Gameslan

Docente
Uzal Alan

Integrantes del proyecto
Guagliardo Dario
Perez Marcos
Gonzalez Daniel

Índice:

Resumen

El objetivo principal de este proyecto es el desarrollo de una aplicación gráfica en Python, inspirada en Steam. El sistema permite a los usuarios registrarse, iniciar sesión, comprar juegos y gestionar su biblioteca personal. La aplicación implementa concurrencia mediante hilos, estructura en capas (interfaz, lógica, acceso a datos y base de datos) y utiliza programación orientada a objetos. Además, se garantiza la persistencia de datos en una base de datos local.

I. Lista de requisitos

Requerimiento funcionales	
1- Al comprar un juego se vincule ese juego al usuario.	✓
2- Que se pueda crear un usuario y se guarde en la base de datos.	✓
3- El usuario pueda descargar juegos	✓
4- El usuario puede cerrar sesión.	✓
5- Al terminar la descarga cambie el estado del juego.	✓
6- El usuario pueda desinstalar sus juegos.	✓
7- No se pueda descargar 2 juegos a la vez	✓

II. Introducción y marco teórico

En este proyecto se abordó la creación de un sistema aplicando conocimientos clave como la concurrencia, la programación orientada a objetos, la arquitectura en capas y la persistencia de datos, integrando todos estos conceptos en una solución concreta desarrollada en Python.

Durante su desarrollo se aplicaron diversos conceptos clave de la programación moderna. Por un lado, se implementó concurrencia mediante hilos, permitiendo ejecutar tareas simultáneamente, como simular la descarga de un juego mientras se navega por la aplicación. Para evitar conflictos entre hilos, se incorporaron mecanismos de exclusión mutua, como

“Lock", garantizando que ciertas operaciones críticas no se ejecuten en paralelo.

El sistema está estructurado según el modelo de arquitectura en capas, dividiendo el código en presentación, lógica de negocio y acceso a datos, lo que mejora la organización y facilita su mantenimiento.

La base de datos local, implementada con “sqlite3", permite almacenar información persistente como usuarios

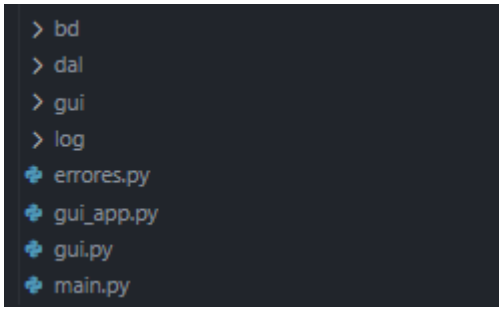
Todo el desarrollo se gestionó de forma colaborativa mediante un repositorio GIT y un google Drive, lo que permitió mantener un historial ordenado de avances y facilitar el trabajo en equipo.

III. Ejecución del proyecto

El desarrollo del proyecto se llevó a cabo de manera progresiva y colaborativa, distribuyendo las tareas según las fortalezas y roles de cada integrante del equipo. Durante las clases, mientras algunos miembros se enfocaban en la elaboración de la documentación técnica y en la creación de diagramas y del diseño (Guagliardo Darío), otros avanzaban en la implementación del código y en la integración de los distintos módulos del sistema (Perez Marcos y Gonzalez Daniel).

Esta división permitió un avance paralelo y ordenado, garantizando que tanto la planificación como la construcción práctica del software se desarrollarán de forma simultánea. Al trabajarse todo en el mismo entorno se facilitó la comunicación, ayudando a la resolución de dudas, la coordinación entre las capas (GUI, lógica de negocio, acceso a datos y base de datos) y la integración efectiva de los componentes concurrentes, como los hilos para la gestión de descargas.

