está compuesto por:
 - **Imagen del juego:** Cabecera visual (160px de alto).
 - **Descuento:** Porcentaje en un fondo llamativo (ej: "-50%").
 - **Precio final:** Precio con descuento en negrita.



- **Ofertas con precio especificado:** Este apartado pretende mostrar cuatro juegos con un precio específico que los administradores podrán cambiar desde el servidor para que, así, los juegos puedan cambiar diariamente, semanalmente, mensualmente, etc. Está compuesto por los mismo componentes que las ofertas especiales, a diferencia que la imagen es un tanto más pequeña. También aparecen ordenados por popularidad y precio.



- **Nuevos lanzamientos:** Este componente es una lista vertical de juegos recientemente agregados a la aplicación filtrados por fecha de lanzamiento, cada juego dentro de este componente está compuesto por:
 - **Imagen:** A la izquierda (231x87px).
 - **Información central:** Donde se visualiza tanto el nombre del videojuego, como más abajo, los géneros a los que este pertenece
 - **Precio y fecha:** Como dice el título, este apartado muestra la fecha de lanzamiento y el precio completo del juego a su derecha

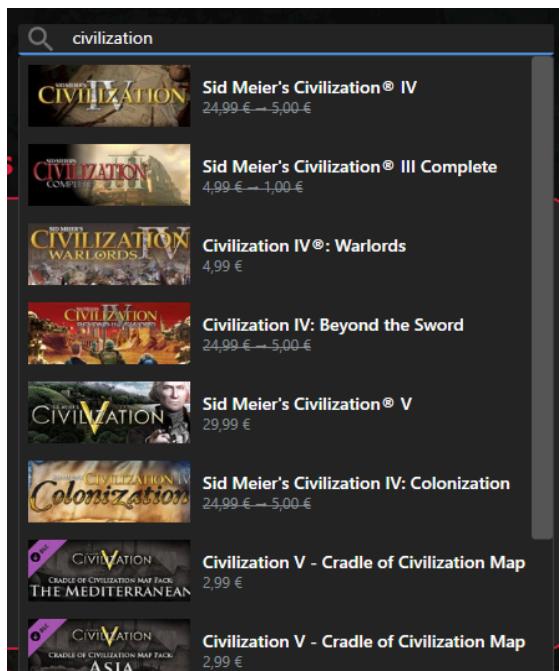
Osiris - Desarrollo

Nuevos lanzamientos			
	F1® 25 Sports, Simulation, Racing	Fecha de lanzamiento: 30 May, 2025	59,99 €
	Lost Soul Aside™ Action, Adventure, RPG	Fecha de lanzamiento: 30 May, 2025	69,99 €
	Onimusha 2: Samurai's Destiny Action	Fecha de lanzamiento: 22 May, 2025	29,99 €
	MOBILE SUIT GUNDAM SEED BATTLE DESTINY REMASTERED Action, RPG, Casual	Fecha de lanzamiento: 21 May, 2025	39,99 €
	DOOM: The Dark Ages Action	Fecha de lanzamiento: 14 May, 2025	79,99 €
	MotoGP™25 Sports, Simulation, Racing	Fecha de lanzamiento: 30 Apr, 2025	59,99 €
	Clair Obscur: Expedition 33 Action, RPG	Fecha de lanzamiento: 24 Apr, 2025	49,99 €
	The Hundred Line -Last Defense Academy- Adventure, Simulation	Fecha de lanzamiento: 23 Apr, 2025	59,99 €
	The Elder Scrolls IV: Oblivion Remastered RPG	Fecha de lanzamiento: 22 Apr, 2025	54,99 €

- **Barra de búsqueda:** Su funcionalidad principal es la búsqueda en tiempo real con delay de 400 ms para evitar sobrecarga. También hemos limitado la visualización de juegos de la búsqueda con un máximo de 10 juegos para evitar sobrecargas. Cada búsqueda incluye estos componentes:

- **Imagen:** Una pequeña imagen que hace referencia al juego
- **Nombre y precio:** Ambos juntos uno encima del otro, además, si el juego tiene descuento, también se mostrará.

Cuando se hace click en un resultado, se abrirá la página que muestra todos los detalles del juego para poder adquirirlo



Osiris - Desarrollo

Juego de la tienda:

Es una pequeña página que muestra los detalles de un juego cuando le haces click desde la tienda.

The screenshot shows a game store interface. At the top, there are tabs for 'BIBLIOTECA' (Library), 'TIENDA' (Store), and 'AMIGOS' (Friends). The store tab is active. In the center, there's a large image of the game's cover art, which depicts a dark, apocalyptic landscape with a massive, horned demon-like creature. Below the cover are several smaller thumbnail images of the game's art. To the right of the main image is a product box with the game's title 'DOOM: THE DARK AGES', a price of '79,99 €', and a 'COMPRAR' (Buy) button. Below the price, there's a brief description: 'DOOM: The Dark Ages is the prequel to the critically acclaimed DOOM (2016) and DOOM Eternal that tells an epic cinematic story of the DOOM Slayer's rage. Players will step into the blood-stained boots of the DOOM Slayer, in this never-before-seen dark and sinister medieval war against Hell.' It also includes the release date 'Fecha de lanzamiento: 14 May, 2025', developer 'Desarrollador: id Software', and publisher 'Editor: Bethesda Softworks'. Below this section is a 'PREORDER' (Preorder) button with a 'PRE-ORDER BONUS' offer. Further down the page, there's a large, ornate decorative banner with the text 'DISCOVER UNKNOWN REALMS' in red, set against a dark, atmospheric background of a ruined castle or city. Below this banner, there's another section titled 'DISCOVER UNKNOWN REALMS' with a descriptive paragraph about the game's setting and challenges. At the bottom of the page, there are two tables: one for 'Categorías' (Categories) listing 'Single-player', 'Full controller support', and 'Family Sharing'; and one for 'Géneros' (Genres) listing 'Action'.

Categorías:
Single-player
Full controller support
Family Sharing

Géneros:
Action

Osiris - Desarrollo

Amigos:

La clase “paginaAmigos” representa la lógica de interacción para la página de amigos en una aplicación cliente de escritorio. Su propósito principal es gestionar y facilitar la interacción entre usuarios, incluyendo la administración de la lista de amigos, el manejo de solicitudes de amistad y la provisión de un sistema de chat en tiempo real. Esta página se encarga de la comunicación con un servidor backend para obtener y enviar datos relacionados con estas funcionalidades.

Funcionalidades Clave: La página “paginaAmigos” ofrece las siguientes funcionalidades principales a los usuarios:

- **Gestión de la Lista de Amigos:**
 - Obtención y visualización de la lista de amigos desde el servidor.
 - Visualización del estado de cada amigo (Conectado, Ausente, Ocupado, Desconectado).
 - Selección de amigos para iniciar un chat.
- **Gestión de Solicitudes de Amistad:**
 - Recepción y visualización de solicitudes de amistad pendientes.
 - Aceptación de solicitudes de amistad, lo que implica la eliminación de la solicitud y la creación de un chat privado en el servidor.
 - Rechazo de solicitudes de amistad, eliminando la solicitud.
 - Envío de solicitudes de amistad a otros usuarios mediante su código de amigo.
 - Actualización periódica automática de las solicitudes pendientes.
- **Funcionalidad de Chat en Tiempo Real:**
 - Envío de mensajes a un amigo seleccionado.
 - Recepción de mensajes en tiempo real a través de Socket.IO.
 - Carga del historial de chat con un amigo específico desde el servidor.
 - Unión y abandono de salas de chat en el servidor para gestionar conversaciones activas.
 - Detección y creación automática de chats privados si no existe uno en común con un amigo.
- **Notificaciones al Usuario:**
 - Muestra notificaciones informativas, de éxito, advertencia o error en la interfaz de usuario para informar al usuario sobre el estado de las operaciones (ej., solicitud enviada, error de conexión).
- **Gestión del Código de Amigo del Usuario Actual:**
 - Muestra el código de amigo del usuario actual.
 - Permite copiar el código de amigo al portapapeles.

Flujos de Trabajo Importantes

- **Inicialización de la Página:**
 1. Se inicializan los componentes de la UI (InitializeComponent()).
 2. Se configura la notificación y el temporizador de actualización.
 3. Se inicializan los datos del código de amigo del usuario actual.
 4. Se aplican los temas visuales actuales de la aplicación.
 5. Se inician tareas asíncronas para actualizar solicitudes pendientes, obtener amigos del servidor y conectar al servidor Socket.IO.
- **Envío y Recepción de Mensajes:**
 1. Cuando el usuario escribe un mensaje y presiona Enter o el botón "Enviar", el método EnviarMensaje() se invoca.
 2. EnviarMensaje() verifica si hay contenido y si hay un amigo seleccionado, y si el cliente Socket.IO está conectado.
 3. Si todo es válido, emite el evento enviar_mensaje al servidor Socket.IO con el ID del remitente, el ID del chat actual y el contenido del mensaje.

Osiris - Desarrollo

4. El campo de texto del mensaje se limpia.
5. Cuando el servidor envía un nuevo mensaje, el evento cliente.On("nuevo_mensaje") se dispara.
6. El mensaje recibido se deserializa (parsea el JSON) para extraer el contenido, el nombre de usuario y el ID del chat.
7. Si el mensaje pertenece al chat actualmente visible (idChatActual), se agrega a la interfaz de usuario (AgregarMensajeAChat()) y se hace scroll al final del chat.

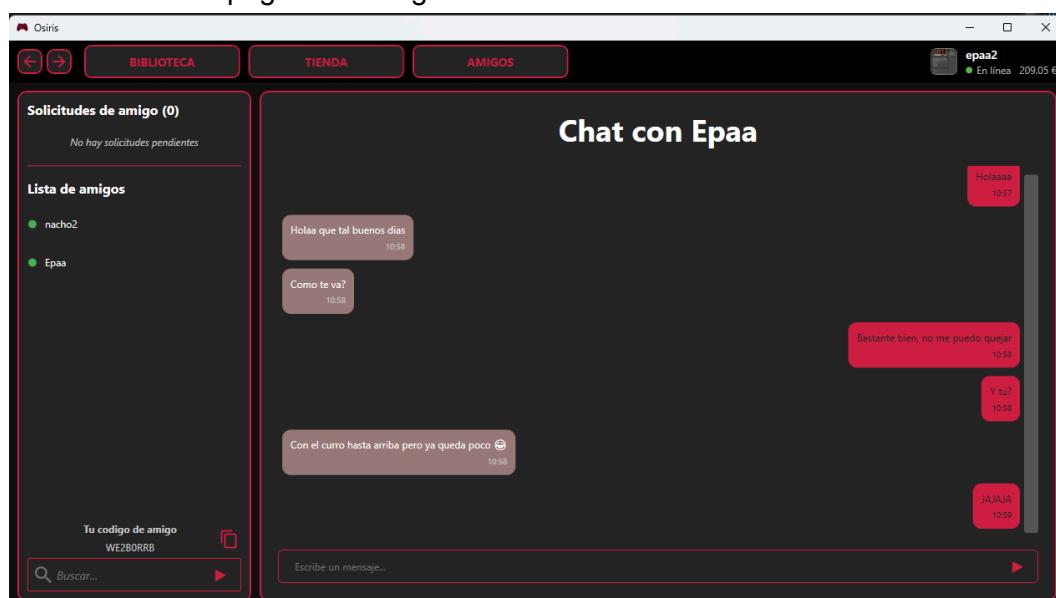
- **Aceptación/Rechazo de Solicitudes de Amistad:**

1. Cuando el usuario hace clic en "✓" o "✗" en una solicitud.
2. Se llama a AceptarSolicitudAmistad() o RechazarSolicitudAmistad().
3. La solicitud se elimina localmente de solicitudesPendientes.
4. Se invoca una función asíncrona (AceptarSolicitudAmistadEnServidorAsync() o RechazarSolicitudAmistadEnServidorAsync()) para notificar al servidor.
5. En caso de aceptación, también se invoca CrearGrupoChatEnServidorAsync() para establecer un chat privado.
6. La UI de solicitudes se actualiza (ActualizarSolicitudesPendientes()).
7. Si se acepta, se llama a ObtenerAmigosDelServidorAsync() para actualizar la lista de amigos.

