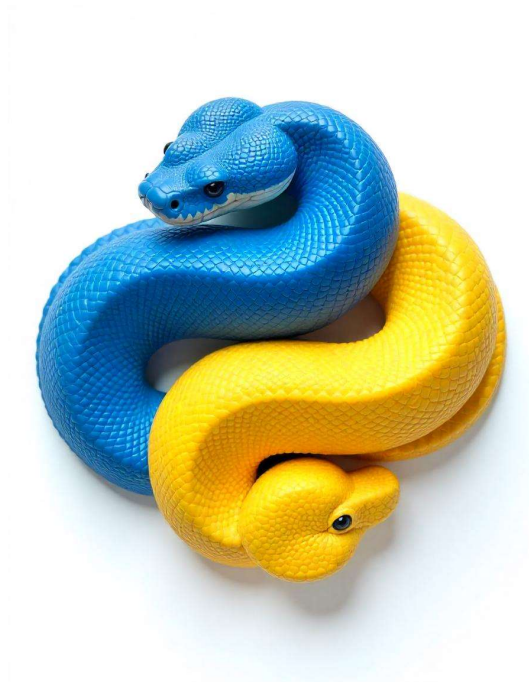




DESARROLLO DE INTERFACES

Tema 04 – Boletín 01



**I.E.S CARRILLO SALCEDO
2º DESARROLLO DE APLICACIONES
MULTIPLATAFORMA
CURSO 2024/2025**

**REALIZADO POR EQUIPO 01:
MOLERO MARÍN, JUAN
RODRÍGUEZ LÓPEZ, JULIO
MARTÍNEZ LORDA, JULIÁN**

Contenido

1.- Despliegue Inicial del Proyecto:	2
2.- Remodelación de la Base de Datos:	4
3.- Arreglo de las Funcionalidades:	9
3.1.- Creación de dos nuevos botones para exportar a PDF:	9
3.2.- Filtración de Videojuegos por género:	11
3.3.- Arreglo Rango Inválido en la Gráfica:	13
3.4.- Modificación Label:	15
4.- Bibliografía Empleada:	16

1.- Despliegue Inicial del Proyecto:

En este caso, para esta práctica nuestro cometido ha sido el de arreglar un proyecto proporcionado por el profesor, implementando también en el camino nuestra base de datos (videojuegos), y sustituyéndola por la de productos.

Para ello, lo primero ha sido descomprimir el proyecto y abrirlo con Visual Studio Code. Tras descomprimirlo, dentro de la carpeta de **'deployment'** hemos tenido que crear nuestro entorno virtual, para ello hemos ejecutado el comando que venimos usando durante todo el curso:

```
python -m venv .env
```

Una vez hemos creado nuestro entorno virtual dentro del directorio especificado, nuestro siguiente paso en esta práctica en concreto será el de activarlo y posteriormente preparar el entorno y levantar un contenedor en docker que permita la conexión con la base de datos.

Para ello, en este caso específico, se nos han proporcionado dos scripts, sin embargo, al intentar ejecutarlos saltaron fallos por la falta de una firma digital en los mismos, por lo que hemos tenido que modificar las políticas de ejecución de Windows antes de ejecutarlos mediante este comando:

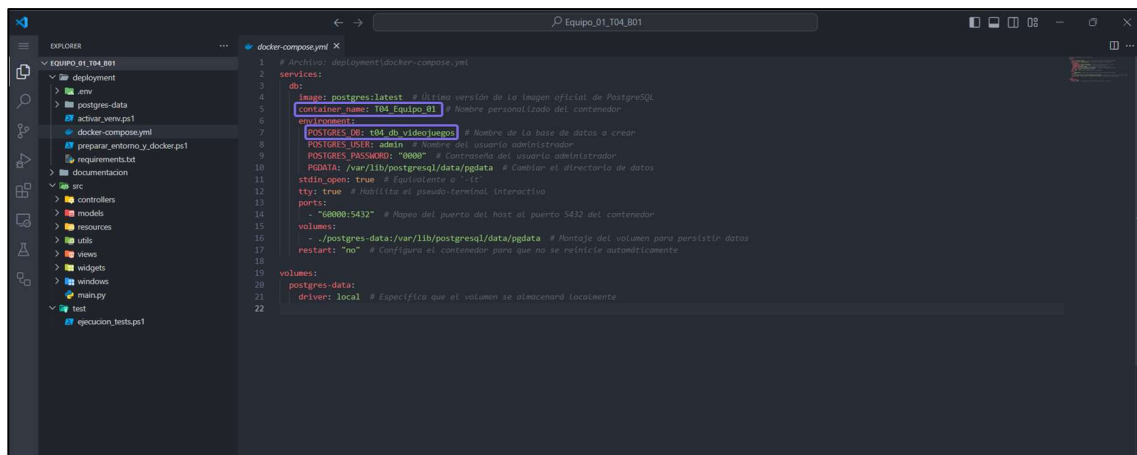
Set-ExecutionPolicy Unrestricted -Scope CurrentUser → Nos permitirá el uso de cualquier script en nuestro sistema

Una vez cambiada dicha política, es hora de ejecutar los scripts:

```
.\activar_venv.ps1
```

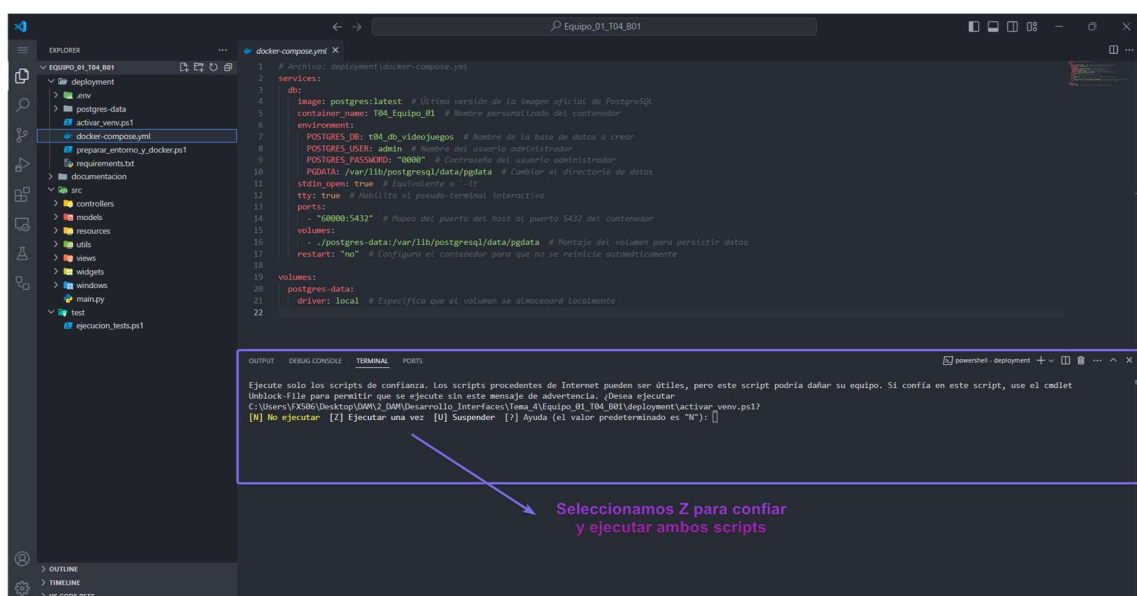
```
.\preparar_entorno_y_docker.ps1
```

Nota → Antes de la ejecución de los scripts, se ha realizado una modificación del docker-compose.yml, tal y como se puede apreciar aquí:



The screenshot shows the VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'EQUIPO_01_T04_B01' with folders like 'deployment', 'postgres-data', 'activar_vemv.ps1', 'docker-compose.yml', 'preparar_entorno_y_docker.ps1', 'requirements.txt', 'documentation', 'src', 'controllers', 'models', 'resources', 'utils', 'views', 'widgets', 'main.py', and 'test'. The code editor shows the 'docker-compose.yml' file with the following content:

```
1 # Archivo: deployment\docker-compose.yml
2 services:
3   db:
4     image: postgres:latest # Última versión de la imagen oficial de PostgreSQL
5     container_name: T04_Equipo_01 # Nombre personalizado del contenedor
6     environment:
7       POSTGRES_DB: t04_db_videojuegos # Nombre de la base de datos a crear
8       POSTGRES_USER: admin # Nombre del usuario administrador
9       POSTGRES_PASSWORD: "0000" # Contraseña del usuario administrador
10      PGDATA: /var/lib/postgresql/data/pgdata # Cambiar el directorio de datos
11      stdin_open: true # Equivalente a "-it"
12      tty: true # Habilita el pseudo-terminal interactivo
13      ports:
14        - "60000:5432" # Mapeo del puerto del host al puerto 5432 del contenedor
15      volumes:
16        - ./postgres-data:/var/lib/postgresql/data/pgdata # Montaje del volumen para persistir datos
17      restart: "no" # Configura el contenedor para que no se reinicie automáticamente
18
19 volumes:
20   postgres-data:
21     driver: local # Especifica que el volumen se almacenará localmente
```



The screenshot shows the VS Code editor with the same file explorer and code editor as the previous image. The terminal output is visible at the bottom, showing the command 'docker-compose up' and its output. A blue arrow points to the 'Z' key in the terminal output, indicating that it was pressed to confirm the execution of the scripts.

```
1 # Archivo: deployment\docker-compose.yml
2 services:
3   db:
4     image: postgres:latest # Última versión de la imagen oficial de PostgreSQL
5     container_name: T04_Equipo_01 # Nombre personalizado del contenedor
6     environment:
7       POSTGRES_DB: t04_db_videojuegos # Nombre de la base de datos a crear
8       POSTGRES_USER: admin # Nombre del usuario administrador
9       POSTGRES_PASSWORD: "0000" # Contraseña del usuario administrador
10      PGDATA: /var/lib/postgresql/data/pgdata # Cambiar el directorio de datos
11      stdin_open: true # Equivalente a "-it"
12      tty: true # Habilita el pseudo-terminal interactivo
13      ports:
14        - "60000:5432" # Mapeo del puerto del host al puerto 5432 del contenedor
15      volumes:
16        - ./postgres-data:/var/lib/postgresql/data/pgdata # Montaje del volumen para persistir datos
17      restart: "no" # Configura el contenedor para que no se reinicie automáticamente
18
19 volumes:
20   postgres-data:
21     driver: local # Especifica que el volumen se almacenará localmente
```

```
Output: Ejecute solo los scripts de confianza. Los scripts procedentes de Internet pueden ser útiles, pero este script podría dañar su equipo. Si confía en este script, use el cmdlet
Unlock-File para permitir que se ejecute sin este mensaje de advertencia. (Omita ejecutar
C:\Users\FX586\Desktop\DAW2\DAW\Desarrollo\Interfaces\Tema_4\Equipo_01_T04_B01\deployment\activar_vemv.ps1?
[N] No ejecutar [Z] Ejecutar una vez [U] Suspender [?] Ayuda (el valor predeterminado es "N"): []
```

Al ejecutar ambos scripts, ya tendremos el entorno listo para comenzar con la corrección del proyecto respectivamente.