IV. Modelo en capas.

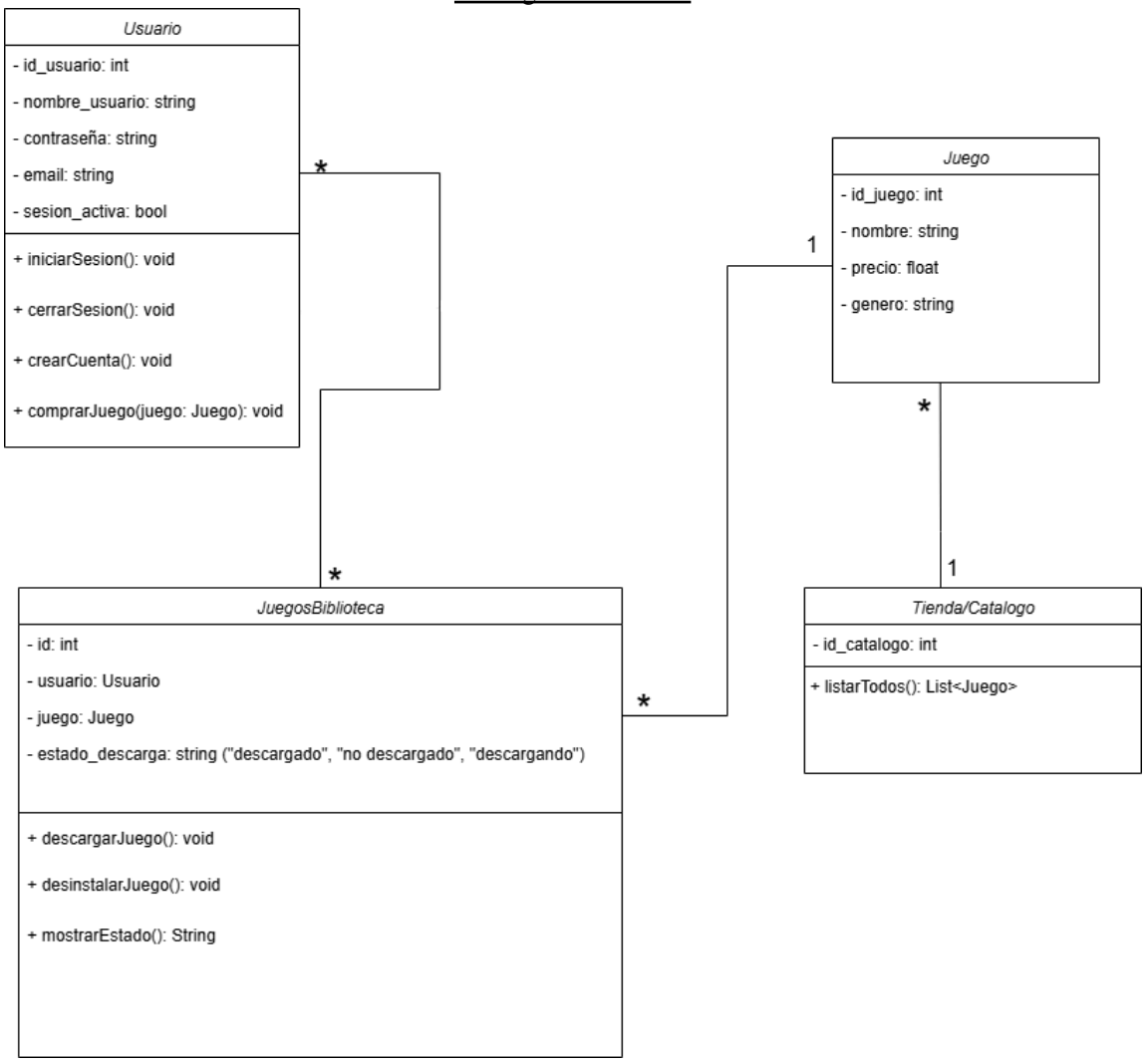


Este proyecto está organizado por capas, lo que mejora su mantenimiento y escalabilidad. La carpeta “bd**”** contiene todo lo relacionado con la base de datos, como conexiones y sentencias SQL. La carpeta “dal" (Data Access Layer) se encarga del acceso a datos, funcionando como intermediaria entre la base de datos y la lógica del programa. La carpeta “gui**”** guarda componentes gráficos reutilizables o pantallas específicas de la interfaz. “log**”** maneja el registro de errores y eventos del sistema.