- **Envío de Solicitudes de Amistad:**

1. El usuario introduce un código de amigo en el campo txtBuscarAmigos y hace clic en "Enviar Solicitud".
2. Se valida que el campo no esté vacío.
3. Se llama a GetUsuarioPorCodigoDeAmigo() para buscar al usuario por su código de amigo en el servidor.
4. Si el usuario existe, se llama a MandarSolicitudDeAmistadAsync().
5. MandarSolicitudDeAmistadAsync() envía una solicitud POST al servidor con el ID del remitente y del destinatario.
6. Se muestra una notificación de éxito o error y se limpia el campo de texto si la solicitud es exitosa.

Un pequeño vistazo a la página de amigos:



Perfiles:

Osiris - Desarrollo

La clase paginaPerfil representa la lógica de interacción para la página de perfil de usuario en una aplicación cliente de escritorio. Su propósito principal es mostrar la información personal del usuario autenticado, permitir la edición de ciertos datos del perfil (como nombre de usuario y descripción), gestionar la foto de perfil, y visualizar los últimos juegos comprados por el usuario. Esta página se encarga de la comunicación con un servidor backend para obtener y actualizar los datos del perfil.

Funcionalidades Clave: La página “paginaPerfil” ofrece las siguientes funcionalidades principales a los usuarios:

- **Visualización de Datos del Perfil:**
 - Muestra el nombre de usuario, el nombre de la cuenta y la descripción del usuario actual.
 - Muestra la foto de perfil del usuario.
 - Muestra el número total de amigos del usuario.
- **Edición del Perfil:**
 - Permite editar el nombre de usuario y la descripción del perfil.
 - Guarda los cambios realizados en el servidor mediante una solicitud PATCH.
 - Valida que el nombre de usuario no esté vacío.
 - Proporciona retroalimentación visual y notificaciones (ventanas modales) sobre el éxito o fracaso de la actualización.
- **Gestión de la Foto de Perfil:**
 - Permite subir una nueva foto de perfil desde un archivo local.
 - La imagen seleccionada se sube a Imgur.
 - La URL de la imagen en Imgur se usa para actualizar la foto de perfil en el servidor.
 - La interfaz de usuario se refresca automáticamente con la nueva foto.
- **Visualización de Últimos Juegos Comprados:**
 - Carga y muestra las carátulas, nombres y fechas de compra de los tres últimos juegos adquiridos por el usuario desde el servidor.
 - Intenta cargar las imágenes de los juegos desde una caché local (AppData) primero; si no están disponibles, las descarga desde Steam.
 - Cambia el fondo de la página cuando el ratón se sitúa sobre la carátula de un juego, mostrando una imagen relacionada con el juego.
 - Si no hay juegos o menos de tres, muestra imágenes por defecto y un mensaje "Sin juego".
 - Permite al usuario abrir una "Tienda" si hace clic en una carátula por defecto.

Flujos de Trabajo Importantes

- **Inicialización y Carga de Datos del Perfil:**
 1. Al construir la paginaPerfil, se inicializan los componentes, se establece la referencia a MainWindow, se carga el tema visual y se llaman a cargarDatosUser().
 2. cargarDatosUser() toma los datos del Usuario de ventanaPrincipal y los asigna a los TextBlock correspondientes, y carga la foto de perfil.
 3. Si la aplicación está Online, se llama a CargarUltimosJuegos().
 4. Una vez cargada la página (Page_Loaded), se inician las tareas asíncronas para obtener el número de amigos (ObtenerNumeroAmigos()) y para recargar la información del usuario desde el servidor (RecargarUsuarioDesdeServidor()). Esto asegura que los datos mostrados estén actualizados.
- **Actualización de la Foto de Perfil:**
 1. El usuario hace clic en el botón para subir una imagen (btnSubirImagen_Click_1).
 2. Se abre un diálogo para seleccionar un archivo de imagen (.jpg, .jpeg, .png).
 3. La ruta del archivo se pasa a SubirImagenAlImgurAsync(), que sube la imagen a Imgur y devuelve su URL pública.

Osiris - Desarrollo

4. Si la URL de Imgur es válida, se llama a ActualizarFotoUsuarioConUrl() para enviar esta URL al servidor y asociarla con el perfil del usuario.
5. Finalmente, se invoca RecargarUsuarioDesdeServidor() para que la UI se actualice con la nueva foto de perfil.

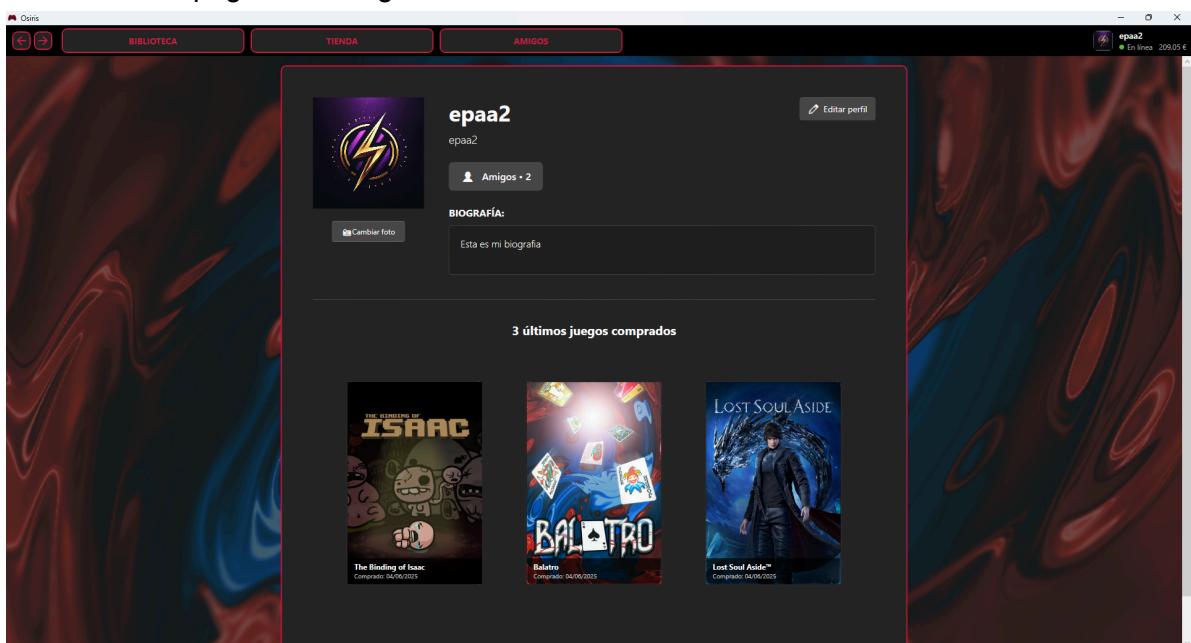
- **Edición y Guardado de Datos del Perfil:**

1. El usuario hace clic en el botón "Editar" (btnEditar_Click), lo que oculta los TextBlock de visualización y muestra los TextBox de edición, y cambia el botón "Editar" por "Guardar".
2. El usuario modifica el nombre de usuario y/o la descripción.
3. Al hacer clic en "Guardar" (btnGuardar_Click), se valida que el nuevo nombre de usuario no esté vacío.
4. Se construye un objeto JSON con los datos actualizados y se envía una solicitud PATCH al servidor para actualizar el perfil.
5. Si la actualización es exitosa, los datos se actualizan localmente (nombreUser.Text, descripcctionCuenta.Text, ventanaPrincipal.user) y se revierte la visibilidad de los elementos para mostrar la información en modo de visualización. Se muestra una notificación de éxito.

- **Carga y Visualización de Últimos Juegos:**

1. Al inicio de la página, si la aplicación está online, se llama a CargarUltimosJuegos().
2. Se realiza una solicitud GET al servidor para obtener los datos de los últimos juegos comprados por el usuario.
3. El JSON recibido se parsea y se extrae la información de los juegos.
4. Para cada juego, se intenta cargar la carátula desde una ruta local específica (AppData\ClienteTFG\imágenes\{ApId}_vertical.jpg); si no existe, se intenta descargar desde una URL de Steam (header).
5. Las imágenes y datos se asignan a los controles de UI correspondientes (caratulaJuegoX, nombreJuegoX, fechaJuegoX).
6. Se configuran los eventos MouseEnter para cambiar el fondo de la página dinámicamente y, si no hay juegos, se muestran imágenes por defecto.

Un pequeño vistazo a la página de amigos:



Osiris - Desarrollo

Manejo de logs

Al igual que en el servidor, en el cliente implementamos un sistema de logs, pero utilizando herramientas disponibles en C#. Para ello usamos System.Diagnostics, permitiendo almacenar logs con diferentes niveles (Info, Warning, Error) en archivos de texto. Esto nos ha resultado útil para la depuración y seguimiento de errores durante las pruebas.

Experiencia y Problemas durante el desarrollo

A lo largo del desarrollo, nos hemos encontrado con diferentes adversidades. En el siguiente apartado, comentamos y dejamos constancia sobre las diferentes dificultades a lo largo de la implementación del software.

- **Sacar los datos de la API de Steam:**

Tras mucho pensar, decidimos que la mejor idea era sacar los datos de los juegos de la API de Steam, pero nos topamos con el inconveniente de que la API de Steam solo permite 200 solicitudes cada 5 minutos. Cada vez que se cumple el cupo, hay que esperar 5 minutos. El API tiene más de 80.000 registros, por lo que para volcar todos los datos a nuestra base de datos, hubo un equipo que se tuvo que quedar inoperativo 3 días enteros.

- **Migrar de NoSQL a SQL:**

Nuestra primera idea fue hacer la base de datos en MongoDB, ya que teníamos curiosidad por aprenderlo, montamos la base de datos en MongoDB y utilizamos ‘pymongo’ para hacer los primeros métodos de la API. Días después, nuestra tutora del proyecto nos pidió cambiar la base de datos a un relacional, lo cual significaba rehacer lo que ya teníamos y volverlo a hacer en una base de datos relacional, y en este caso nos decantamos por PostgreSQL.

Tuvimos varios problemas para migrar de MongoDB a PostgreSQL, lo cual nos retrasó una semana.

- Tuvimos que realizar un script en Python para convertir todos los datos a SQL. El script dió muchos problemas con los tipos de datos. Dado que es la primera vez que utilizamos Python y PostgreSQL. Cada *SGBD* tiene su manera de trabajar y con los nombres de los tipos de datos tuvimos muchos errores.
- Al tener el servidor montado en Python con el framework Flask. Necesitábamos tener un modelo de datos para después realizar las consultas. Tras una pequeña búsqueda, descubrimos una herramienta llamada ‘sqlacodegen’, la cual realiza una consulta a la base de datos y coge todas las tablas, con todos las columnas y el tipo de dato generando el modelo en un script.

Primero probamos la herramienta para ver si funcionaba con otro modelo de datos y funcionaba perfectamente. El problema fue cuando lo probamos con nuestra base de datos. Al intentar automatizar el volcado de datos al modelo nos daba errores con el tipo de dato, con ligeras sospechas de que el script no estaba convirtiendo los datos de manera correcta.

La solución fué consultar a través de *cmd* todas las tablas, y generar el modelo a mano.