A continuación vamos a realizar los cambios necesarios para poder poner en funcionamiento la aplicación.

2.- Remodelación de la Base de Datos:

En nuestro caso, el proyecto se nos ha otorgado con una Base de Datos basadas en productos. Al desplegarlo de forma inicial, podremos observar lo siguiente:

	fecha_agregado	stock	ventas	id_categoria	precio	producto	descripcion
1	2024-12-08	20	150	1	75.00	Teclado Mecánico	Teclado RGB con switches rojos
2	2024-12-08	50	300	1	30.00	Ratón inalámbrico	Ratón óptico inalámbrico
3	2024-12-08	30	120	1	50.00	Auriculares Bluetooth	Auriculares inalámbricos con micrófono
4	2024-12-08	40	80	1	25.00	Altavoces USB	Altavoces estéreo para ordenador
5	2024-12-08	25	75	1	60.00	Cámara Web HD	Cámara web de alta definición
6	2024-12-08	15	50	1	40.00	Teclado inalámbrico	Teclado Bluetooth ultrafino
7	2024-12-08	35	180	1	45.00	Ratón Gaming	Ratón con sensor óptico de alta precisión

Antes de ponernos manos a la obra para arreglar las funcionalidades principales de la ventana, hemos decidido previamente implementar nuestra base de datos de **‘Videojuegos’** ya, para ello, nuestro primer paso ha sido la creación de las entidades necesarias, en este caso concreto, videojuego, venta, género, usuario y rol:

```
class RolEntity:
    """
    Clase que representa un rol en la aplicación.
    Cada instancia corresponde a un registro en la tabla 'roles' de la base de datos.
    """
    def __init__(self, id_rol, nombre_rol):
        """
        Inicializa un objeto RolEntity con los datos de un rol.
        """
        self.id_rol = id_rol
        self.nombre_rol = nombre_rol

    def __dict__(self):
        """
        Convierte la instancia actual en un diccionario.
        """
        return {
            "id_rol": self.id_rol,
            "nombre_rol": self.nombre_rol
        }

class UsuarioEntity:
    """
    Clase que representa un usuario en la aplicación.
    Cada instancia corresponde a un registro en la tabla 'usuarios' de la base de datos.
    """
    def __init__(self, email, nombre_usuario, password, id_rol):
        """
        Inicializa un objeto UsuarioEntity con los datos de un usuario.
        """
        self.email = email
        self.nombre_usuario = nombre_usuario
        self.password = password
        self.id_rol = id_rol

    def __dict__(self):
        """
        Convierte la instancia actual en un diccionario.
        """
        return {
            "email": self.email,
            "nombre_usuario": self.nombre_usuario,
            "password": self.password,
            "id_rol": self.id_rol
        }
```

```
1 class VideojuegoEntity:
2     """
3     Clase que representa un videojuego en la aplicación.
4     Cada instancia corresponde a un registro en la tabla 'videojuegos' de la base de datos.
5
6     Métodos disponibles:
7     - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
8     - from_dict(data): Crea una instancia de VideojuegoEntity a partir de un diccionario.
9     - __str__(): Representación legible para humanos.
10    - __repr__(): Representación técnica detallada del objeto.
11    - __eq__(): Compara dos videojuegos basándose en su atributo 'titulo'.
12    - __hash__(): Genera un hash único basado en 'titulo'.
13    """
14
15    def __init__(self, titulo, genero, plataformas, desarrollador, fecha_lanzamiento, sinopsis):
16        """
17        Inicializa un objeto VideojuegoEntity con los datos del videojuego.
18
19        Parámetros:
20        - titulo (str): Título del videojuego.
21        - genero (str): Género del videojuego (e.g., Aventura, RPG).
22        - plataformas (str): Plataforma (e.g., PC, PlayStation, Xbox).
23        - desarrollador (str): Desarrollador del videojuego.
24        - fecha_lanzamiento (str): Fecha de lanzamiento del videojuego.
25        - sinopsis (str): Resumen o descripción breve del videojuego.
26        """
27        self.titulo = titulo
28        self.genero = genero
29        self.plataformas = plataformas
30        self.desarrollador = desarrollador
31        self.fecha_lanzamiento = fecha_lanzamiento
32        self.sinopsis = sinopsis
33
34    # __init__ (fin)
35
36    def to_dict(self):
37        """
38        Convierte la instancia actual en un diccionario.
39
40        Retorno:
41        - dict: Representación del videojuego como un diccionario.
42        """
43        return {
44            "titulo": self.titulo,
45            "genero": self.genero,
46            "plataformas": self.plataformas,
47            "desarrollador": self.desarrollador,
48            "fecha_lanzamiento": self.fecha_lanzamiento,
49            "sinopsis": self.sinopsis
50        }
51
52    # to_dict (fin)
53
54    @classmethod
55    def from_dict(cls, data):
56        """
57        Crea una instancia de VideojuegoEntity a partir de un diccionario.
58
59        Parámetros:
60        - data (dict): Diccionario con los datos del videojuego.
61
62        Retorno:
63        - VideojuegoEntity: Instancia de VideojuegoEntity creada a partir de los datos del diccionario.
64        """
65        return cls(
66            data.get("titulo"),
67            data.get("genero"),
68            data.get("plataformas"),
69            data.get("desarrollador"),
70            data.get("fecha_lanzamiento"),
71            data.get("sinopsis")
72        )
73
74    # from_dict (fin)
75
76    def __str__(self):
77        return f"VideojuegoEntity: {self.titulo} - {self.genero} - {self.plataformas} - {self.desarrollador} - {self.fecha_lanzamiento} - {self.sinopsis}"
78
79    def __repr__(self):
80        return f"VideojuegoEntity(titulo='{self.titulo}', genero='{self.genero}', plataformas='{self.plataformas}', desarrollador='{self.desarrollador}', fecha_lanzamiento='{self.fecha_lanzamiento}', sinopsis='{self.sinopsis}')"
81
82    def __eq__(self, other):
83        return self.titulo == other.titulo
84
85    def __hash__(self):
86        return hash(self.titulo)
87
88    # __eq__ (fin)
89    # __hash__ (fin)
90
91    # VideojuegoEntity (fin)
92
93
94 class VentaEntity:
95     """
96     Clase que representa una venta en la aplicación.
97     Cada instancia corresponde a un registro en la tabla 'ventas' de la base de datos.
98
99     Métodos disponibles:
100    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
101    - from_dict(data): Crea una instancia de VentaEntity a partir de un diccionario.
102    - __str__(): Representación legible para humanos.
103    - __repr__(): Representación técnica detallada del objeto.
104    - __eq__(): Compara dos ventas basándose en su atributo 'id_venta'.
105    - __hash__(): Genera un hash único basado en 'id_venta'.
106    """
107
108    def __init__(self, id_venta, codigo_producto, email_usuario, cantidad_vendida, fecha_venta):
109        """
110        Inicializa un objeto VentaEntity con los datos de una venta.
111
112        Parámetros:
113        - id_venta (int): ID único de la venta.
114        - codigo_producto (str): Código del producto vendido.
115        - email_usuario (str): Email del usuario que realizó la compra.
116        - cantidad_vendida (int): Cantidad de producto vendido.
117        - fecha_venta (str): Fecha en la que se realizó la venta.
118        """
119        self.id_venta = id_venta
120        self.codigo_producto = codigo_producto
121        self.email_usuario = email_usuario
122        self.cantidad_vendida = cantidad_vendida
123        self.fecha_venta = fecha_venta
124
125    # __init__ (fin)
126
127    def to_dict(self):
128        """
129        Convierte la instancia actual en un diccionario.
130
131        Retorno:
132        - dict: Representación de la venta como un diccionario.
133        """
134        return {
135            "id_venta": self.id_venta,
136            "codigo_producto": self.codigo_producto,
137            "email_usuario": self.email_usuario,
138            "cantidad_vendida": self.cantidad_vendida,
139            "fecha_venta": self.fecha_venta
140        }
141
142    # to_dict (fin)
143
144    @classmethod
145    def from_dict(cls, data):
146        """
147        Crea una instancia de VentaEntity a partir de un diccionario.
148
149        Parámetros:
150        - data (dict): Diccionario con los datos de la venta.
151
152        Retorno:
153        - VentaEntity: Instancia de VentaEntity creada a partir de los datos del diccionario.
154        """
155        return cls(
156            data.get("id_venta"),
157            data.get("codigo_producto"),
158            data.get("email_usuario"),
159            data.get("cantidad_vendida"),
160            data.get("fecha_venta")
161        )
162
163    # from_dict (fin)
164
165    def __str__(self):
166        return f"VentaEntity: {self.id_venta} - {self.codigo_producto} - {self.email_usuario} - {self.cantidad_vendida} - {self.fecha_venta}"
167
168    def __repr__(self):
169        return f"VentaEntity(id_venta='{self.id_venta}', codigo_producto='{self.codigo_producto}', email_usuario='{self.email_usuario}', cantidad_vendida='{self.cantidad_vendida}', fecha_venta='{self.fecha_venta}')"
170
171    def __eq__(self, other):
172        return self.id_venta == other.id_venta
173
174    def __hash__(self):
175        return hash(self.id_venta)
176
177    # __eq__ (fin)
178    # __hash__ (fin)
179
180    # VentaEntity (fin)
181
182
183 class GeneroEntity:
184     """
185     Clase que representa un género en la aplicación.
186     Cada instancia corresponde a un registro en la tabla 'generos' de la base de datos.
187
188     Métodos disponibles:
189    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
190    - from_dict(data): Crea una instancia de GeneroEntity a partir de un diccionario.
191    - __str__(): Representación legible para humanos.
192    - __repr__(): Representación técnica detallada del objeto.
193    - __eq__(): Compara dos géneros basándose en su atributo 'id_genero'.
194    - __hash__(): Genera un hash único basado en 'id_genero'.
195    """
196
197    def __init__(self, id_genero, nombre_genero):
198        """
199        Inicializa un objeto GeneroEntity con los datos de un género.
200
201        Parámetros:
202        - id_genero (int): ID único del género.
203        - nombre_genero (str): Nombre descriptivo del género (e.g., Aventura, RPG, Acción).
204        """
205        self.id_genero = id_genero
206        self.nombre_genero = nombre_genero
207
208    # __init__ (fin)
209
210    def to_dict(self):
211        """
212        Convierte la instancia actual en un diccionario.
213
214        Retorno:
215        - dict: Representación del género como un diccionario.
216        """
217        return {
218            "id_genero": self.id_genero,
219            "nombre_genero": self.nombre_genero
220        }
221
222    # to_dict (fin)
223
224    @classmethod
225    def from_dict(cls, data):
226        """
227        Crea una instancia de GeneroEntity a partir de un diccionario.
228
229        Parámetros:
230        - data (dict): Diccionario con los datos del género.
231
232        Retorno:
233        - GeneroEntity: Instancia de GeneroEntity creada a partir de los datos del diccionario.
234        """
235        return cls(
236            data.get("id_genero"),
237            data.get("nombre_genero")
238        )
239
240    # from_dict (fin)
241
242    def __str__(self):
243        return f"GeneroEntity: {self.id_genero} - {self.nombre_genero}"
244
245    def __repr__(self):
246        return f"GeneroEntity(id_genero='{self.id_genero}', nombre_genero='{self.nombre_genero}')"
247
248    def __eq__(self, other):
249        return self.id_genero == other.id_genero
250
251    def __hash__(self):
252        return hash(self.id_genero)
253
254    # __eq__ (fin)
255    # __hash__ (fin)
256
257    # GeneroEntity (fin)
258
259
260 class UsuarioEntity:
261     """
262     Clase que representa un usuario en la aplicación.
263     Cada instancia corresponde a un registro en la tabla 'usuarios' de la base de datos.
264
265     Métodos disponibles:
266    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
267    - from_dict(data): Crea una instancia de UsuarioEntity a partir de un diccionario.
268    - __str__(): Representación legible para humanos.
269    - __repr__(): Representación técnica detallada del objeto.
270    - __eq__(): Compara dos usuarios basándose en su atributo 'id_usuario'.
271    - __hash__(): Genera un hash único basado en 'id_usuario'.
272    """
273
274    def __init__(self, id_usuario, nombre_usuario, email_usuario, password_usuario):
275        """
276        Inicializa un objeto UsuarioEntity con los datos de un usuario.
277
278        Parámetros:
279        - id_usuario (int): ID único del usuario.
280        - nombre_usuario (str): Nombre de usuario.
281        - email_usuario (str): Email del usuario.
282        - password_usuario (str): Contraseña del usuario.
283        """
284        self.id_usuario = id_usuario
285        self.nombre_usuario = nombre_usuario
286        self.email_usuario = email_usuario
287        self.password_usuario = password_usuario
288
289    # __init__ (fin)
290
291    def to_dict(self):
292        """
293        Convierte la instancia actual en un diccionario.
294
295        Retorno:
296        - dict: Representación del usuario como un diccionario.
297        """
298        return {
299            "id_usuario": self.id_usuario,
300            "nombre_usuario": self.nombre_usuario,
301            "email_usuario": self.email_usuario,
302            "password_usuario": self.password_usuario
303        }
304
305    # to_dict (fin)
306
307    @classmethod
308    def from_dict(cls, data):
309        """
310        Crea una instancia de UsuarioEntity a partir de un diccionario.
311
312        Parámetros:
313        - data (dict): Diccionario con los datos del usuario.
314
315        Retorno:
316        - UsuarioEntity: Instancia de UsuarioEntity creada a partir de los datos del diccionario.
317        """
318        return cls(
319            data.get("id_usuario"),
320            data.get("nombre_usuario"),
321            data.get("email_usuario"),
322            data.get("password_usuario")
323        )
324
325    # from_dict (fin)
326
327    def __str__(self):
328        return f"UsuarioEntity: {self.id_usuario} - {self.nombre_usuario} - {self.email_usuario} - {self.password_usuario}"
329
330    def __repr__(self):
331        return f"UsuarioEntity(id_usuario='{self.id_usuario}', nombre_usuario='{self.nombre_usuario}', email_usuario='{self.email_usuario}', password_usuario='{self.password_usuario}')"
332
333    def __eq__(self, other):
334        return self.id_usuario == other.id_usuario
335
336    def __hash__(self):
337        return hash(self.id_usuario)
338
339    # __eq__ (fin)
340    # __hash__ (fin)
341
342    # UsuarioEntity (fin)
343
344
345 class ProductoEntity:
346     """
347     Clase que representa un producto en la aplicación.
348     Cada instancia corresponde a un registro en la tabla 'productos' de la base de datos.
349
350     Métodos disponibles:
351    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
352    - from_dict(data): Crea una instancia de ProductoEntity a partir de un diccionario.
353    - __str__(): Representación legible para humanos.
354    - __repr__(): Representación técnica detallada del objeto.
355    - __eq__(): Compara dos productos basándose en su atributo 'id_producto'.
356    - __hash__(): Genera un hash único basado en 'id_producto'.
357    """
358
359    def __init__(self, id_producto, nombre_producto, precio_producto, stock_producto):
360        """
361        Inicializa un objeto ProductoEntity con los datos de un producto.
362
363        Parámetros:
364        - id_producto (int): ID único del producto.
365        - nombre_producto (str): Nombre del producto.
366        - precio_producto (float): Precio del producto.
367        - stock_producto (int): Cantidad de stock del producto.
368        """
369        self.id_producto = id_producto
370        self.nombre_producto = nombre_producto
371        self.precio_producto = precio_producto
372        self.stock_producto = stock_producto
373
374    # __init__ (fin)
375
376    def to_dict(self):
377        """
378        Convierte la instancia actual en un diccionario.
379
380        Retorno:
381        - dict: Representación del producto como un diccionario.
382        """
383        return {
384            "id_producto": self.id_producto,
385            "nombre_producto": self.nombre_producto,
386            "precio_producto": self.precio_producto,
387            "stock_producto": self.stock_producto
388        }
389
390    # to_dict (fin)
391
392    @classmethod
393    def from_dict(cls, data):
394        """
395        Crea una instancia de ProductoEntity a partir de un diccionario.
396
397        Parámetros:
398        - data (dict): Diccionario con los datos del producto.
399
400        Retorno:
401        - ProductoEntity: Instancia de ProductoEntity creada a partir de los datos del diccionario.
402        """
403        return cls(
404            data.get("id_producto"),
405            data.get("nombre_producto"),
406            data.get("precio_producto"),
407            data.get("stock_producto")
408        )
409
410    # from_dict (fin)
411
412    def __str__(self):
413        return f"ProductoEntity: {self.id_producto} - {self.nombre_producto} - {self.precio_producto} - {self.stock_producto}"
414
415    def __repr__(self):
416        return f"ProductoEntity(id_producto='{self.id_producto}', nombre_producto='{self.nombre_producto}', precio_producto='{self.precio_producto}', stock_producto='{self.stock_producto}')"
417
418    def __eq__(self, other):
419        return self.id_producto == other.id_producto
420
421    def __hash__(self):
422        return hash(self.id_producto)
423
424    # __eq__ (fin)
425    # __hash__ (fin)
426
427    # ProductoEntity (fin)
428
429
430 class ReportEntity:
431     """
432     Clase que representa un reporte en la aplicación.
433     Cada instancia corresponde a un registro en la tabla 'reportes' de la base de datos.
434
435     Métodos disponibles:
436    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
437    - from_dict(data): Crea una instancia de ReportEntity a partir de un diccionario.
438    - __str__(): Representación legible para humanos.
439    - __repr__(): Representación técnica detallada del objeto.
440    - __eq__(): Compara dos reportes basándose en su atributo 'id_reporte'.
441    - __hash__(): Genera un hash único basado en 'id_reporte'.
442    """
443
444    def __init__(self, id_reporte, usuario_reporte, producto_reporte, cantidad_reporte):
445        """
446        Inicializa un objeto ReportEntity con los datos de un reporte.
447
448        Parámetros:
449        - id_reporte (int): ID único del reporte.
450        - usuario_reporte (str): Nombre del usuario que realizó el reporte.
451        - producto_reporte (str): Nombre del producto reportado.
452        - cantidad_reporte (int): Cantidad reportada.
453        """
454        self.id_reporte = id_reporte
455        self.usuario_reporte = usuario_reporte
456        self.producto_reporte = producto_reporte
457        self.cantidad_reporte = cantidad_reporte
458
459    # __init__ (fin)
460
461    def to_dict(self):
462        """
463        Convierte la instancia actual en un diccionario.
464
465        Retorno:
466        - dict: Representación del reporte como un diccionario.
467        """
468        return {
469            "id_reporte": self.id_reporte,
470            "usuario_reporte": self.usuario_reporte,
471            "producto_reporte": self.producto_reporte,
472            "cantidad_reporte": self.cantidad_reporte
473        }
474
475    # to_dict (fin)
476
477    @classmethod
478    def from_dict(cls, data):
479        """
480        Crea una instancia de ReportEntity a partir de un diccionario.
481
482        Parámetros:
483        - data (dict): Diccionario con los datos del reporte.
484
485        Retorno:
486        - ReportEntity: Instancia de ReportEntity creada a partir de los datos del diccionario.
487        """
488        return cls(
489            data.get("id_reporte"),
490            data.get("usuario_reporte"),
491            data.get("producto_reporte"),
492            data.get("cantidad_reporte")
493        )
494
495    # from_dict (fin)
496
497    def __str__(self):
498        return f"ReportEntity: {self.id_reporte} - {self.usuario_reporte} - {self.producto_reporte} - {self.cantidad_reporte}"