Entre los archivos sueltos, “errores.py” define y gestiona excepciones o mensajes de error personalizados. “gui_ app.py" es el archivo principal de la interfaz gráfica, donde se inicializa y ejecuta la aplicación visual.

Debido a la problemática encontrada al implementar la concurrencia junto con la barra de descarga, la idea general del proyecto cambió, se pasó de desarrollar una aplicación de consola a una aplicación con interfaz gráfica. Los siguientes archivos representan cómo era la estructura del proyecto antes de realizar ese cambio. El archivo “gui.py” se utilizaba para manejar la interfaz desde la consola, cumpliendo un rol similar al que luego tendría “gui_app.py” en la versión gráfica, aunque sin elementos visuales complejos. Este archivo se conectaba directamente con “main.py”, que actuaba como punto de entrada del sistema y desde donde se ponía en marcha toda la aplicación, conectando las distintas capas y componentes del programa.

V. Diagrama de clases



Explicación del diagrama:

El sistema está centrado en los usuarios, quienes pueden crear cuentas, iniciar sesión, cerrar sesión y comprar juegos. Cada usuario tiene datos personales como nombre de usuario, email y contraseña, además de un indicador para saber si tiene la sesión activa.

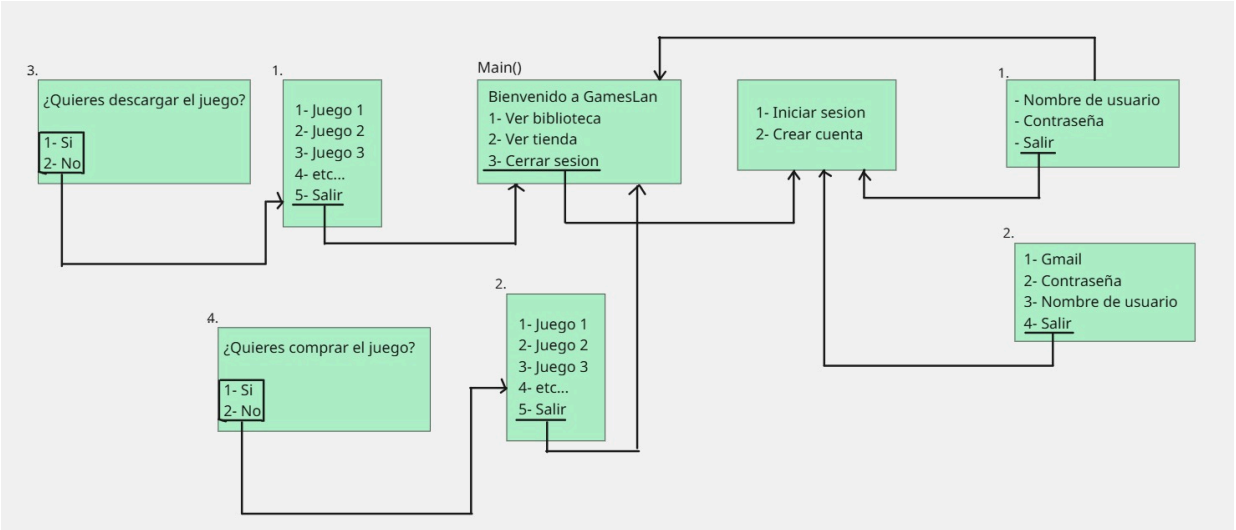
Los juegos están representados por una clase que incluye información básica como el nombre, género, id y precio. Cada juego también tiene un método para mostrar sus detalles.

Los juegos están agrupados dentro de una tienda/ catálogo, que es donde los usuarios pueden buscar juegos por nombre o ver una lista completa de todos los juegos disponibles. El catálogo tiene una relación de uno a muchos con los juegos, ya que puede contener varios títulos.