Osiris - Desarrollo

- **Primera vez que utilizamos Python y Flask:**

Es nuestra primera vez utilizando Python para un proyecto, dado que nunca habíamos hecho más que pequeños Scripts. Nos estamos enfrentando a problemas de sintaxis, y aprendizaje de funcionamiento de un lenguaje, como por ejemplo que sea interpretado o que tenga una sintaxis tan sencilla como por ejemplo que no haya que declarar el tipo de la variable, o al tabular las líneas de código, el lenguaje no las interpreta si no están tabuladas.

- **Primera vez que creamos una API y utilizamos una API.**

A lo largo del curso, no tuvimos la oportunidad de utilizar API o crearla. Ha supuesto un reto para nosotros este desarrollo, pero lo hemos afrontado con bastantes ganas y después de ver varios videos de youtube supimos hacerlo correctamente.

- **Primera vez que utilizamos PostgreSQL:** Como se ha comentado previamente, es la primera vez que utilizamos PostgreSQL y nos hemos tenido que adaptar a que sea objeto-relación o diferencias en sintaxis, como por ejemplo, si nombras una tabla con la primera letra en mayúscula, en PostgreSQL debemos poner comillas dobles a ese nombre para realizar una consulta, de la otra manera, si la nombras toda en minúsculas, no es necesario poner comillas dobles.

- **Puesta en marcha de la API:** Tras 3 semanas de análisis, creación de diagramas y puesta en marcha de la base de datos, empezamos a dividirnos las tareas y a echar bastantes horas para recuperar el tiempo perdido.

- **Reparto de trabajo:** Jesús se encargó principalmente del cliente, ya que era el más hábil creando interfaces gráficas, y tenía grandes ideas. Por otro lado, Ignacio y Rubén, se encargaron del servidor, de manera conjunta realizaron la conexión a la base de datos, y la modularización del proyecto con Flask. Una vez listo, se dividieron las tareas.

- **Jesús:** Lo primero en lo que se centró fue en el diseño de la tienda principal con múltiples secciones destacadas, la barra de búsqueda, la vista de cada juego en la tienda y todo lo referente a la tienda. Su siguiente prioridad fue la biblioteca, y en este punto, ya tenía gran parte del servidor funcionando, así que cambió los datos de prueba por datos reales en la tienda, se encargó del registro y login, junto a un sistema de tokens, que permite al usuario loguearse de forma automática, también hizo modificaciones en el servidor conforme lo fue necesitando.

- **Ignacio:** Se centró en la parte de la API referente a los juegos, la biblioteca de los usuarios, y las listas de las secciones destacadas de la tienda, también implementó el sistema de logs en el servidor y trabajó junto a Rubén en la vista web. Más adelante se encargó de los apartados del cliente de recargar saldo, lista de amigos y chat.

- **Rubén:** Se centró en la parte de API de Usuarios, encargándose de los apartados de la api de los datos de los usuarios, sus perfiles, el login de los mismos, la lista de amigos y los chats, así como trabajar junto a Ignacio en la elaboración de la vista web

Osiris - Desarrollo

Más adelante se encargó del apartado del cliente de perfiles de usuario y aportó en gran medida al desarrollo del chat.

- **Peticiones a la API:** Mientras desarrollamos la API decidimos empezar por los métodos GET, y nuestra forma de comprobarlo era poniendo los endpoints en el navegador, a la hora de ponernos a hacer peticiones POST, PUT, DELETE etc. No podíamos hacerlo de esa forma, y al no tener el conocimiento necesario, hicimos un pequeño endpoint con un HTML, para poder realizar estas peticiones.

Cambiar Contraseña del Usuario

ID del Usuario:

Nueva Descripción:

Nueva Contraseña:

[Actualizar](#)

[Eliminar Usuario](#)

No file chosen

Nos dimos cuenta de que hacer esto por cada endpoint era muy tedioso y decidimos investigar, hasta que dimos con Postman, una APP que permite hacer peticiones a una API seleccionando el tipo de petición, lo cual nos ahorró muchos problemas.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (My first collection), 'Environments', 'Flows', 'APIs', and 'History'. In the main area, a collection named 'My first collection' is expanded, showing two folders: 'First folder inside collection' and 'Second folder inside collection', each containing several requests. The current request being viewed is a POST to <http://127.0.0.1:5000/store/buy?user=0&game=208200>. The 'Params' tab is selected, showing 'user' with value '0' and 'game' with value '208200'. Below the params, there are sections for 'Authorization', 'Headers', 'Body', 'Scripts', and 'Settings'. At the bottom, the response is displayed as a 200 OK status with a JSON body: { "message": "Juego comprado exitosamente. Dinero restante: 939.89", "success": true }.

Osiris - Desarrollo

- **Pruebas de Chat en tiempo real:** Al querer integrar el chat en tiempo real en los clientes, tras un arduo trabajo de investigación acerca del funcionamiento de Flask con SocketIO. Nos dimos cuenta de que era imposible probarlo en Postman, dado que Postman sólo es válido en APIs REST. Entonces, creamos un chat básico para comprobar que las comunicaciones funcionan de manera correcta.

Cliente Test Chat Socket.IO

Id Usuario:

Id Chat:

[Error] Usuario no pertenece al chat

[Status] Usuario 5 se unió al chat 3

[Status] Usuario 3 se unió al chat 3

Sofía López dice: Hola buenas tardes [16:13:29]

Carlos Ruiz dice: Hola que tal [16:13:35]

Escribe tu mensaje...

Conectado al servidor Socket.IO con id: zRmnGtB5ZX6REKdzAAAB

Cliente Test Chat Socket.IO

Id Usuario:

Id Chat:

[Status] Usuario 3 se unió al chat 3

Sofía López dice: Hola buenas tardes [16:13:29]

Carlos Ruiz dice: Hola que tal [16:13:35]

Escribe tu mensaje...

Conectado al servidor Socket.IO con id: MXdytr5zb-soEgDVAAAD

- **Problemas al implementar el chat en el cliente:** Después de probar los chats, nos dimos cuenta de varios problemas, estábamos intentando establecer una conexión TCP/IP en un cliente donde no vas a estar siempre conectado a la pagina del chat, por lo que podrías perderte los mensajes, ya que no se estarían guardando de ninguna forma, por lo que tuvimos problemas para decidir cómo hacer esto, al final tuvimos la idea de cargar el historial de mensajes en la base de datos, y que cuando mandaras un mensaje por socket, también lo subieses a la base de datos, de esa forma si estas en el chat podrías recibir los mensajes en tiempo real y si no lo

Osiris - Desarrollo

estas, al entrar los recibirías de la base de datos. Implementar esta funcionalidad fue un verdadero quebradero de cabeza que se extendió hasta 2 días antes de la entrega del proyecto.

- **Problemas en el desarrollo del modo offline en Osiris:** Durante el proceso de implementación del modo offline en la aplicación Osiris, se han identificado y enfrentado varios desafíos técnicos y de diseño que han afectado tanto la estabilidad como la experiencia de usuario. A continuación, se detallan los principales problemas detectados:
 1. **Sincronización de datos entre modo online y offline:** La gestión de la sincronización entre los datos obtenidos desde la API y los almacenados localmente ha resultado más compleja de lo previsto. Se han producido inconsistencias cuando un usuario cambia información en modo online y luego accede en modo offline con datos desactualizados.
 2. **Persistencia de datos localmente:** Inicialmente, no se contaba con un sistema robusto para guardar información crítica como la biblioteca de juegos, estado de usuario o configuración personalizada. La implementación de un almacenamiento JSON local supuso un rediseño parcial de la arquitectura para asegurar que los datos se guardarán correctamente y se leyeron sin errores.
 3. **Gestión de la sesión de usuario sin conexión:** Se presentó un problema con la validación de la sesión. Al no contar con una verificación directa contra la API, se tuvo que desarrollar un sistema local de tokens que garantizara que el usuario ya había iniciado sesión previamente, sin comprometer la seguridad.
 4. **Desactivación dinámica de funciones dependientes de internet:** Algunas secciones de la app, como la tienda o las ofertas del día, requerían ocultarse o mostrar un aviso cuando no hubiera conexión. Sin embargo, al principio no se manejaban correctamente estas condiciones, provocando errores o bloqueos en la interfaz.
 5. **Gestión de errores y mensajes al usuario:** En las primeras versiones, al no poder contactar con la API, la aplicación simplemente fallaba sin mostrar información clara. Se tuvo que mejorar el sistema de logs y notificaciones para que el usuario entendiera por qué ciertas funciones no estaban disponibles.
 6. **Detección del estado de conectividad:** Aunque parecía trivial, detectar de forma fiable si el usuario tenía conexión (y diferenciar entre conexión limitada, VPN o proxy) fue más complicado de lo esperado. Esto impacta en la lógica que decide si se activa el modo offline o se intenta acceder a la API.
 7. **Modularización del código para compatibilidad dual:** Fue necesario reestructurar partes del código para que ciertas funciones pudieran operar tanto en modo online como offline, sin duplicar lógica ni romper dependencias. Esto implicó una refactorización de varias clases y controladores.

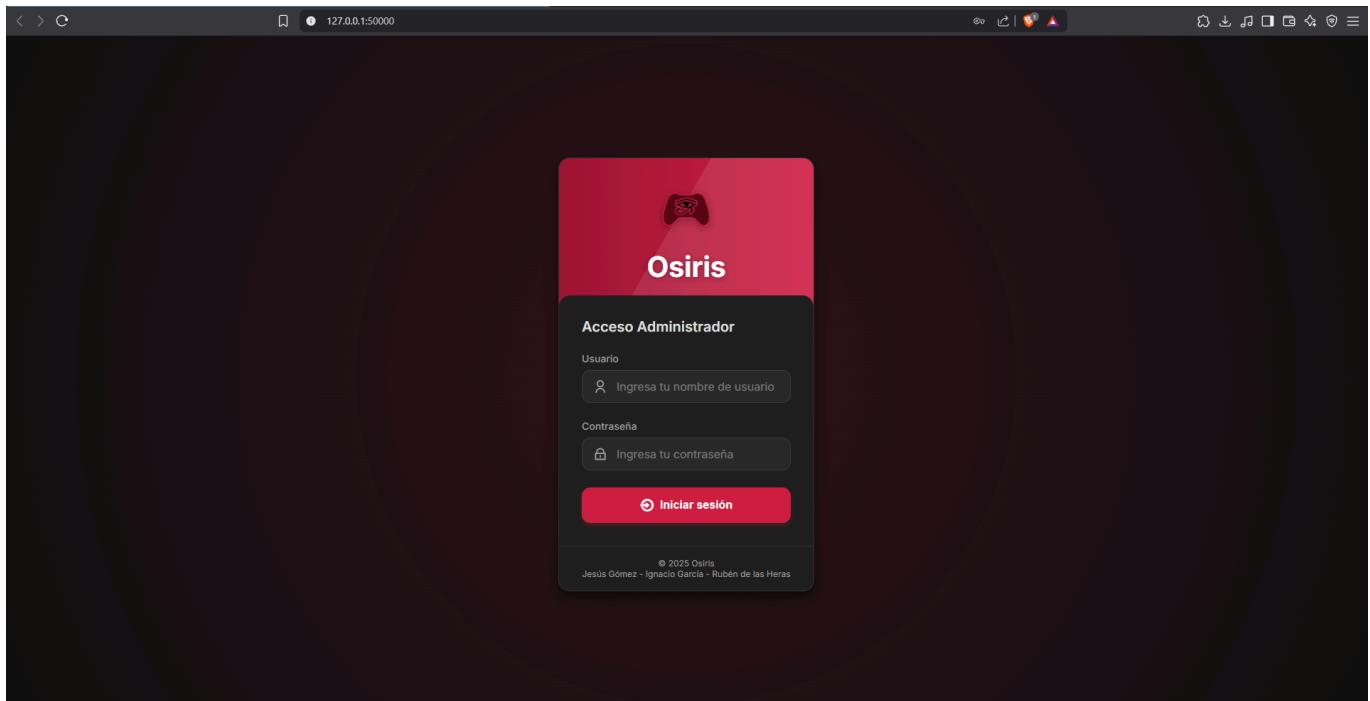
Resultados

Servidor

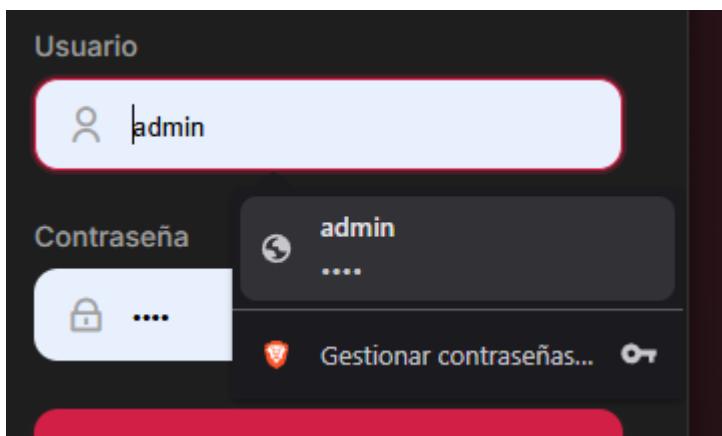
Al abrir el servidor, flask genera un servidor web para que puedas utilizar diferentes endpoints para pasar información, pero también puedes devolver páginas web a estos endpoints, por lo que hemos

Osiris - Resultados

aprovechado a crear una vista web que permite al administrador del sistema poder monitorizar el servidor, si vas al endpoint raíz, te toparas con esta página de login:



El navegador es capaz de registrar tu contraseña y guardarla para volver a completar el formulario automáticamente, lo cual es muy cómodo para el administrador.