499
500    def __repr__(self):
501        return f"ReportEntity(id_reporte='{self.id_reporte}', usuario_reporte='{self.usuario_reporte}', producto_reporte='{self.producto_reporte}', cantidad_reporte='{self.cantidad_reporte}')"
502
503    def __eq__(self, other):
504        return self.id_reporte == other.id_reporte
505
506    def __hash__(self):
507        return hash(self.id_reporte)
508
509    # __eq__ (fin)
510    # __hash__ (fin)
511
512    # ReportEntity (fin)
513
514
515 class RolEntity:
516     """
517     Clase que representa un rol en la aplicación.
518     Cada instancia corresponde a un registro en la tabla 'roles' de la base de datos.
519
520     Métodos disponibles:
521    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
522    - from_dict(data): Crea una instancia de RolEntity a partir de un diccionario.
523    - __str__(): Representación legible para humanos.
524    - __repr__(): Representación técnica detallada del objeto.
525    - __eq__(): Compara dos roles basándose en su atributo 'id_rol'.
526    - __hash__(): Genera un hash único basado en 'id_rol'.
527    """
528
529    def __init__(self, id_rol, nombre_rol):
530        """
531        Inicializa un objeto RolEntity con los datos de un rol.
532
533        Parámetros:
534        - id_rol (int): ID único del rol.
535        - nombre_rol (str): Nombre del rol.
536        """
537        self.id_rol = id_rol
538        self.nombre_rol = nombre_rol
539
540    # __init__ (fin)
541
542    def to_dict(self):
543        """
544        Convierte la instancia actual en un diccionario.
545
546        Retorno:
547        - dict: Representación del rol como un diccionario.
548        """
549        return {
550            "id_rol": self.id_rol,
551            "nombre_rol": self.nombre_rol
552        }
553
554    # to_dict (fin)
555
556    @classmethod
557    def from_dict(cls, data):
558        """
559        Crea una instancia de RolEntity a partir de un diccionario.
560
561        Parámetros:
562        - data (dict): Diccionario con los datos del rol.
563
564        Retorno:
565        - RolEntity: Instancia de RolEntity creada a partir de los datos del diccionario.
566        """
567        return cls(
568            data.get("id_rol"),
569            data.get("nombre_rol")
570        )
571
572    # from_dict (fin)
573
574    def __str__(self):
575        return f"RolEntity: {self.id_rol} - {self.nombre_rol}"
576
577    def __repr__(self):
578        return f"RolEntity(id_rol='{self.id_rol}', nombre_rol='{self.nombre_rol}')"
579
580    def __eq__(self, other):
581        return self.id_rol == other.id_rol
582
583    def __hash__(self):
584        return hash(self.id_rol)
585
586    # __eq__ (fin)
587    # __hash__ (fin)
588
589    # RolEntity (fin)
590
591
592 class CategoriaEntity:
593     """
594     Clase que representa una categoría en la aplicación.
595     Cada instancia corresponde a un registro en la tabla 'categorias' de la base de datos.
596
597     Métodos disponibles:
598    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
599    - from_dict(data): Crea una instancia de CategoriaEntity a partir de un diccionario.
600    - __str__(): Representación legible para humanos.
601    - __repr__(): Representación técnica detallada del objeto.
602    - __eq__(): Compara dos categorías basándose en su atributo 'id_categoria'.
603    - __hash__(): Genera un hash único basado en 'id_categoria'.
604    """
605
606    def __init__(self, id_categoria, nombre_categoria):
607        """
608        Inicializa un objeto CategoriaEntity con los datos de una categoría.
609
610        Parámetros:
611        - id_categoria (int): ID único de la categoría.
612        - nombre_categoria (str): Nombre de la categoría.
613        """
614        self.id_categoria = id_categoria
615        self.nombre_categoria = nombre_categoria
616
617    # __init__ (fin)
618
619    def to_dict(self):
620        """
621        Convierte la instancia actual en un diccionario.
622
623        Retorno:
624        - dict: Representación de la categoría como un diccionario.
625        """
626        return {
627            "id_categoria": self.id_categoria,
628            "nombre_categoria": self.nombre_categoria
629        }
630
631    # to_dict (fin)
632
633    @classmethod
634    def from_dict(cls, data):
635        """
636        Crea una instancia de CategoriaEntity a partir de un diccionario.
637
638        Parámetros:
639        - data (dict): Diccionario con los datos de la categoría.
640
641        Retorno:
642        - CategoriaEntity: Instancia de CategoriaEntity creada a partir de los datos del diccionario.
643        """
644        return cls(
645            data.get("id_categoria"),
646            data.get("nombre_categoria")
647        )
648
649    # from_dict (fin)
650
651    def __str__(self):
652        return f"CategoriaEntity: {self.id_categoria} - {self.nombre_categoria}"
653
654    def __repr__(self):
655        return f"CategoriaEntity(id_categoria='{self.id_categoria}', nombre_categoria='{self.nombre_categoria}')"
656
657    def __eq__(self, other):
658        return self.id_categoria == other.id_categoria
659
660    def __hash__(self):
661        return hash(self.id_categoria)
662
663    # __eq__ (fin)
664    # __hash__ (fin)
665
666    # CategoriaEntity (fin)
667
668
669 class InicializadorEntity:
670     """
671     Clase que representa un inicializador en la aplicación.
672     Cada instancia corresponde a un registro en la tabla 'inicializadores' de la base de datos.
673
674     Métodos disponibles:
675    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
676    - from_dict(data): Crea una instancia de InicializadorEntity a partir de un diccionario.
677    - __str__(): Representación legible para humanos.
678    - __repr__(): Representación técnica detallada del objeto.
679    - __eq__(): Compara dos inicializadores basándose en su atributo 'id_inicializador'.
680    - __hash__(): Genera un hash único basado en 'id_inicializador'.
681    """
682
683    def __init__(self, id_inicializador, nombre_inicializador):
684        """
685        Inicializa un objeto InicializadorEntity con los datos de un inicializador.
686
687        Parámetros:
688        - id_inicializador (int): ID único del inicializador.
689        - nombre_inicializador (str): Nombre del inicializador.
690        """
691        self.id_inicializador = id_inicializador
692        self.nombre_inicializador = nombre_inicializador
693
694    # __init__ (fin)
695
696    def to_dict(self):
697        """
698        Convierte la instancia actual en un diccionario.
699
700        Retorno:
701        - dict: Representación del inicializador como un diccionario.
702        """
703        return {
704            "id_inicializador": self.id_inicializador,
705            "nombre_inicializador": self.nombre_inicializador
706        }
707
708    # to_dict (fin)
709
710    @classmethod
711    def from_dict(cls, data):
712        """
713        Crea una instancia de InicializadorEntity a partir de un diccionario.
714
715        Parámetros:
716        - data (dict): Diccionario con los datos del inicializador.
717
718        Retorno:
719        - InicializadorEntity: Instancia de InicializadorEntity creada a partir de los datos del diccionario.
720        """
721        return cls(
722            data.get("id_inicializador"),
723            data.get("nombre_inicializador")
724        )
725
726    # from_dict (fin)
727
728    def __str__(self):
729        return f"InicializadorEntity: {self.id_inicializador} - {self.nombre_inicializador}"
730
731    def __repr__(self):
732        return f"InicializadorEntity(id_inicializador='{self.id_inicializador}', nombre_inicializador='{self.nombre_inicializador}')"
733
734    def __eq__(self, other):
735        return self.id_inicializador == other.id_inicializador
736
737    def __hash__(self):
738        return hash(self.id_inicializador)
739
740    # __eq__ (fin)
741    # __hash__ (fin)
742
743    # InicializadorEntity (fin)
744
745
746 class ReportModelEntity:
747     """
748     Clase que representa un modelo de reporte en la aplicación.
749     Cada instancia corresponde a un registro en la tabla 'report_modelos' de la base de datos.
750
751     Métodos disponibles:
752    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
753    - from_dict(data): Crea una instancia de ReportModelEntity a partir de un diccionario.
754    - __str__(): Representación legible para humanos.
755    - __repr__(): Representación técnica detallada del objeto.
756    - __eq__(): Compara dos modelos de reporte basándose en su atributo 'id_report_modelo'.
757    - __hash__(): Genera un hash único basado en 'id_report_modelo'.
758    """
759
760    def __init__(self, id_report_modelo, nombre_report_modelo):
761        """
762        Inicializa un objeto ReportModelEntity con los datos de un modelo de reporte.
763
764        Parámetros:
765        - id_report_modelo (int): ID único del modelo de reporte.
766        - nombre_report_modelo (str): Nombre del modelo de reporte.
767        """
768        self.id_report_modelo = id_report_modelo
769        self.nombre_report_modelo = nombre_report_modelo
770
771    # __init__ (fin)
772
773    def to_dict(self):
774        """
775        Convierte la instancia actual en un diccionario.
776
777        Retorno:
778        - dict: Representación del modelo de reporte como un diccionario.
779        """
780        return {
781            "id_report_modelo": self.id_report_modelo,
782            "nombre_report_modelo": self.nombre_report_modelo
783        }
784
785    # to_dict (fin)
786
787    @classmethod
788    def from_dict(cls, data):
789        """
790        Crea una instancia de ReportModelEntity a partir de un diccionario.
791
792        Parámetros:
793        - data (dict): Diccionario con los datos del modelo de reporte.
794
795        Retorno:
796        - ReportModelEntity: Instancia de ReportModelEntity creada a partir de los datos del diccionario.
797        """
798        return cls(
799            data.get("id_report_modelo"),
800            data.get("nombre_report_modelo")
801        )
802
803    # from_dict (fin)
804
805    def __str__(self):
806        return f"ReportModelEntity: {self.id_report_modelo} - {self.nombre_report_modelo}"
807
808    def __repr__(self):
809        return f"ReportModelEntity(id_report_modelo='{self.id_report_modelo}', nombre_report_modelo='{self.nombre_report_modelo}')"
810
811    def __eq__(self, other):
812        return self.id_report_modelo == other.id_report_modelo
813
814    def __hash__(self):
815        return hash(self.id_report_modelo)
816
817    # __eq__ (fin)
818    # __hash__ (fin)
819
820    # ReportModelEntity (fin)
821
822
823 class RolUsuarioEntity:
824     """
825     Clase que representa un rol de usuario en la aplicación.
826     Cada instancia corresponde a un registro en la tabla 'rol_usuario' de la base de datos.
827
828     Métodos disponibles:
829    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
830    - from_dict(data): Crea una instancia de RolUsuarioEntity a partir de un diccionario.
831    - __str__(): Representación legible para humanos.
832    - __repr__(): Representación técnica detallada del objeto.
833    - __eq__(): Compara dos roles de usuario basándose en su atributo 'id_rol_usuario'.
834    - __hash__(): Genera un hash único basado en 'id_rol_usuario'.
835    """
836
837    def __init__(self, id_rol_usuario, rol_usuario):
838        """
839        Inicializa un objeto RolUsuarioEntity con los datos de un rol de usuario.
840
841        Parámetros:
842        - id_rol_usuario (int): ID único del rol de usuario.
843        - rol_usuario (str): Nombre del rol de usuario.
844        """
845        self.id_rol_usuario = id_rol_usuario
846        self.rol_usuario = rol_usuario
847
848    # __init__ (fin)
849
850    def to_dict(self):
851        """
852        Convierte la instancia actual en un diccionario.
853
854        Retorno:
855        - dict: Representación del rol de usuario como un diccionario.
856        """
857        return {
858            "id_rol_usuario": self.id_rol_usuario,
859            "rol_usuario": self.rol_usuario
860        }
861
862    # to_dict (fin)
863
864    @classmethod
865    def from_dict(cls, data):
866        """
867        Crea una instancia de RolUsuarioEntity a partir de un diccionario.
868
869        Parámetros:
870        - data (dict): Diccionario con los datos del rol de usuario.
871
872        Retorno:
873        - RolUsuarioEntity: Instancia de RolUsuarioEntity creada a partir de los datos del diccionario.
874        """
875        return cls(
876            data.get("id_rol_usuario"),
877            data.get("rol_usuario")
878        )
879
880    # from_dict (fin)
881
882    def __str__(self):
883        return f"RolUsuarioEntity: {self.id_rol_usuario} - {self.rol_usuario}"
884
885    def __repr__(self):
886        return f"RolUsuarioEntity(id_rol_usuario='{self.id_rol_usuario}', rol_usuario='{self.rol_usuario}')"
887
888    def __eq__(self, other):
889        return self.id_rol_usuario == other.id_rol_usuario
890
891    def __hash__(self):
892        return hash(self.id_rol_usuario)
893
894    # __eq__ (fin)
895    # __hash__ (fin)
896
897    # RolUsuarioEntity (fin)
898
899
900 class InicializadorModelEntity:
901     """
902     Clase que representa un modelo de inicializador en la aplicación.
903     Cada instancia corresponde a un registro en la tabla 'inicializador_modelos' de la base de datos.
904
905     Métodos disponibles:
906    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
907    - from_dict(data): Crea una instancia de InicializadorModelEntity a partir de un diccionario.
908    - __str__(): Representación legible para humanos.
909    - __repr__(): Representación técnica detallada del objeto.
910    - __eq__(): Compara dos modelos de inicializador basándose en su atributo 'id_inicializador_modelo'.
911    - __hash__(): Genera un hash único basado en 'id_inicializador_modelo'.
912    """
913
914    def __init__(self, id_inicializador_modelo, nombre_inicializador_modelo):
915        """
916        Inicializa un objeto InicializadorModelEntity con los datos de un modelo de inicializador.
917
918        Parámetros:
919        - id_inicializador_modelo (int): ID único del modelo de inicializador.
920        - nombre_inicializador_modelo (str): Nombre del modelo de inicializador.
921        """
922        self.id_inicializador_modelo = id_inicializador_modelo
923        self.nombre_inicializador_modelo = nombre_inicializador_modelo
924
925    # __init__ (fin)
926
927    def to_dict(self):
928        """
929        Convierte la instancia actual en un diccionario.
930
931        Retorno:
932        - dict: Representación del modelo de inicializador como un diccionario.
933        """
934        return {
935            "id_inicializador_modelo": self.id_inicializador_modelo,
936            "nombre_inicializador_modelo": self.nombre_inicializador_modelo
937        }
938
939    # to_dict (fin)
940
941    @classmethod
942    def from_dict(cls, data):
943        """
944        Crea una instancia de InicializadorModelEntity a partir de un diccionario.
945
946        Parámetros:
947        - data (dict): Diccionario con los datos del modelo de inicializador.
948
949        Retorno:
950        - InicializadorModelEntity: Instancia de InicializadorModelEntity creada a partir de los datos del diccionario.
951        """
952        return cls(
953            data.get("id_inicializador_modelo"),
954            data.get("nombre_inicializador_modelo")
955        )
956
957    # from_dict (fin)
958
959    def __str__(self):
960        return f"InicializadorModelEntity: {self.id_inicializador_modelo} - {self.nombre_inicializador_modelo}"
961
962    def __repr__(self):
963        return f"InicializadorModelEntity(id_inicializador_modelo='{self.id_inicializador_modelo}', nombre_inicializador_modelo='{self.nombre_inicializador_modelo}')"
964
965    def __eq__(self, other):
966        return self.id_inicializador_modelo == other.id_inicializador_modelo
967
968    def __hash__(self):
969        return hash(self.id_inicializador_modelo)
970
971    # __eq__ (fin)
972    # __hash__ (fin)
973
974    # InicializadorModelEntity (fin)
975
976
977 class ReportControllerEntity:
978     """
979     Clase que representa un controlador de reporte en la aplicación.
980     Cada instancia corresponde a un registro en la tabla 'report_controllers' de la base de datos.
981
982     Métodos disponibles:
983    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
984    - from_dict(data): Crea una instancia de ReportControllerEntity a partir de un diccionario.
985    - __str__(): Representación legible para humanos.
986    - __repr__(): Representación técnica detallada del objeto.
987    - __eq__(): Compara dos controladores de reporte basándose en su atributo 'id_report_controller'.
988    - __hash__(): Genera un hash único basado en 'id_report_controller'.
989    """
990
991    def __init__(self, id_report_controller, nombre_report_controller):
992        """
993        Inicializa un objeto ReportControllerEntity con los datos de un controlador de reporte.
994
995        Parámetros:
996        - id_report_controller (int): ID único del controlador de reporte.
997        - nombre_report_controller (str): Nombre del controlador de reporte.
998        """
999        self.id_report_controller = id_report_controller
1000        self.nombre_report_controller = nombre_report_controller
1001
1002    # __init__ (fin)
1003
1004    def to_dict(self):
1005        """
1006        Convierte la instancia actual en un diccionario.
1007
1008        Retorno:
1009        - dict: Representación del controlador de reporte como un diccionario.
1010        """
1011        return {
1012            "id_report_controller": self.id_report_controller,
1013            "nombre_report_controller": self.nombre_report_controller
1014        }
1015
1016    # to_dict (fin)
1017
1018    @classmethod
1019    def from_dict(cls, data):
1020        """
1021        Crea una instancia de ReportControllerEntity a partir de un diccionario.
1022
1023        Parámetros:
1024        - data (dict): Diccionario con los datos del controlador de reporte.
1025
1026        Retorno:
1027        - ReportControllerEntity: Instancia de ReportControllerEntity creada a partir de los datos del diccionario.
1028        """
1029        return cls(
1030            data.get("id_report_controller"),
1031            data.get("nombre_report_controller")
1032        )
1033
1034    # from_dict (fin)
1035
1036    def __str__(self):
1037        return f"ReportControllerEntity: {self.id_report_controller} - {self.nombre_report_controller}"
1038
1039    def __repr__(self):
1040        return f"ReportControllerEntity(id_report_controller='{self.id_report_controller}', nombre_report_controller='{self.nombre_report_controller}')"
1041
1042    def __eq__(self, other):
1043        return self.id_report_controller == other.id_report_controller
1044
1045    def __hash__(self):
1046        return hash(self.id_report_controller)
1047
1048    # __eq__ (fin)
1049    # __hash__ (fin)
1050
1051    # ReportControllerEntity (fin)
1052
1053
1054 class RolReportEntity:
1055     """
1056     Clase que representa un rol de reporte en la aplicación.
1057     Cada instancia corresponde a un registro en la tabla 'rol_report' de la base de datos.
1058
1059     Métodos disponibles:
1060    - to_dict(): Convierte la instancia en un diccionario para facilitar la manipulación de
1061    - from_dict(data): Crea una instancia de RolReportEntity a partir de un diccionario.
1062    - __str__(): Representación legible para humanos.
1063    - __repr__(): Representación técnica detallada del objeto.
1064    - __eq__(): Compara dos roles de reporte basándose en su atributo 'id_rol_report'.
1065    - __hash__(): Genera un hash único basado en 'id_rol_report'.
1066    """
1067
1068    def __init__(self, id_rol_report, rol_report):
1069        """
1070        Inicializa un objeto RolReportEntity con los datos de un rol de reporte.
1071
1072        Parámetros:
1073        - id_rol_report (int): ID único del rol de reporte.
1074        - rol_report (str): Nombre del rol de reporte.
1075        """
1076       
```