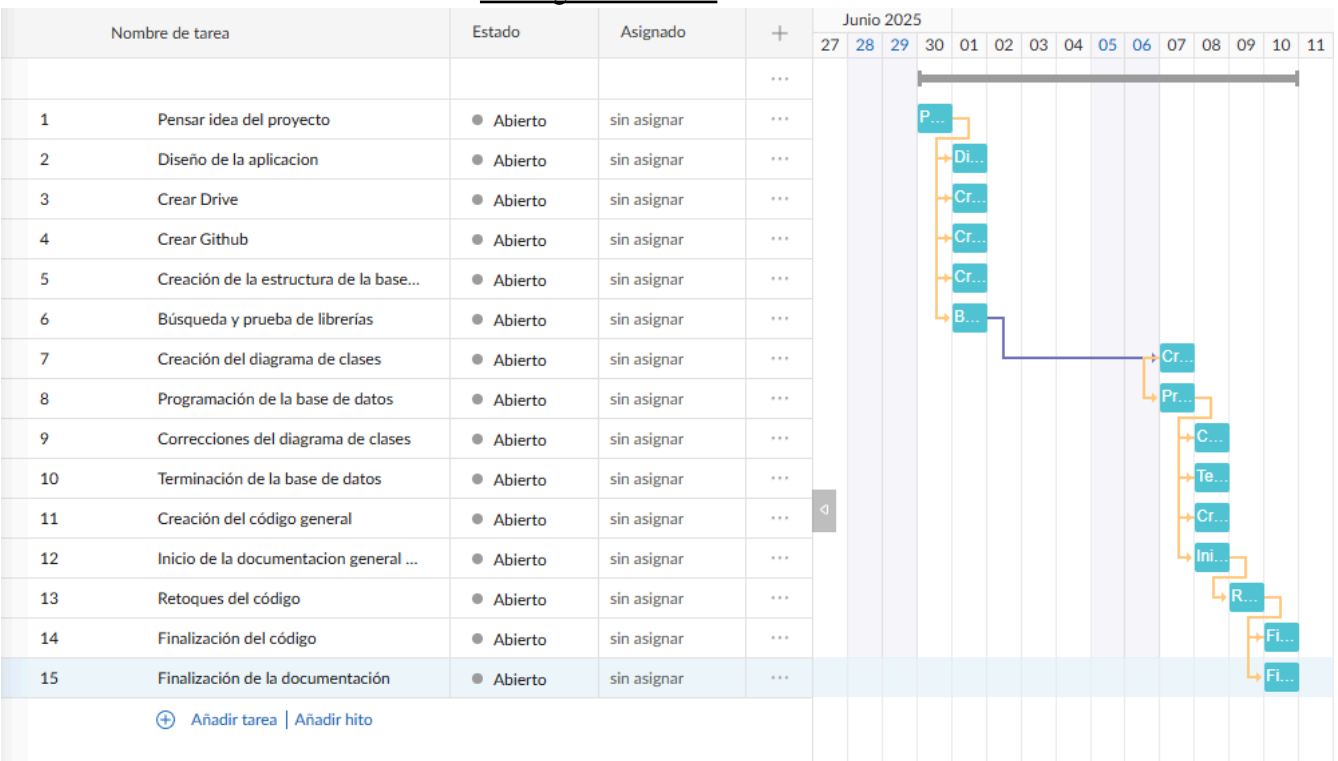
Una vez que un usuario compra un juego, este se agrega a su biblioteca personal, representada por la clase JuegosBiblioteca. Esta clase funciona como un registro que relaciona a un usuario con los juegos que posee, incluyendo el estado de descarga de cada uno. Por ejemplo, un juego puede estar en proceso de descarga, ya descargado o aún no descargado. Además, la biblioteca cuenta con métodos que permiten gestionar cada juego: descargarlos, desinstalarlos y consultar su estado. De esta manera, desde esta clase el usuario puede controlar el acceso técnico a los juegos que compró.