Si consigues iniciar sesión con éxito, el administrador accede a la página principal del administrador. La cual mostrará información de la aplicación como cuantos juegos hay registrados en la base de datos y

Osiris - Resultados

cuántos usuarios están registrados en la aplicación, estos datos se actualizan en tiempo real.

The screenshot shows the 'Panel de Administración' (Admin Panel) interface. At the top, there's a red header with the text '¡Bienvenido, Administrador!' (Welcome, Administrator!) and a message 'Gestiona la base de datos del servidor desde aquí.' (Manage the server database from here). Below the header, there are three main statistics boxes:

- Juegos registrados:** 80329 (Última actualización: hoy)
- Usuarios Registrados:** 24 (Última actualización: hoy)
- Estado del servidor:** En línea (Conexión estable)

Below these boxes is a section titled 'Acciones rápidas' (Quick Actions) with two prominent red buttons:

- Ver logs del servidor** (View Server Logs)
- Ver créditos de la aplicación** (View Application Credits)

At the bottom of the panel, there's a copyright notice: © 2025 Osiris, Jesús Gómez - Ignacio García - Rubén de las Heras.

En esta página, hay 2 botones, el primero permite al usuario acceder a la página de logs y la segunda permite al usuario acceder a una página de créditos.

La página de logs es una página que aprovecha el endpoint `/logs` para mostrar una vista web de los logs más agradable a la vista y más profesional, en ella se pueden ver los mensajes que aportan información sobre los procesos por los que pasa el servidor internamente.

The screenshot shows the 'Logs del Servidor' (Server Logs) page. The page has a header with the Osiris logo and the word 'Osiris'. Below the header is a table containing the following log entries:

Timestamp	Level	Category	Message
2025-06-02 17:13:28,889	DEBUG	Server.games	La petición se ha convertido en diccionario de formato corto correctamente
2025-06-02 17:13:28,889	INFO	Server.games	La solicitud se ha completado y será devuelta, numero de juegos encontrados: 4
2025-06-02 17:13:38,467	DEBUG	Server.games	La petición se ha convertido en diccionario de formato corto correctamente
2025-06-02 17:13:38,467	WARNING	Server.games	No se ha encontrado ningun juego en esta solicitud
2025-06-02 17:13:50,382	DEBUG	Server.games	La petición se ha convertido en diccionario de formato corto correctamente
2025-06-02 17:13:50,383	INFO	Server.games	La solicitud se ha completado y será devuelta, numero de juegos encontrados: 8
2025-06-02 17:18:10,903	DEBUG	Server.games	La petición se ha convertido en diccionario de formato corto correctamente
2025-06-02 17:17:31,227	DEBUG	Server.games	La petición se ha convertido en diccionario de formato entero correctamente
2025-06-02 17:24:23,704	INFO	Server.friendList	Se ha enviado una solicitud amistad de el user 11 al user 1

La página de créditos es una página web estática en la que aparece información sobre los desarrolladores de la aplicación, una breve descripción sobre qué es la aplicación , qué tecnologías se

Osiris - Resultados

han utilizado para su desarrollo y agradecimientos a las personas que han ayudado a la realización del proyecto.

**Osiris**

[← Volver](#)

Créditos de la Aplicación

Plataforma de distribución digital de videojuegos

Equipo de Desarrollo

**Jesús Gómez**

Desarrollador Principal - Cliente

Responsable del desarrollo del cliente en C# con WPF. Se encargó del diseño de la tienda, biblioteca, sistema de registro/login y tokens de autenticación. Especializado en interfaces gráficas y experiencia de usuario.

**Ignacio García**

Desarrollador Backend - API de Juegos

Desarrolló las API de los juegos, la biblioteca, la tienda y el sistema de logs del servidor. En el cliente, trabajó en el sistema de recarga de saldo y lista de amigos del cliente. También trabajó junto a Rubén en la vista web administrativa y el chat del cliente.

**Rubén de las Heras**

Desarrollador Backend - API de Usuarios

Se especializó en la API de usuarios, perfiles, sistema de login y autenticación. Desarrolló el sistema de perfiles de usuario y trabajó junto a Ignacio en el chat en tiempo real con SocketIO y la vista web administrativa.

Sobre el Proyecto

OSIRIS es una plataforma de distribución digital de videojuegos que busca unificar las bibliotecas de juegos de diferentes plataformas en una sola aplicación. Inspirado en el éxito de Steam y Epic Games Store, nuestro objetivo es ofrecer una alternativa intuitiva y liviana que permita a los usuarios gestionar todos sus juegos desde un mismo lugar.

El proyecto incluye una tienda virtual con catálogo dinámico, biblioteca personal de juegos, sistema social con lista de amigos y chat en tiempo real. La aplicación permite tanto comprar nuevos juegos como importar bibliotecas existentes de otras plataformas.

Tecnologías del Servidor

Python

Lenguaje principal del servidor backend

Flask

Framework web para crear la API REST

SQLAlchemy

ORM para gestión de base de datos

PostgreSQL

Base de datos relacional principal

Flask-SocketIO

Chat en tiempo real y comunicación bidireccional

Logging

Sistema de logs y monitorización

Tecnologías del Cliente

C# con WPF

Aplicación de escritorio para Windows

Visual Studio

Entorno de desarrollo integrado

System.Diagnostics

Sistema de logs del cliente

Herramientas de Desarrollo

GitHub

Control de versiones y colaboración

Postman

Pruebas y documentación de API

Visual Studio Code

Editor para desarrollo del servidor

Adobe Photoshop

Diseño del logo y recursos gráficos

draw.io

Creación de diagramas de análisis

StarUML

Diagramas UML y casos de uso

dbDiagram.io

Diseño de la base de datos relacional

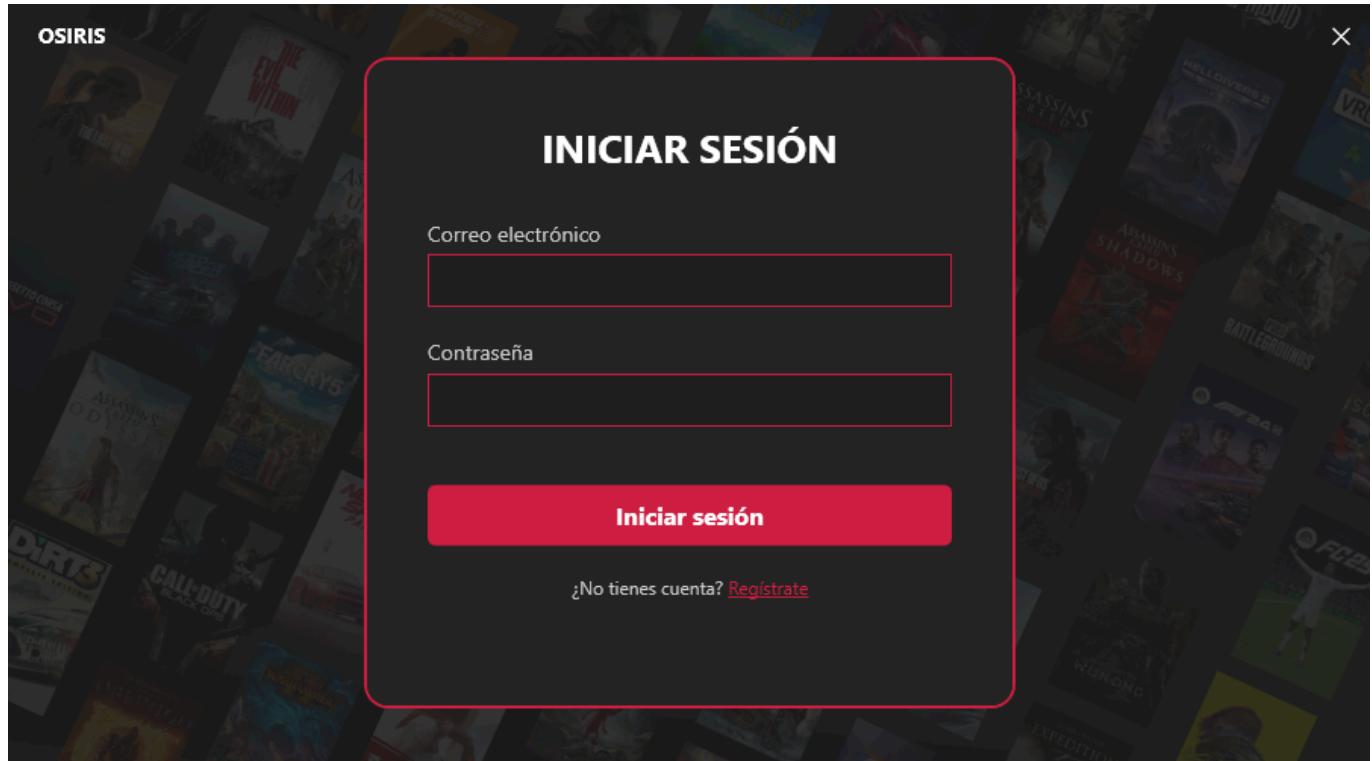
Agradecimientos

Queremos agradecer a todos los profesionales y compañeros que nos apoyaron durante el desarrollo de este proyecto. En especial a Natalia, nuestra tutora.