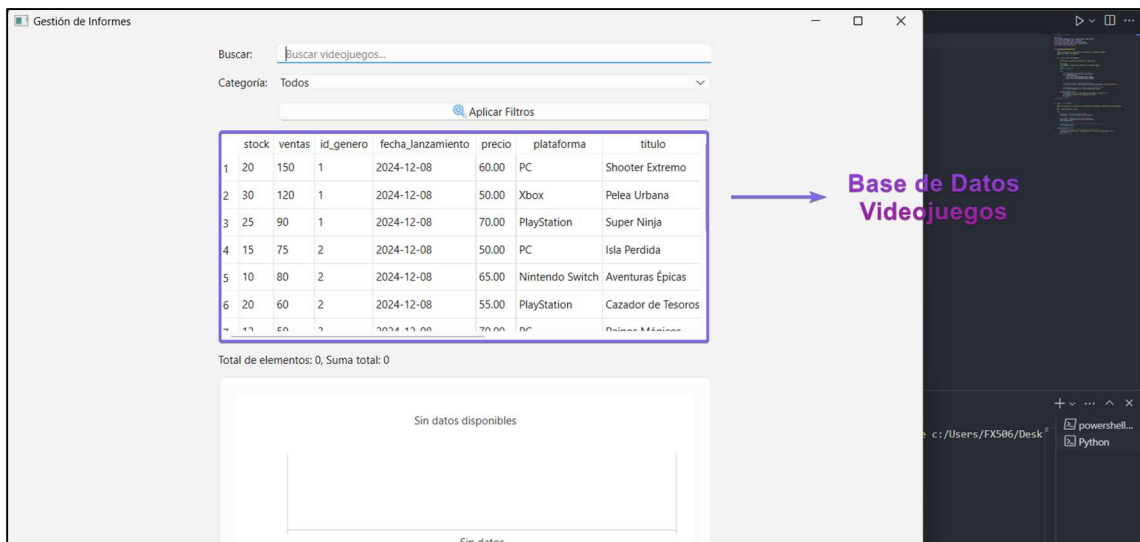
Tras comprobar que funciona, se ha implementado dentro del proyecto, tal y como podemos observar aquí.