Las relaciones entre clases reflejan el funcionamiento de este tipo de plataformas: un usuario puede tener muchos juegos en su biblioteca, y un juego puede estar en la biblioteca de muchos usuarios distintos. Lo mismo ocurre con el catálogo: puede contener muchos juegos distintos, y cada juego pertenece a ese catálogo.

Boceto del proyecto



VI.Diagrama de Gantt



VII. Seguimiento y control

Uno de los principales problemas fue el poder completar todas las pautas requeridas para la entrega final dentro del tiempo disponible en las horas de clase. Esto reflejándose en el trabajo extracurricular en el hogar de cada integrante.

Otro de los problemas que se encontraron fue que, al momento de implementar la barra de descarga, esta se superpone en la consola cada vez que se quería realizar otra acción. Debido a esta limitación, se decidió a último momento incorporar una interfaz gráfica y transformar la aplicación de consola en una aplicación con entorno gráfico. De esta manera, el problema de superposición se resolvió, ya que en una interfaz gráfica cada elemento puede mostrarse de forma ordenada y sin interferencias.

VIII. Conclusiones

Conclusiones sacadas del proyecto:

- Planificación adecuada: Fue importante realizar una planificación detallada antes de comenzar el proyecto. Esto incluye definir los objetivos a resolver, el tiempo invertido, los plazos y las etapas del proyecto.

- Documentación adecuada: A pesar de que el proyecto se desarrolló en un período corto de dos semanas a tres semanas, se mantuvo un seguimiento ordenado de cada etapa, registrando las decisiones tomadas, los cambios realizados y cómo se resolvieron los problemas que surgieron. Esto ayudó a mantener una buena organización y comunicación dentro del equipo.

IX.Biblioteca usada

Tkinter es la librería estándar de Python para crear interfaces gráficas de usuario (GUIs). Funciona sobre Tcl/Tk, un toolkit gráfico muy usado, y viene incorporado con la mayoría de las instalaciones de Python. Es compatible con Linux, macOS y Windows, lo que la hace muy accesible para desarrollar aplicaciones visuales multiplataforma.

¿Qué permite hacer Tkinter?
Tkinter permite crear ventanas, botones, etiquetas, campos de texto, barras de progreso, diálogos, entre muchos otros elementos gráficos.
También incluye una versión más moderna llamada ttk (Themed Tk), que ofrece componentes visuales con mejor apariencia en distintos sistemas operativos (como botones más estilizados o pestañas modernas).

¿Cómo funciona internamente?
Cuando usás una función de Tkinter, Python traduce eso a comandos Tcl/Tk. Estos comandos son ejecutados por un intérprete Tcl interno, que se comunica con el sistema operativo para mostrar la ventana o el componente correspondiente. En segundo plano, Tkinter usa:

- **Xlib** en Linux,
- **Cocoa** en macOS,
- **GDI** en Windows.

Ejemplo:

```python
from tkinter import *
from tkinter import ttk

root = Tk()  # Crea la ventana principal
# Acá se agregan widgets como botones, etiquetas, etc.
root.mainloop()  # Inicia el bucle principal de la interfaz
```

¿Cómo se aplicó Tkinter en este proyecto?
En este proyecto, Tkinter se utilizó para construir una interfaz gráfica completa, reemplazando totalmente el uso de la consola. Toda la interacción con el usuario se realiza a través de esta interfaz.
Dentro de esta interfaz, se implementó una barra de progreso que simula el proceso de descarga de un juego, permitiendo al usuario visualizar en tiempo real el avance de dicha descarga.
Además, el uso de Tkinter combinado con un sistema de control de tareas concurrentes permite que la barra se actualice mientras la descarga se ejecuta en segundo plano, sin que la interfaz se congele o deje de responder. Esto demuestra el aprovechamiento de la concurrencia dentro de un entorno gráfico, permitiendo que distintas tareas se gestionen de forma fluida y simultánea.
Biblioteca Tkinter

X. Links:

Drive

Github