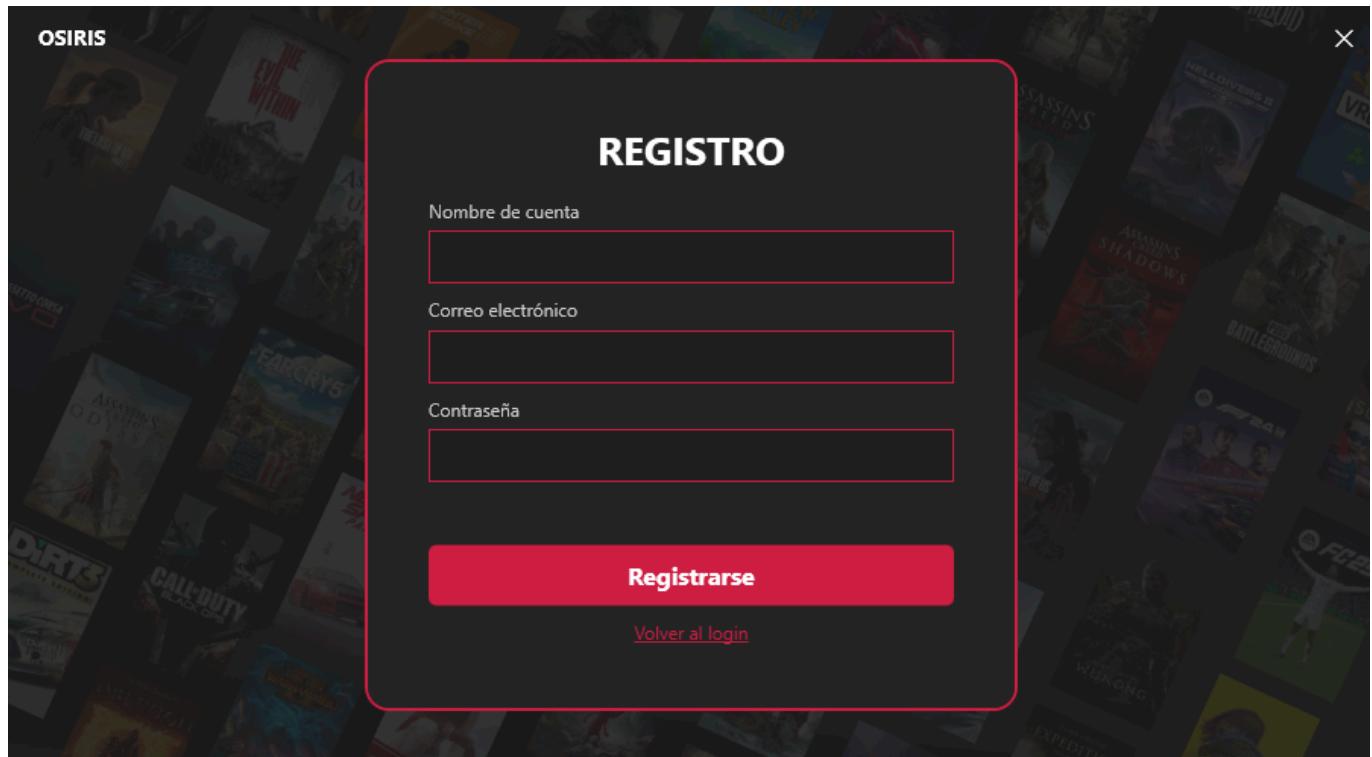
Osiris - Resultados

Cliente

El usuario al abrir la aplicación por primera vez, se le mostrará la pantalla de login. La cual, como es normal, no tendrá usuario y deberá de hacer clic en “Regístrate” la cual mostrará la pantalla que hablaremos a continuación.



Una vez, dado el clic en “Regístrate”. Aparece la ventana de registro de esta manera.

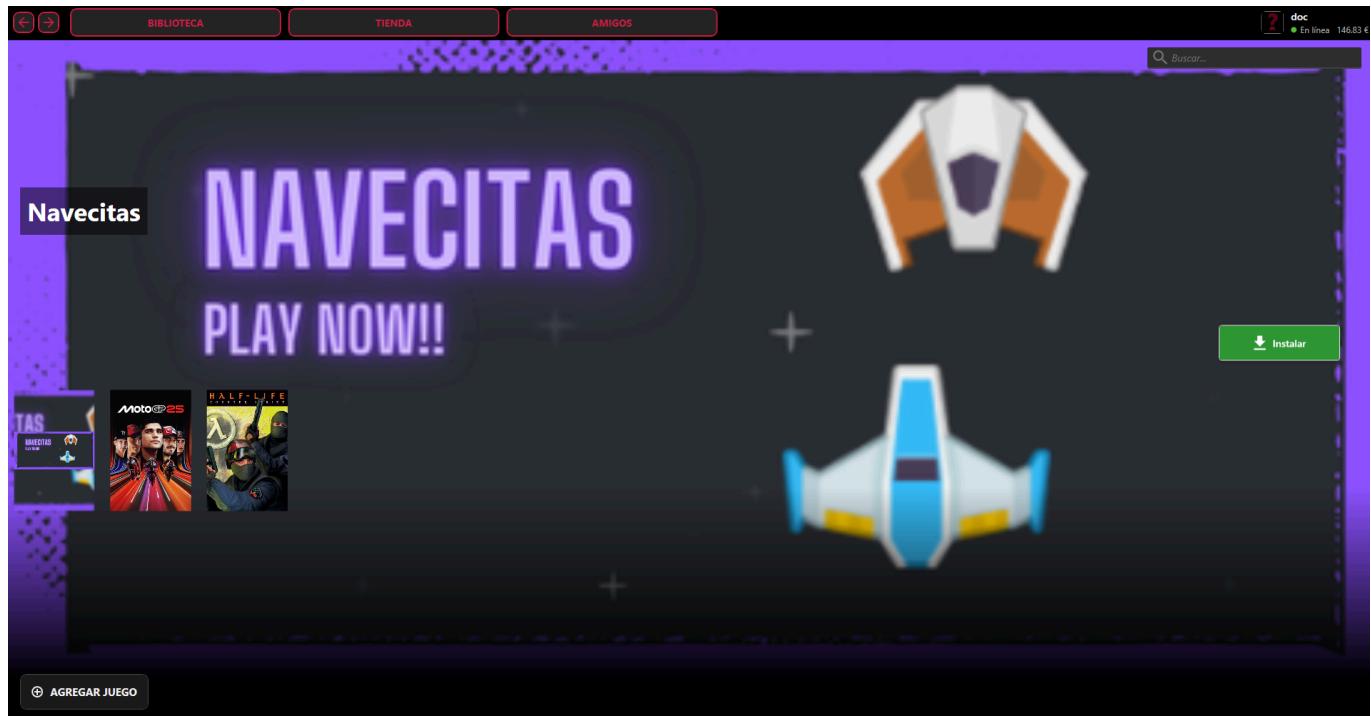


Osiris - Resultados

Una vez, el usuario tiene acceso con su usuario, le aparece esta interfaz, dentro del apartado de la biblioteca.

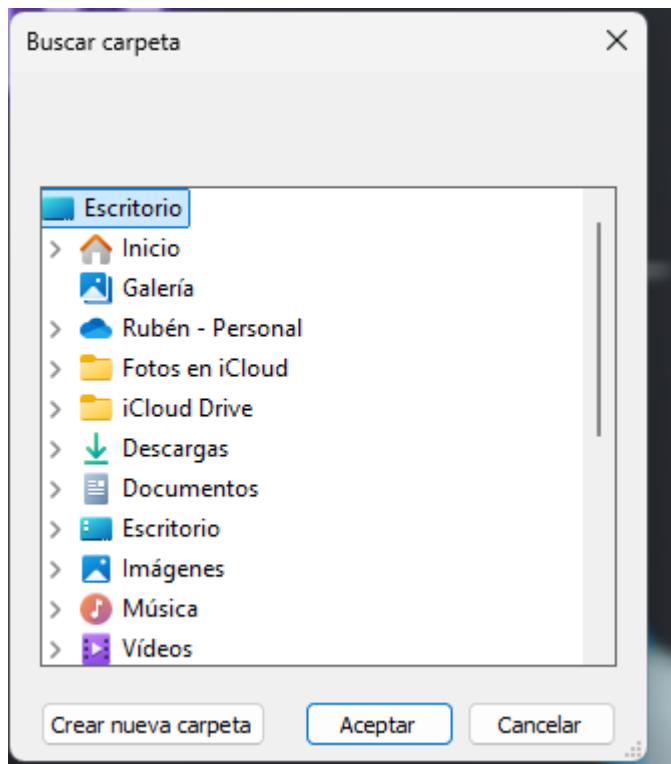


La biblioteca cambia su aspecto visual cuando tenemos juegos, aplicando el banner de fondo del juego al que hagamos clic. Como en este ejemplo:

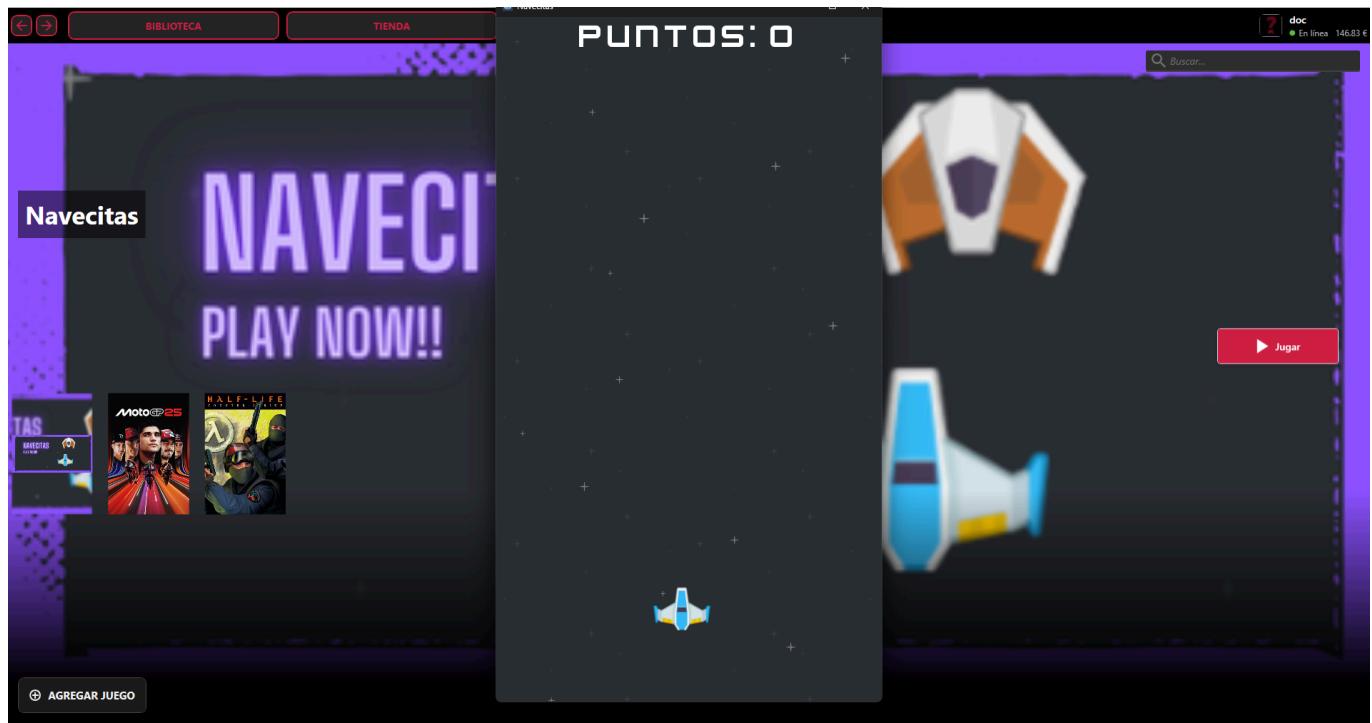


Aquí podemos comprobar que al tener el juego en la biblioteca, le podemos dar a instalar, seleccionar la ruta y obtener nuestro juego