Nota: se implementaron consultas adicionales para verificar su funcionalidad:


```
1 -- Crear la base de datos
2 -- # Archivo: inicializacion_db.sql #
3 -- # # #
4
5 -- Crear la tabla 'roles' para definir los perfiles de los usuarios
6 CREATE TABLE IF NOT EXISTS roles (
7     id_rol SERIAL PRIMARY KEY,
8     nombre_rol VARCHAR(20) UNIQUE NOT NULL
9 );
10
11 -- Insertar datos iniciales en la tabla 'roles'
12 -- Se utilizan los usuarios: 'admin', 'desarrollador' y 'jugador' como roles de ejemplo
13 INSERT INTO roles (nombre_rol)
14 VALUES
15     ('admin'),
16     ('desarrollador'),
17     ('jugador');
18 ON CONFLICT DO NOTHING;
19
20 -- Crear la tabla 'usuarios' con referencia a 'roles'
21 -- Cada usuario está asociado a un rol específico mediante la columna 'id_rol'
22 CREATE TABLE IF NOT EXISTS usuarios (
23     email VARCHAR(255) PRIMARY KEY,
24     nombre_usuario VARCHAR(255) NOT NULL,
25     password VARCHAR(255) NOT NULL,
26     id_rol INT REFERENCES roles(id_rol) DEFAULT 3 -- Rol de jugador por defecto
27 );
28
29 -- Insertar datos iniciales en la tabla 'usuarios'
30 -- Definimos tres usuarios de ejemplo con diferentes roles
31 INSERT INTO usuarios (email, nombre_usuario, password, id_rol)
32 VALUES
33     ('admin@example.com', 'admin', 'adminpass', 1),
34     ('dev@example.com', 'dev', 'devpass', 2),
35     ('jugador@example.com', 'jugador', 'jugadpass', 3);
36 ON CONFLICT (email) DO NOTHING;
37
38 -- Crear la tabla 'generos' para clasificar videojuegos, con un índice de 'nombre_genero'
39 -- Se utiliza un índice transitable o modificado para evitar duplicados (ej: 'Acción' y 'accion' serán tratados como iguales)
40 CREATE TABLE IF NOT EXISTS generos (
41     id_genero SERIAL PRIMARY KEY,
42     nombre_genero VARCHAR(255) NOT NULL UNIQUE
43 );
44
45 -- Añadir un índice único en 'nombre_genero' en minúsculas para evitar duplicados transatables o modificados
46 CREATE UNIQUE INDEX IF NOT EXISTS idx_nombre_genero_lower
47 ON generos ((LOWER(nombre_genero)));
48
49 -- Insertar datos de ejemplo en la tabla 'generos'
50 -- Se insertan cuatro géneros de videojuegos
51 INSERT INTO generos (nombre_genero)
52 VALUES
53     ('Acción'),
54     ('Aventura'),
55     ('RPG'),
56     ('Deportes');
57 ON CONFLICT (nombre_genero) DO NOTHING;
58
59 -- Crear la tabla 'videojuegos' para almacenar información sobre los videojuegos
60 -- Cada videojuego se relaciona con un género mediante 'id_genero'
61 CREATE TABLE IF NOT EXISTS videojuegos (
62     codigo VARCHAR(10) PRIMARY KEY,
63     titulo VARCHAR(255) NOT NULL,
64     descripcion VARCHAR(255),
65     precio DECIMAL(10, 2) NOT NULL,
66     plataforma VARCHAR(50) NOT NULL,
67     stock INT NOT NULL,
68     ventas INT NOT NULL DEFAULT 0,
69     id_genero INT REFERENCES generos(id_genero),
70     fecha_lanzamiento DATE DEFAULT CURRENT_DATE
71 );
72
73 -- Insertar datos de ejemplo en la tabla 'videojuegos'
74 -- Definimos videojuegos de ejemplo para los géneros de 'Acción', 'Aventura', 'RPG' y 'Deportes'
75 INSERT INTO videojuegos (codigo, titulo, descripcion, precio, plataforma, stock, ventas, id_genero)
76 VALUES
77     -- Acción
78     ('001', 'Shooter Extremo', 'Juego de disparos intensos', 60, 'PC', 20, 150, 1),
79     ('002', 'Polso Infame', 'Juego de lucha en las calles', 50, 'Xbox', 30, 120, 1),
80     ('003', 'Supra Ninja', 'Aventuras y acción ninja', 70, 'Playstation', 25, 90, 1),
81
82     -- Aventura
83     ('004', 'Isla Perdida', 'Exploración en una isla misteriosa', 50, 'PC', 15, 75, 2),
84     ('005', 'Aventuras Épicas', 'Viaje por mundos fantásticos', 65, 'Nintendo Switch', 18, 80, 2),
85     ('006', 'Cazador de Tesoros', 'Busca el tesoro perdido', 55, 'Playstation', 20, 60, 2),
86
87     -- RPG
88     ('007', 'Reinos Mágicos', 'Conquista tierras mágicas', 70, 'PC', 12, 50, 3),
89     ('008', 'Guerreros del Alba', 'RPG épico con batallas tácticas', 60, 'Xbox', 11, 45, 3),
90     ('009', 'Legendarios Dragones', 'Historia épica de héroes', 80, 'PS', 14, 30, 3),
91
92     -- Deportes
93     ('010', 'Fútbol Pro', 'Simulación de fútbol realista', 40, 'PC', 50, 200, 4),
94     ('011', 'Básquet Estelar', 'Juego de baloncesto arcade', 30, 'Playstation', 30, 150, 4),
95     ('012', 'Tennis Master', 'Juego de tenis competitivo', 50, 'Nintendo Switch', 20, 100, 4)
96 ON CONFLICT (codigo) DO NOTHING;
97
98 -- Crear la tabla 'ventas' para registrar ventas individuales, con relación a 'usuarios' y 'videojuegos'
99 -- Cada venta registra el videojuego vendido, el usuario que realizó la compra, y la cantidad
100 CREATE TABLE IF NOT EXISTS ventas (
101     id_venta SERIAL PRIMARY KEY,
102     codigo_videojuego VARCHAR(10) REFERENCES videojuegos(codigo),
103     email_usuario VARCHAR(255) REFERENCES usuarios(email),
104     cantidad_vendida INT NOT NULL,
105     fecha_venta DATE DEFAULT CURRENT_DATE
106 );
107
108 -- Insertar datos de ejemplo en la tabla 'ventas'
109 -- Definimos ventas de ejemplo para relacionar videojuegos y usuarios
110 INSERT INTO ventas (codigo_videojuego, email_usuario, cantidad_vendida, fecha_venta)
111 VALUES
112     ('001', 'jugador@example.com', 2, '2024-11-13'),
113     ('004', 'jugador@example.com', 1, '2024-11-14'),
114     ('007', 'dev@example.com', 1, '2024-11-15'),
115     ('010', 'jugador@example.com', 3, '2024-11-15'),
116     ('012', 'jugador@example.com', 1, '2024-11-16');
117
118 -- # Consultas de verificación de datos en las tablas #
119 -- # # #
120
121 -- Consultar todos los datos de la tabla 'roles'
122 SELECT * FROM roles;
123
124 -- Consultar todos los datos de la tabla 'usuarios'
125 SELECT * FROM usuarios;
126
127 -- Consultar todos los datos de la tabla 'generos'
128 SELECT * FROM generos;
129
130 -- Consultar todos los datos de la tabla 'videojuegos'
131 SELECT * FROM videojuegos;
132
133 -- Consultar todos los datos de la tabla 'ventas'
134 SELECT * FROM ventas;
135
136 -- Consultar todos los usuarios y sus roles
137 -- Esta consulta une las tablas 'usuarios' y 'roles' para mostrar el rol de cada usuario
138 SELECT u.email, u.nombre_usuario, r.nombre_rol
139 FROM usuarios u
140 JOIN roles r ON u.id_rol = r.id_rol;
141
142 -- Consultar todos los videojuegos con su género
143 -- Esta consulta une las tablas 'videojuegos' y 'generos' para mostrar el género de cada videojuego
144 SELECT v.codigo, v.titulo, v.descripcion, v.precio, v.plataforma, v.stock, v.ventas, g.nombre_genero
145 FROM videojuegos v
146 JOIN generos g ON v.id_genero = g.id_genero;
147
148 -- Consultar todas las ventas con detalles de usuario y videojuego
149 -- Esta consulta une las tablas 'ventas', 'usuarios' y 'videojuegos' para mostrar quién compró qué videojuego
150 SELECT v.id_venta, u.email AS usuario, vg.titulo, v.cantidad_vendida, v.fecha_venta
151 FROM ventas v
152 JOIN usuarios u ON v.email_usuario = u.email
153 JOIN videojuegos vg ON v.codigo_videojuego = vg.codigo;
154
155 -- Consultar el stock total de videojuegos
156 -- Esta consulta muestra el stock de cada videojuego, el stock disponible y el total vendido.
157 SELECT titulo, stock, ventas FROM videojuegos;
158
159 -- Fin del archivo 'inicializacion_db.sql'
```