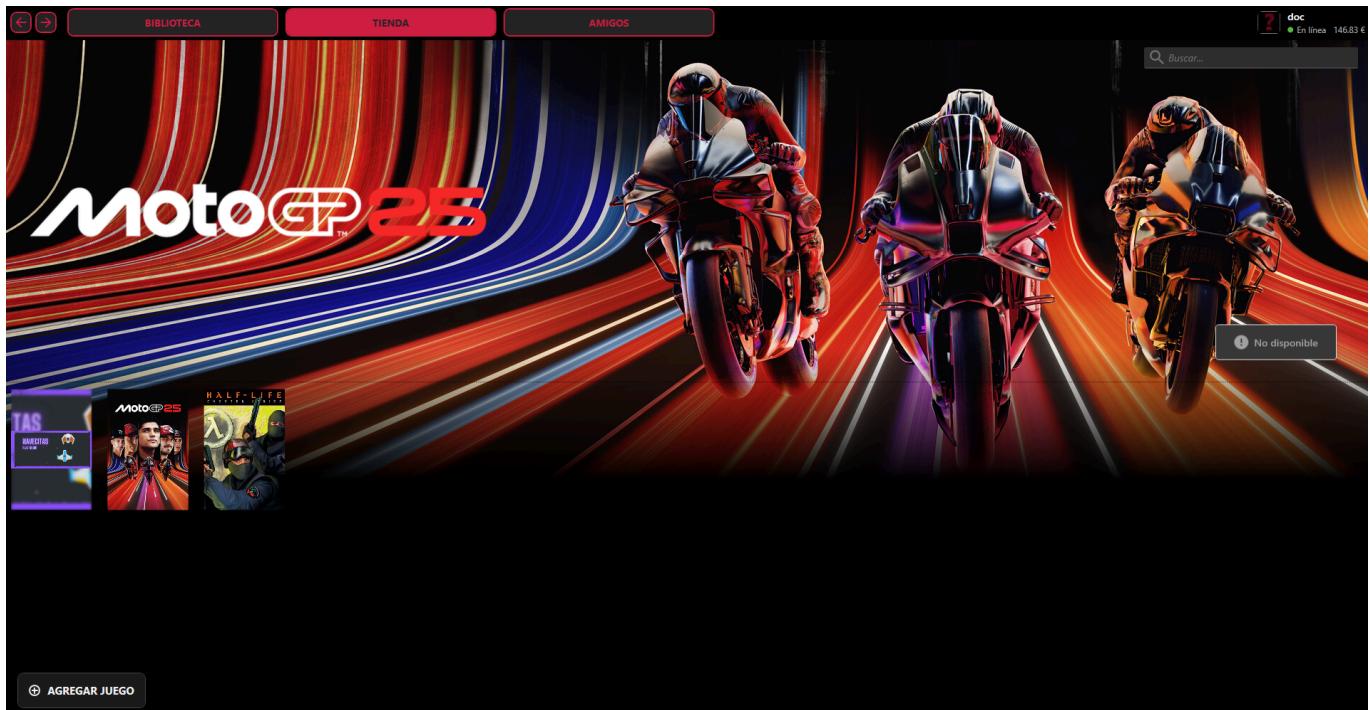
Osiris - Resultados



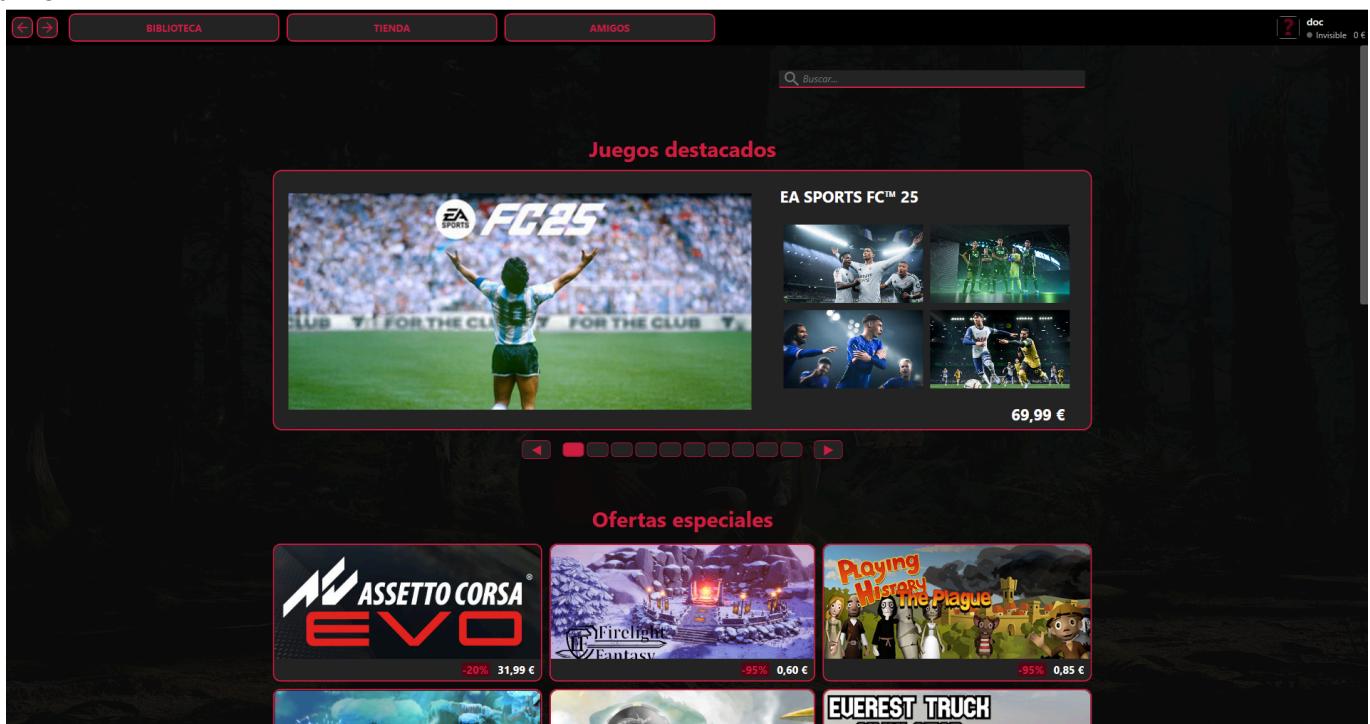
Como podemos comprobar, es perfectamente jugable.



Osiris - Resultados



Cuando el usuario accede al apartado Tienda, le aparece la interfaz la cual va a poder adquirir los juegos que el usuario desee.



Si hace scroll hacia abajo. Le aparecerán diferentes secciones, como por ejemplo (Ofertas especiales, Juegos por menos de 10 Euros o Nuevos Lanzamientos).

Osiris - Resultados

The screenshot shows the main interface of the Osiris game library. At the top, there are three tabs: 'BIBLIOTECA' (Library), 'TIENDA' (Store), and 'AMIGOS' (Friends). In the top right corner, there are icons for help, documentation, and user status.

Juegazos por menos de 10€

- NAVECITAS - PLAY NOW! - 9,99 €
- COUNTER STRIKE - 8,19 €
- THE BINDING OF ISAAC - 4,99 €
- QUAKE - 9,99 €

Nuevos lanzamientos

- F1® 25 - Sports, Simulation, Racing - Fecha de lanzamiento: 30 May, 2025 - 59,99 €
- Lost Soul Aside™ - Action, Adventure, RPG - Fecha de lanzamiento: 30 May, 2025 - 69,99 €
- Onimusha 2: Samurai's Destiny - Action - Fecha de lanzamiento: 22 May, 2025 - 29,99 €
- MOBILE SUIT GUNDAM SEED BATTLE DESTINY REMASTERED - Action, RPG, Casual - Fecha de lanzamiento: 21 May, 2025 - 39,99 €
- DOOM: The Dark Ages - Action - Fecha de lanzamiento: 14 May, 2025 - 79,99 €
- MotoGP™25 - Sports, Simulation, Racing - Fecha de lanzamiento: 30 Apr, 2025 - 59,99 €

El usuario tiene la posibilidad de ingresar dentro de un juego, para realizar la compra, pudiendo visualizar la interfaz, además del botón de compra.

The screenshot shows the game store interface of Osiris. At the top, there are three tabs: 'BIBLIOTECA' (Library), 'TIENDA' (Store), and 'AMIGOS' (Friends). In the top right corner, there are icons for help, documentation, and user status.

DOOM: The Dark Ages

79,99 €

COMPRAR

DOOM: The Dark Ages is the prequel to the critically acclaimed DOOM (2016) and DOOM Eternal that tells an epic cinematic story of the DOOM Slayer's rage. Players will step into the blood-stained boots of the DOOM Slayer, in this never-before-seen dark and sinister medieval war against Hell.

Fecha de lanzamiento: 14 May, 2025

Desarrollador: id Software

Editor: Bethesda Softworks

PREORDER

PRE-ORDER BONUS

UPGRADE TO

Osiris - Resultados

Si el usuario desea obtener información detallada acerca del juego, deberá de hacer scroll hacia abajo para mirar los detalles del juego.

The screenshot shows a game store interface with a dark background. At the top, there are three tabs: 'BIBLIOTECA' (Library), 'TIENDA' (Store), and 'AMIGOS' (Friends). In the top right corner, there is a user icon with the name 'doc' and the status 'Invisible 0 €'. A large, semi-transparent overlay window is centered over the store interface. This overlay has a dark background with red highlights. It features two main sections: 'PREORDER' at the top and 'PREMIUM EDITION' below it. The 'PREORDER' section contains an image of a character in a 'VOID SLAYER SKIN' and a box stating 'UPGRADE TO PREMIUM FOR UP TO 2-DAYS ADVANCED ACCESS* AND MORE'. Below this, a small note says 'Pre-order now to receive the Void DOOM Slayer Skin at launch.' The 'PREMIUM EDITION' section contains an image of the game cover and a box stating 'UP TO 2-DAYS ADVANCED ACCESS*'. The overall design is modern and professional, typical of a digital game store.

This screenshot shows the same game store interface as the previous one, but the overlay window is now focused on the 'ABOUT THE GAME' section of the Premium Edition. The text in this section reads: 'Get a head start on slaying demons with the Premium Edition, including up to 2-Day Advanced Access* and the campaign DLC**. You'll also receive the Digital Artbook and Soundtrack, and the Divinity Skin Pack, which includes matching skins for the DOOM Slayer, dragon and Atlan.' Below this, a list titled 'INCLUDES:' lists the contents of the Premium Edition: 'Base Game', 'Up to 2-Day Early Access', 'Campaign DLC', 'Digital Artbook and Soundtrack', and 'Divinity Skin Pack'. A note states: '*Actual play time depends on purchase date and applicable time zone differences, subject to possible outages. **DLC availability to be provided at a later date.' At the bottom of this section is the game's cover art for 'DOOM: THE DARK AGES', featuring a soldier in combat with demons. The overall layout is clean and organized, providing all the necessary information for potential buyers.

Osiris - Resultados

REIGN IN HELL

As the super weapon of gods and kings, shred enemies with devastating favorites like the Super Shotgun while also wielding a variety of new bone-chewing weapons, including the versatile Shield Saw. Players will stand and fight on the demon-infested battlefields in the vicious, grounded combat the original DOOM is famous for.

STAND AND FIGHT

Experience an epic story of the DOOM Slayer's rage in this cinematic and action-packed story. Bound to serve as the super weapon of gods and kings, the DOOM Slayer fends off demon hordes as their leader seeks to destroy the Slayer and become the only one that is feared. Witness the creation of a legend as the Slayer takes on all of Hell and turns the tide of war.

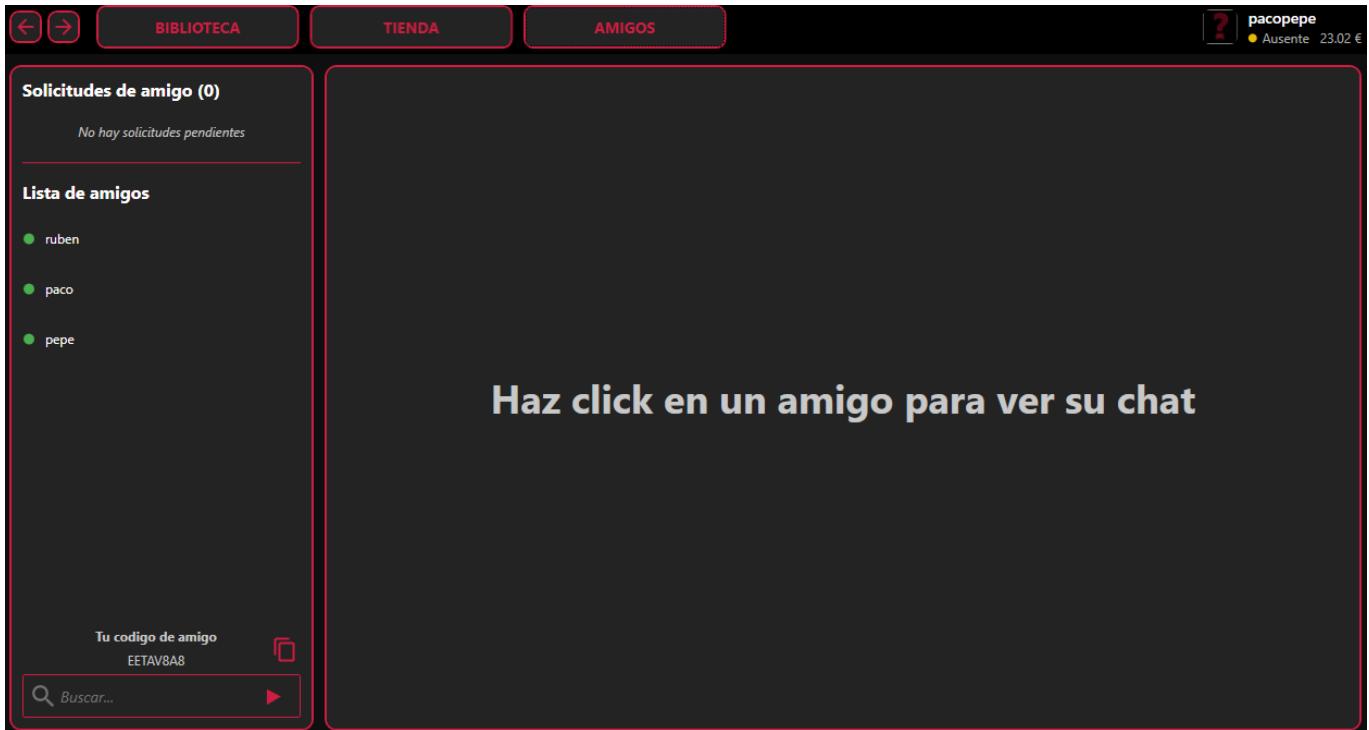
DISCOVER UNKNOWN REALMS

In his quest to crush the legions of Hell, the Slayer must take the fight to never-before-seen realms. Mystery, challenges, and rewards lurk in every shadow of ruined castles, epic battlefields, dark forests, ancient hellscapes, and worlds beyond. Armed with the viciously powerful Shield Saw, cut through a dark world of menace and secrets in id's largest and most expansive levels to date.

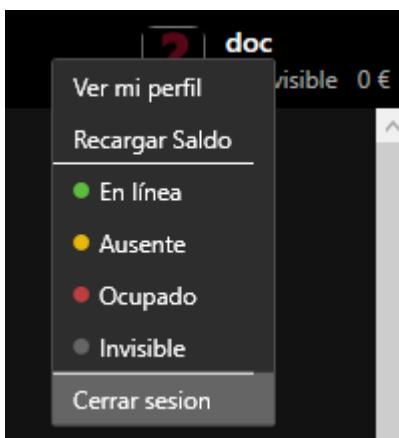
Categorías: Single-player Full controller support Family Sharing	Géneros: Action
--	---------------------------

Osiris - Resultados

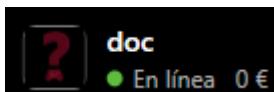
Esta pantalla es la de amigos, la cual permite agregar amigos, además de chatear.



Cuando haces clic en el perfil de usuario, aparecen todas estas opciones que detallaremos a continuación:



Se puede cambiar el estado del usuario. Indicando su disponibilidad a la hora de jugar.



Osiris - Resultados

Si nos dirigimos a “Recargar Saldo”, aparece la interfaz para ingresar dinero en la cuenta para la posterior compra de juegos o productos.

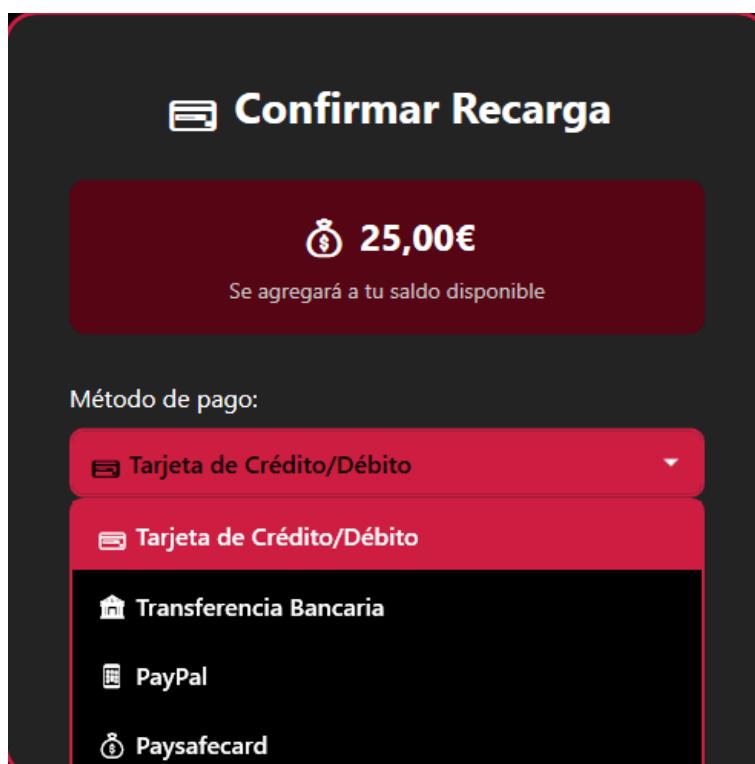


Al hacer scroll hacia abajo, podemos agregar cantidad personalizada.



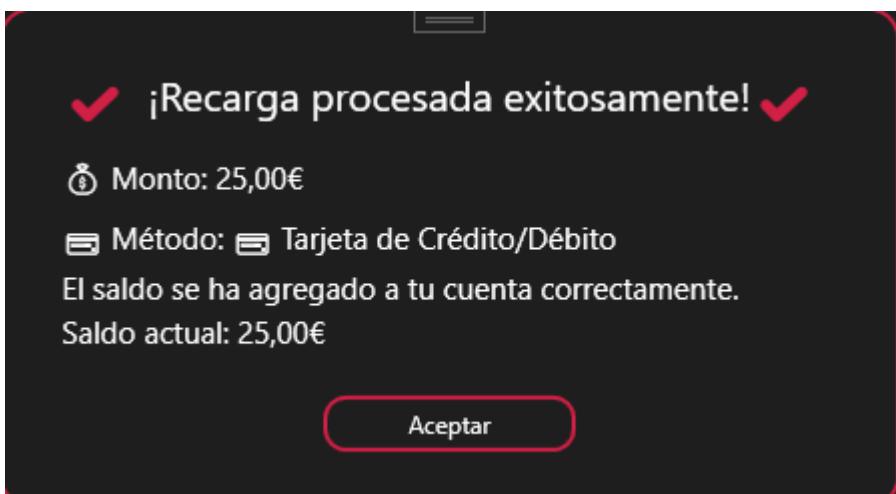
Osiris - Resultados

Una vez le demos a “*continuar con recarga*”, nos aparecerá un cartel para confirmar, pudiendo cambiar los métodos de pago:

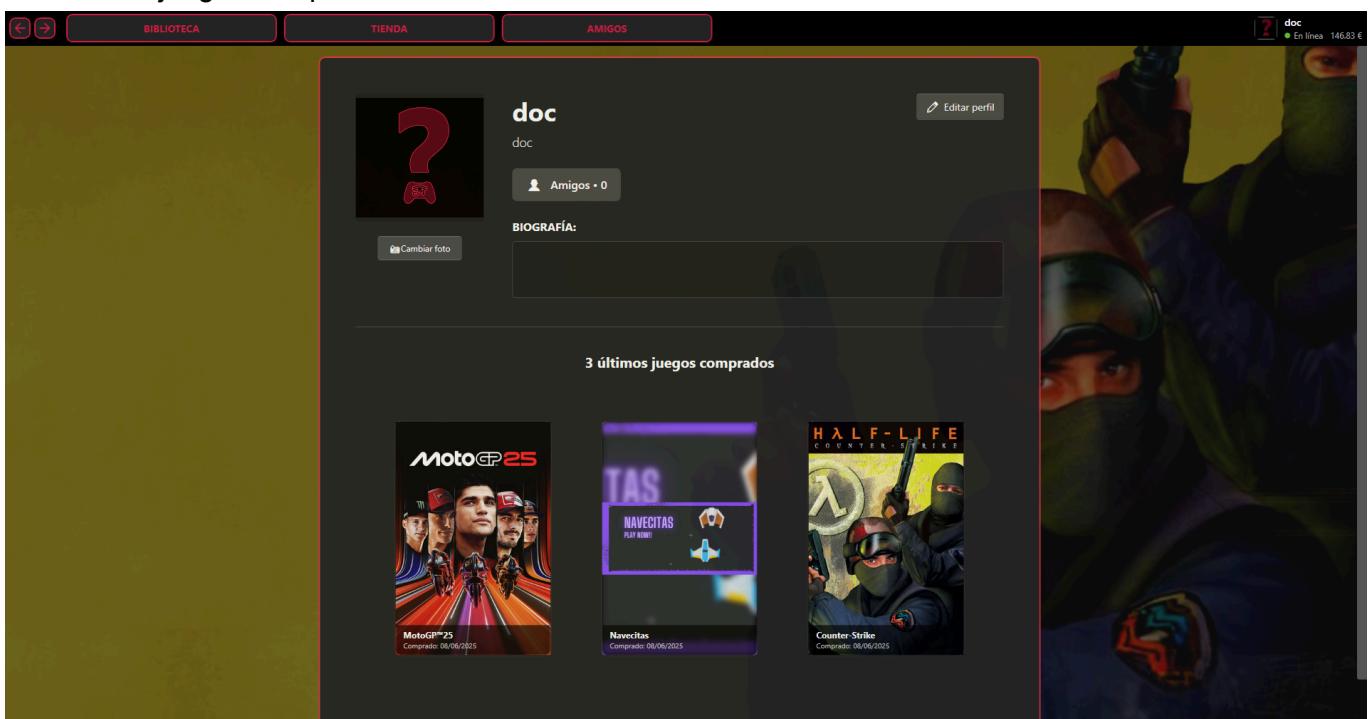


Osiris - Resultados

Una vez pulsado en *Confirmar*, nos aparece una ventana indicando que la recarga se ha procesado correctamente.