Una vez tenemos la base de datos, es momento de modificar la clase del controlador proporcionada, para modificar la base de datos que va a tratar respectivamente. Para ello, se implementaron los siguientes cambios dentro de la clase:

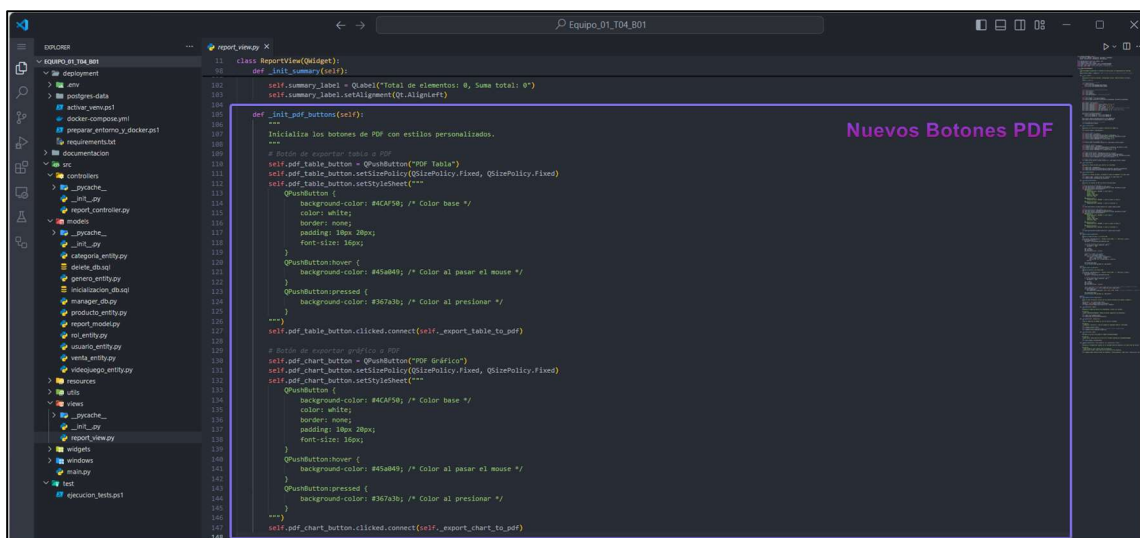
[illegible]

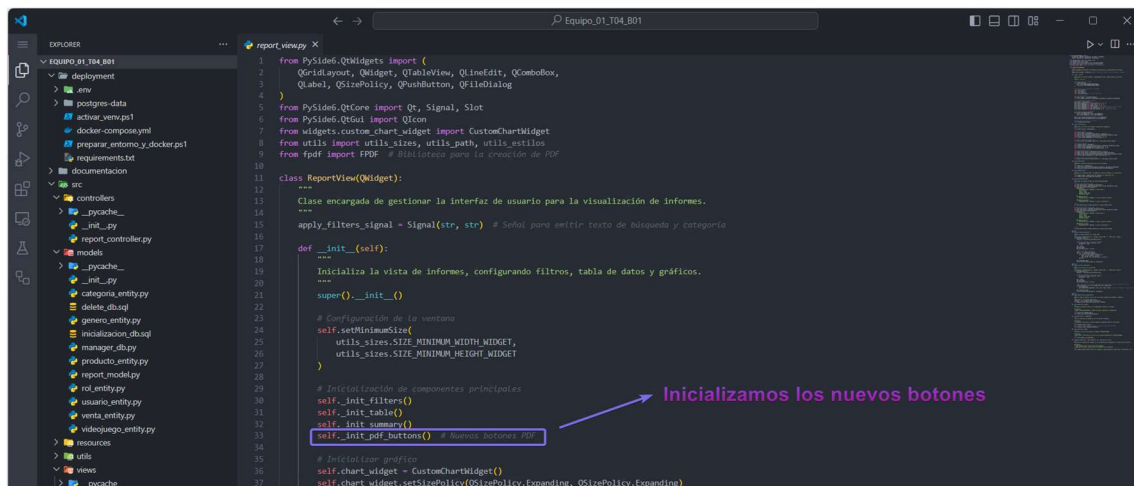


3.- Arreglo de las Funcionalidades:

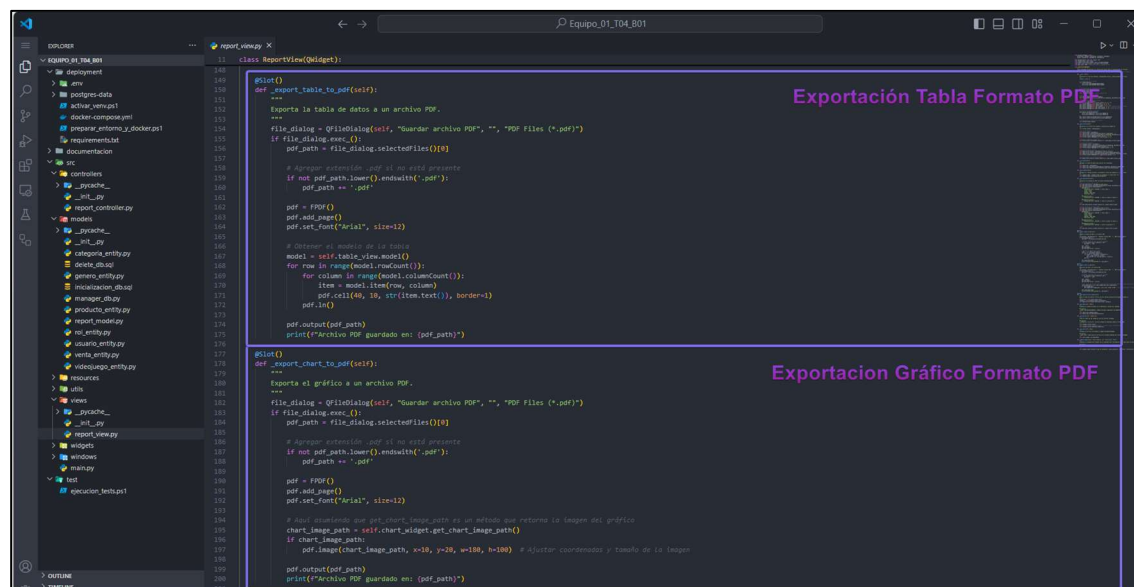
3.1.- Creación de dos nuevos botones para exportar a PDF:

Uno de los requerimientos especificados ha sido la implementación de un botón que exportará los resultados obtenidos tanto en la gráfica como en la tabla de la vista a formato PDF. Para ello, hemos tomado la decisión en nuestro caso de implementar dos, separando la obtención de dichos resultados en dos documentos distintos:

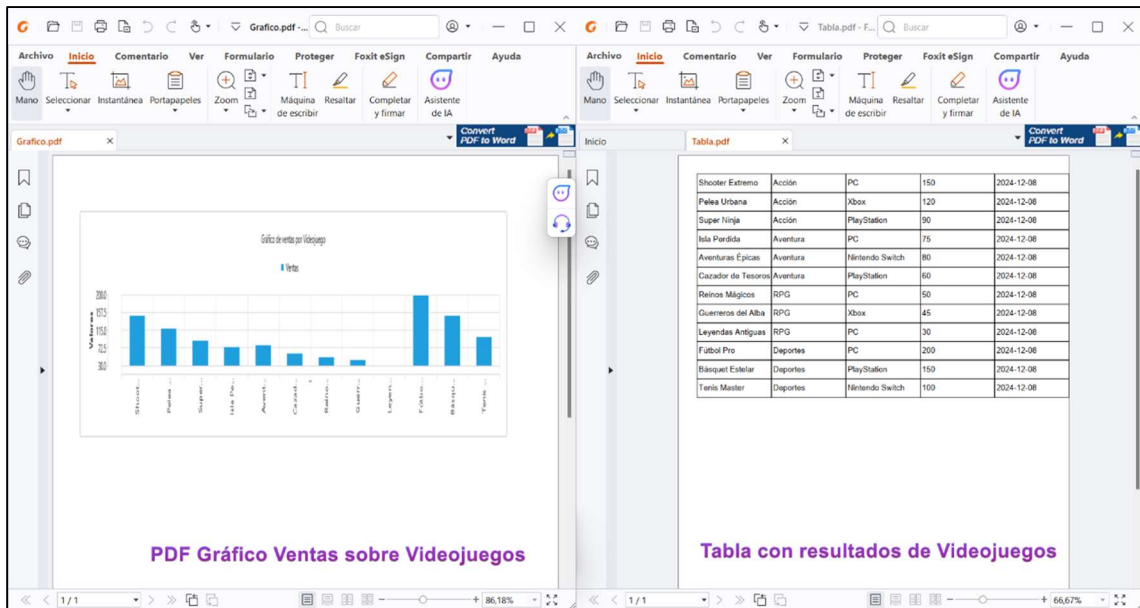




Además de ello, como se puede apreciar en la primera captura, cada uno de ellos está conectado a su método para poder exportar en el formato especificado de forma concreta, los cuáles podemos apreciar a continuación:



Implementando todos estos cambios, al poner en marcha las posteriores funcionalidades que se van a explicar, se podrán generar resultados como los siguientes:



Importante → Se ha explicado el proceso para exportarlo, el cual cabe recalcar que necesitará la instalación de la librería **fpdf**. Además de todo ello, para generar los documentos, en las clases **widgets** se ha tenido que instalar también la librería **reportlab**.

3.2.- Filtración de Videojuegos por género:

Una de las funcionalidades principales que debíamos de conseguir dentro de nuestra Aplicación era poder filtrar los resultados, en nuestro caso concreto hemos escogido el género del videojuego.

Para ello, se ha preseleccionado todas las categorías existentes dentro de nuestra Base de Datos (**Acción, Aventuras, RPG y Deportes**), así como una categoría extra denominada **'Todos'**, el cual nos proporcionará todos los videojuegos concretamente.

Todo esto se ha conseguido mediante la implementación de modificaciones dentro de la función **apply_filters**, situada en el controlador concretamente. Los cambios han sido los siguientes:

```

class ReportController:
    def __init__(self, report_view: ReportView, report_model: ReportModel, popup_parent: Optional[QWidget] = None):
        """
        Inicializa el controlador.
        """
        # 1. Se inicializa el controlador
        self._report_view = report_view
        self._report_model = report_model
        self._popup_parent = popup_parent

        # 2. Se inicializa la vista
        self._initialize_view()

        # 3. Se conecta la señal de aplicar filtros
        self._apply_filters_signal.connect(self._apply_filters)

        # 4. Se inicializa el controlador
        self._initialize_controller()

        # 5. Se inicializa el controlador
        self._initialize_controller()

        # 6. Se inicializa el controlador
        self._initialize_controller()

    def _initialize_view(self):
        """
        Inicializa la vista.
        """
        # 1. Se inicializa la vista
        self._report_view.show()

        # 2. Se inicializa la vista
        self._report_view.show()

        # 3. Se inicializa la vista
        self._report_view.show()

        # 4. Se inicializa la vista
        self._report_view.show()

        # 5. Se inicializa la vista
        self._report_view.show()

        # 6. Se inicializa la vista
        self._report_view.show()

    def _initialize_controller(self):
        """
        Inicializa el controlador.
        """
        # 1. Se inicializa el controlador
        self._report_model = ReportModel()

        # 2. Se inicializa el controlador
        self._report_model = ReportModel()

        # 3. Se inicializa el controlador
        self._report_model = ReportModel()

        # 4. Se inicializa el controlador
        self._report_model = ReportModel()

        # 5. Se inicializa el controlador
        self._report_model = ReportModel()

        # 6. Se inicializa el controlador
        self._report_model = ReportModel()

```

Dicho método se conecta con la vista mediante una señal, respectivamente:

```

class ReportController:
    """
    Controlador para gestionar la interacción entre modelo, vista y base de datos.
    """
    def __init__(self, report_view: ReportView, report_model: ReportModel, popup_parent: Optional[QWidget] = None):
        """
        Inicializa el controlador.
        """
        # Parámetros:
        # - report_view (ReportView): Vista para la interfaz de usuario.
        # - report_model (ReportModel): Modelo de datos.
        # - popup_parent (QWidget | None): Padre opcional para los popups.
        if not report_view or not report_model:
            raise ValueError("Se requieren tanto la vista como el modelo para inicializar el controlador.")

        self._report_view = report_view
        self._report_model = report_model
        self._popup_parent = popup_parent

        # Conectar señales
        self._report_view.apply_filters_signal.connect(self._apply_filters)

        # Inicializar vista
        self._initialize_view()