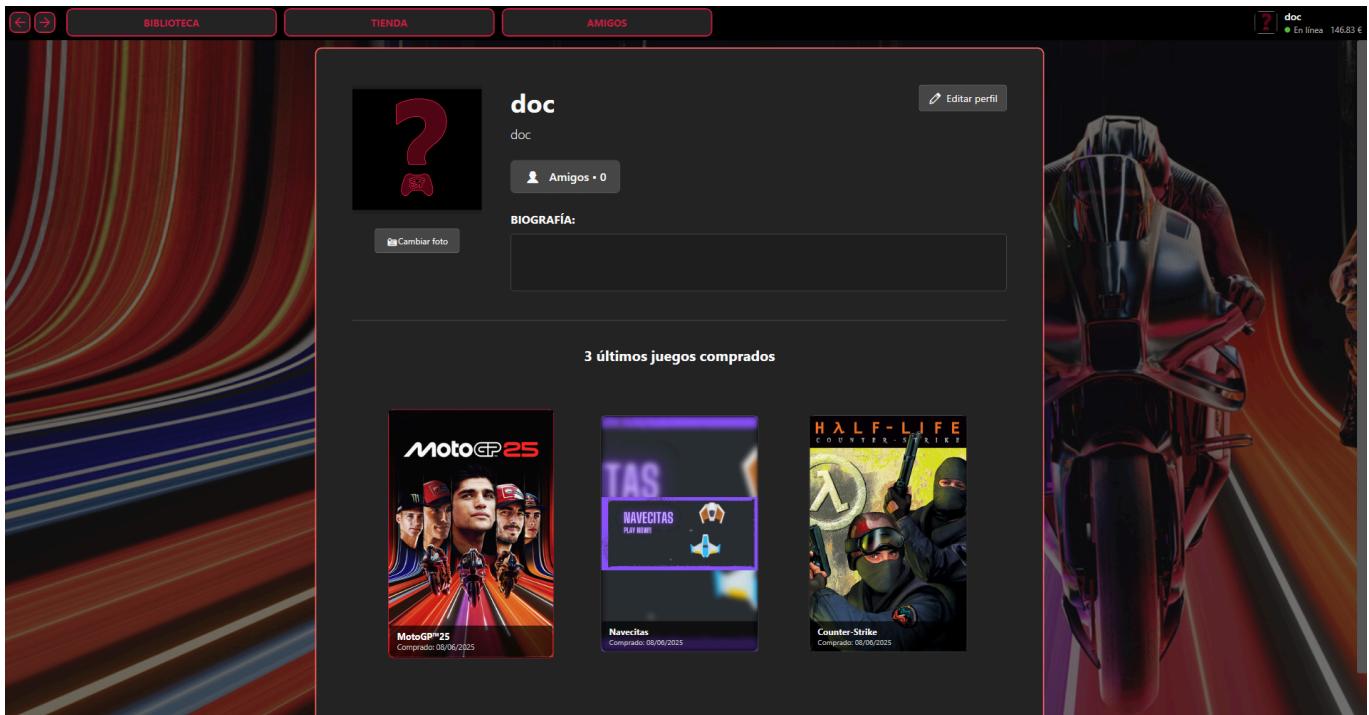


Al dirigirnos a nuestro perfil de usuario, podemos comprobar que tenemos todos nuestros datos y los últimos tres juegos comprados.



Al pasar el mouse por encima de las carátulas de los juegos, nos cambiará el fondo del perfil.

Osiris - Resultados



Discusión

En este proyecto se han podido cumplir con los objetivos principales, aunque también se pensaron funcionalidades que no llegaron a poder implementarse por falta de tiempo, en esta sección haremos un repaso a lo que se pudo implementar en el proyecto y aquellas funcionalidades que no llegaron a ver la luz.

Analizaremos estas funcionalidades según los objetivos principales:

1. *Implementar una tienda virtual con un catálogo dinámico y simulación de compra.*

En este apartado, se pudo cumplir con el objetivo inicial, pero teníamos varias ideas que nos fueron surgiendo que por limitaciones de tiempo nunca llegaron a la aplicación, como es el caso de una página de búsqueda con una gran cantidad de filtros aplicables sobre los juegos, estos filtros, si que se llegaron a hacer en el servidor, el endpoint `/games`, admitía más de 10 parámetros que filtraban la información y la entregaba paginada, fue una gran lastima no poder implementarla teniendo ya gran parte de ese sistema funcionando, adicionalmente también se planteó implementar que la búsqueda de juegos utilizara “elastic search”, un método de búsqueda que admite errores en la escritura, pero tampoco se pudo implementar. Por otro lado, el sistema de compra de juegos lo tuvimos que simplificar mucho respecto a la idea que teníamos, nuestra idea, era simular el pago usando Stripe, una pasarela de pago que nos permite poner tarjetas de crédito de prueba para simular el proceso real de principio a fin, esto

Osiris - Discusión

hubiese aportado seguridad y confianza al usuario que prueba la app. Lamentablemente no pudo ser implementado debido a la falta de tiempo.

2. *Desarrollar una biblioteca que permita ejecutar, agregar e importar juegos desde otras plataformas.*

Este apartado se pudo cumplir con creces, la biblioteca permite ejecutar juegos de la tienda que estén disponibles para ejecutar, juegos que agregue el usuario, cambiar el ejecutable de los juegos no disponibles etc. Algo que se tenía pensado poder añadir, era una forma de importar directamente tus juegos desde otras plataformas, para automatizar el proceso de que el usuario tenga que agregar sus ejecutables de sus juegos uno a uno. Lamentablemente no pudo ser implementado debido a la falta de tiempo.

3. *Crear un sistema de usuarios con perfiles interactivos, listas de amigos y un chat integrado.*

En este sistema también llegamos a cumplir con los objetivos iniciales planteados, sin embargo, este sistema es el que se quedó más incompleto, se cumplió con los objetivos iniciales pero faltaron sistemas muy importantes por implementar como puede ser ver el perfil de los usuarios agregados o que te llegara una notificación cada vez que te llegara un mensaje de un usuario, este último fue el sistema con mayor prioridad pero no se pudo realizar debido a que el chat en tiempo real usando sockets, se logró completar un día antes de la entrega del proyecto debido a su complejidad y el tiempo que nos llevó implementarlo, fue una gran lástima ya que es un sistema crucial en una app con chat en tiempo real.

El resumen general es que se pudo cumplir con los objetivos iniciales pero el tiempo no fue suficiente para implementar todos los sistemas que fueron planteados.

Conclusión

El proyecto ha logrado cumplir con los objetivos principales establecidos, desarrollando una aplicación de gestión de videojuegos que integra una tienda virtual, una biblioteca de juegos funcional y un sistema social con perfiles de usuario, listas de amigos y chat en tiempo real. La implementación de una arquitectura RESTful con Flask, el uso de PostgreSQL y la interfaz de usuario en C# con WPF han permitido crear una solución modular y fácilmente escalable.

A pesar de los desafíos enfrentados, como la migración de MongoDB a PostgreSQL, la curva de aprendizaje de nuevas tecnologías y las limitaciones de tiempo que impidieron incorporar funcionalidades adicionales, como la importación automática de juegos de otras plataformas o un sistema de notificaciones para el chat, los resultados obtenidos demuestran un sistema funcional que es capaz de competir con plataformas existentes. Este desarrollo no solo centraliza bibliotecas de juegos de distintas plataformas, sino que también ofrece una base sólida para futuras mejoras, dándonos la oportunidad de expandir las capacidades del proyecto dedicando más tiempo al desarrollo, podría ser una alternativa atractiva para los usuarios respecto a la competencia.

Anexo

Hoja de pruebas

En esta hoja de excel hemos realizado validación funcional y técnica que recoge un conjunto de pruebas realizadas sobre la aplicación, con el fin de verificar que se cumplen los requisitos especificados.

La hoja de excel está adjuntada junto con los archivos del proyecto.

PRUEBA	TIPO DE PRUEBA	ID_PRUEBA	REQUISITO ASOCIADO	REQUISITOS PREVIOS	RESULTADOS ESPERADOS	RESULTADOS OBTENIDOS
Registro de usuario	Seguridad	PNJR-004	(REQ-014)	Comprobar que el token corresponde al usuario	Al registrar, cerrar la ventana y volver a abrirla, ahí debe de abrirse la cuenta sin logearse. Después, cerrar sesión, y abrir la aplicación otra vez. No debe permitir acceder sin logearse.	Funciona correctamente el token. Permite acceder una vez cerrada la aplicación si no se ha cerrado sesión. Solo pide logearse pulsas en el botón "Cerrar Sesión".
	Usabilidad	PNJR-005	(REQ-016)	Asegurarse de que la interfaz es intuitiva y fácil para el usuario.	Los campos deben estar fácilmente visibles y claramente identificados donde tiene que ir cada dato.	Los campos están claramente visibles e indican con bastante obviedad donde debe de ir cada dato.
	Usabilidad	PNJR-006	(REQ-013)	Verificar que los errores sean claros y útiles	Al ingresar un dato incorrecto, debe indicar al usuario claramente que los datos ingresados son incorrectos. Además, si no se ingresa ningún dato, debe de aparecer un mensaje indicando que se debe de llenar todos los campos.	Cuándo el formato de registro no es el correcto se indica claramente resaltando la celda en rojo y se indica debajo qué formato debe de seguir. Además, si ningún dato ha sido llenado, se indica al usuario que se deben de llenar todas las casillas.
	Seguridad	PNJR-007	(REQ-014)	Asegurarse de que la contraseña se encripta correctamente en la base de datos.	Al registrar el usuario, comprobar en la base de datos que la contraseña es hasheada.	La contraseña se encripta correctamente. Ocultando la verdadera clave del usuario.

Bibliografía

Codenautas. (2022b, julio 5). *Aplicación CRUD con Python, Flask y MongoDB*

[Vídeo]. YouTube. https://www.youtube.com/watch?v=A05cBJ_P7Q8

Donweb Cloud. (2025, 11 abril). *CURL: La navaja suiza que necesitas si eres programador* [Vídeo].

YouTube. <https://www.youtube.com/watch?v=n3NtrQYrjDw>

Kampa Plays. (2022, 3 noviembre). *C# WPF Tutorial #1 - What is WPF?*

Osiris - Anexo

[Vídeo]. YouTube. https://www.youtube.com/watch?v=t9ivUosw_il

hdeleon.net. (2020, 9 diciembre). *¿Qué tipos de proyectos se pueden hacer con C#?*

[Vídeo]. YouTube. https://www.youtube.com/watch?v=8e_u2Ch-GMc

hdeleon.net. (2022, 28 junio). *5 FRAMEWORKS para PROGRAMAR BACKEND*

[Vídeo]. YouTube. <https://www.youtube.com/watch?v=bDRFiomUfdc>

Danisable. (2020, 23 abril). *Curso C# gráfico | Windows Forms | Cursor | Parte #21*

[Vídeo]. YouTube. <https://www.youtube.com/watch?v=LyYSNam-1YU>

Informática y Programación. (2022, 9 abril). *6 - Crear librería (DLL) y añadirla a proyecto con C# y .NET*

[Vídeo]. YouTube. https://www.youtube.com/watch?v=JAU8Cc7jX_Y

Javier Pinilla. (2023, 17 agosto). *Crea un API con Python en SOLO 10 MINUTOS | Tutorial Flask API*

[Vídeo]. YouTube. <https://www.youtube.com/watch?v=b0ZrmhyvCY4>

MICHEL DAVALOS BOITES. (2020, 28 octubre). *RESTful API con Flask* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=rzsiyZe4I4Y>

UskoKruM2010. (2024, 8 agosto). *Qué es una API en Programación y cómo funciona | La mejor explicación en español, para principiantes* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=rMPLHPnltmM>

UskoKruM2010. (2024b, agosto 15). *Qué es una REST API y cómo funciona | La mejor explicación en español, para principiantes ✓* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=LQdJG8Qc5xo>

Osiris - Anexo

pildorasinformaticas. Curso Python desde 0. (2017, 24 julio). YouTube.

<https://youtube.com/playlist?list=PLU8oAIHdN5BlvPxziopYZRd55pdqFwkeS&si=KFRZEmx9QW3W-IAW>

logging — Logging facility for Python. (s. f.). Python Documentation.

<https://docs.python.org/3/library/logging.html>

Flask — Flask Documentation (3.1.x). (s. f.).

<https://flask.palletsprojects.com/en/stable/#user-s-guide>

SQLAlchemy Documentation — SQLAlchemy 2.0 documentation. (s. f.).

<https://docs.sqlalchemy.org/en/20/>

Flask-SocketIO — Flask-SocketIO documentation. (s. f.).

<https://flask-socketio.readthedocs.io/en/latest/>