```

Posteriormente, en la vista, también nos encontramos con su señal correspondiente, y con un método que nos permite aplicar los filtros para que todo funcione:

```

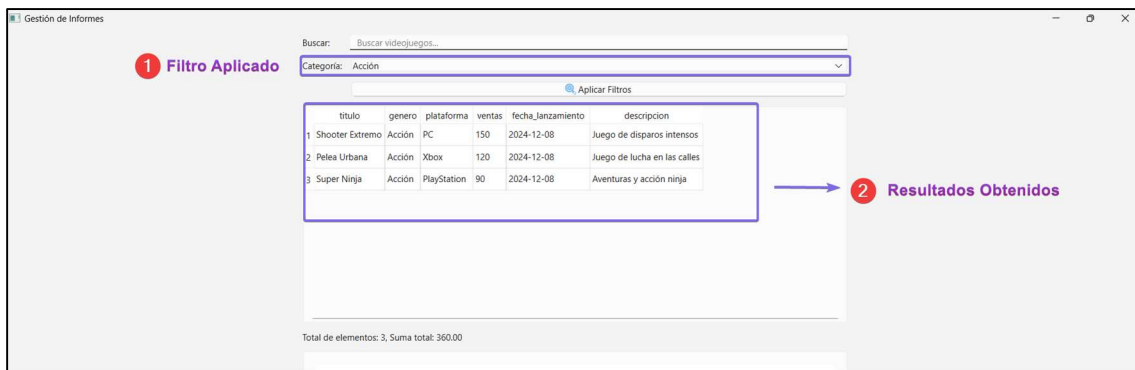
# Conectar botón a la señal de filtros
self.apply_filter_button.clicked.connect(self._emit_apply_filters_signal)

```

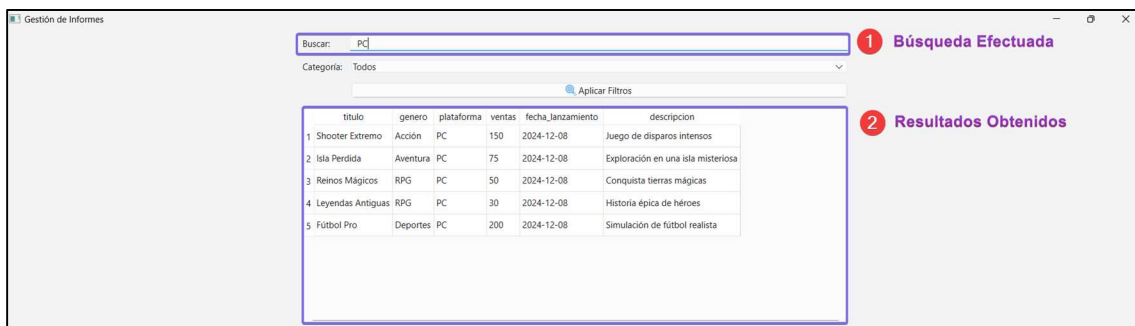
```

@Slot()
def _emit_apply_filters_signal(self):
    """
    Emite la señal de aplicar filtros con los valores actuales de búsqueda y categoría.
    """
    search_text = self.search_input.text()
    category = self.category_select.currentText()
    self.apply_filters_signal.emit(search_text, category)

```



De la misma forma, se puede escribir dentro de la barra de búsqueda, y en función del input se irán obteniendo los resultados que coincidan, de forma independiente a la columna de la que se trate:



3.3.- Arreglo Rango Inválido en la Gráfica:

Al iniciar la aplicación, se estaba obteniendo el siguiente fallo tanto al intentar aplicar cualquier filtro o intentar usar la barra de búsqueda:



Para solucionarlo, se ha tenido que modificar la clase Custom Chart Widget, concretamente los métodos **set data y configure axes**:


```

42
43
44 def _set_data(self, data: Dict[str, Any]) -> None:
45     """
46     Configura el gráfico con los datos proporcionados.
47     """
48     eje_x: List[str] = data.get(fnumEjes.EJE_X_value, [])
49     barras_datos: Dict[str, List[Any]] = data.get(fnumEjes.EJE_Y_value, [])
50
51     self._set_empty_chart()
52     self._chart.remove_all_series()
53     self._chart.setTitle("Gráfico de ventas por Videojuego")
54
55     if not eje_x or not any(barras_datos.values()):
56         self._set_empty_chart()
57         return
58
59     bar_series = QBarSeries()
60     for name, values in barras_datos.items():
61         cleaned_values = [
62             value if value is not None and np.isfinite(value) else 0
63             for value in values
64         ]
65
66         if len(cleaned_values) != len(eje_x):
67             print(f"Advertencia: El tamaño de los datos de '{name}' no coincide con las categorías del eje X.")
68             continue
69
70         bar_set = QBarSet(name)
71         bar_set.append(cleaned_values)
72         bar_set.hovered.connect(lambda status, index, bar_set: self._show_tooltip(status, index, bar_set))
73         bar_series.append(bar_set)
74
75     if not bar_series.barsets():
76         self._set_empty_chart()
77         return
78
79     self._chart.addSeries(bar_series)
80     self._configure_axes(eje_x, bar_series)

```

Método Set Data

```

81
82 def _configure_axes(self, eje_x: List[str], bar_series: QBarSeries) -> None:
83     """
84     Configura los ejes del gráfico.
85     """
86     axis_x = QBarCategoryAxis()
87     axis_x.append(eje_x)
88     axis_x.setLabelsAngle(-90)
89     self._chart.setAxisX(axis_x, bar_series)
90
91     all_values = [
92         bar_set.at(i)
93         for bar_set in bar_series.barsets()
94         for i in range(bar_set.count())
95     ]
96     finite_values = [v for v in all_values if np.isfinite(v)]
97
98     if finite_values:
99         min_y = min(finite_values)
100         max_y = max(finite_values)
101     else:
102         print("Advertencia: No hay valores válidos para el eje Y. Configurando rango predeterminado.")
103         min_y, max_y = 0, 1
104
105     axis_y = QValueAxis()
106     axis_y.setRange(min_y, max_y)
107     axis_y.setTitleText("Valores")
108     self._chart.setAxisY(axis_y, bar_series)
109
110     self._chart_view.setMinimumWidth(len(eje_x) * 180)
111

```

Método Configure Axes

Una vez realizados estos cambios, la aplicación ha solventado ese error, permitiendo escribir y filtrar sin ningún tipo de error en el formato:

Buscar:

Categoría: Acción

	titulo	genero	plataforma	ventas	fecha_lanzamiento	des
1	Shooter Extremo	Acción	PC	150	2024-12-08	Juego de di
2	Pelea Urbana	Acción	Xbox	120	2024-12-08	Juego de lu
3	Super Ninja	Acción	PlayStation	90	2024-12-08	Aventuras y

Total de elementos: 3, Suma total: 360.00

Gráfico de ventas por Videojuego

PDF Tabla

PDF Gráfico

Resultados Obtenidos
sin problemas de formato

3.4.- Modificación Label:

La vista principal nos traía incorporada ya una Label, sin embargo, ésta no poseía funcionalidad. En nuestro caso concreto, para que funcione, hemos implementado ciertos cambios dentro de la vista y del controlador, de tal forma que cuando filtras te mostrará el total de productos y el importe total de los mismos:

Cambios realizados en la Vista → Métodos `init summary` y `update summary`:

```
235
236
237 def _init_summary(self):
238     """
239     Configura el resumen de datos, incluyendo el total de elementos y la suma total.
240     """
241     self.summary_label = QLabel("Total de elementos: 0, Suma total: 0")
242     self.summary_label.setAlignment(Qt.AlignLeft)
243
244 def _update_summary(self, total_products: int, total_price: float):
245     """
246     Actualiza la etiqueta del resumen con la cantidad total de productos y la suma total de precios.
247
248     Parámetros:
249     - total_products (int): Número total de productos.
250     - total_price (float): Suma total de los precios de los productos.
251     """
252     self.summary_label.setText(f"Total de elementos: {total_products}, Suma total: {total_price:.2f}")
```

1 Init Summary

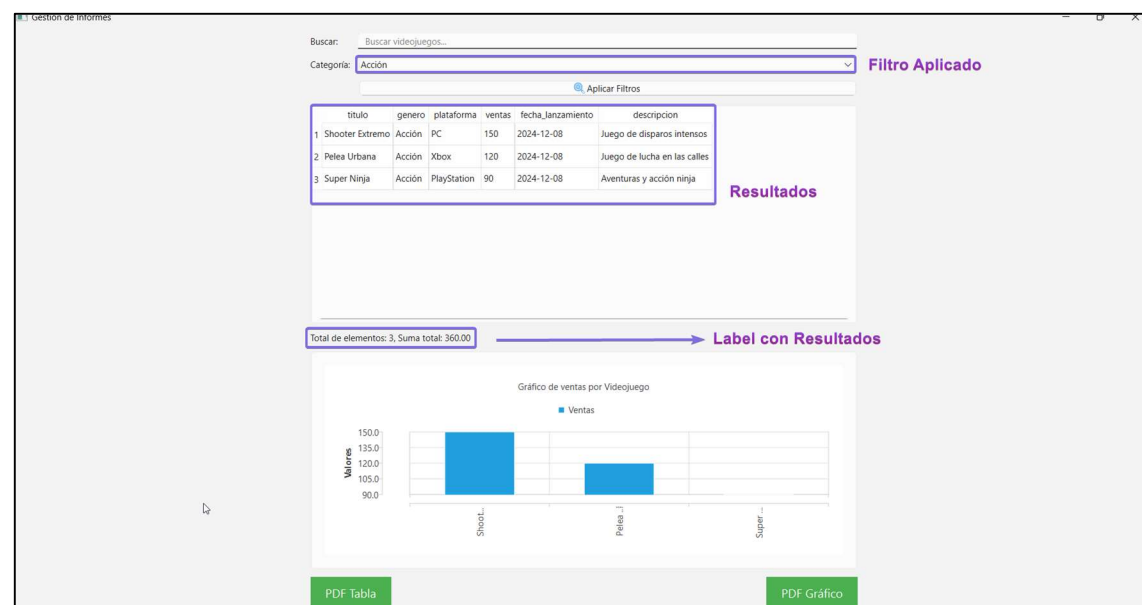
2 Update Summary

Posteriormente, `update summary` será llamado en el controlador:

```
# Actualizar resumen en la vista
# No hay precios en VideojuegoEntity
self.view._update_summary(total_games, total_ventas)
```

Llamada en el controlador

Con esta implementación, al realizar cualquier búsqueda o aplicar nuestro filtro, obtendremos el resultado deseado:



4.- Bibliografía Empleada:

Python Tutorial. (s. f.). W3schools.com. Recuperado 8 de diciembre de 2024, de <https://www.w3schools.com/python/>

Qt for Python. (s. f.). Doc.qt.io. Recuperado 8 de diciembre de 2024, de <https://doc.qt.io/qtforpython-6/>

Stack Overflow - Where Developers Learn, Share, & Build Careers. (s. f.). Stack Overflow. Recuperado 8 de diciembre de 2024, de <https://stackoverflow.com/>