

GameSide

Implementa un proyecto web Django para **venta online de videojuegos**.

1. Puesta en marcha

Lleva a cabo los siguientes comandos para la puesta en marcha del proyecto:

```
just create-venv
source .venv/bin/activate
just setup
```

¿Qué ha ocurrido?

- Se ha creado un entorno virtual en la carpeta `.venv`
- Se han instalado las dependencias del proyecto.
- Se ha creado un proyecto Django en la carpeta `main`
- Se han aplicado las migraciones iniciales del proyecto.
- Se ha creado un superusuario con credenciales: `admin - admin`

2. Aplicaciones

Habrás que añadir las siguientes aplicaciones:

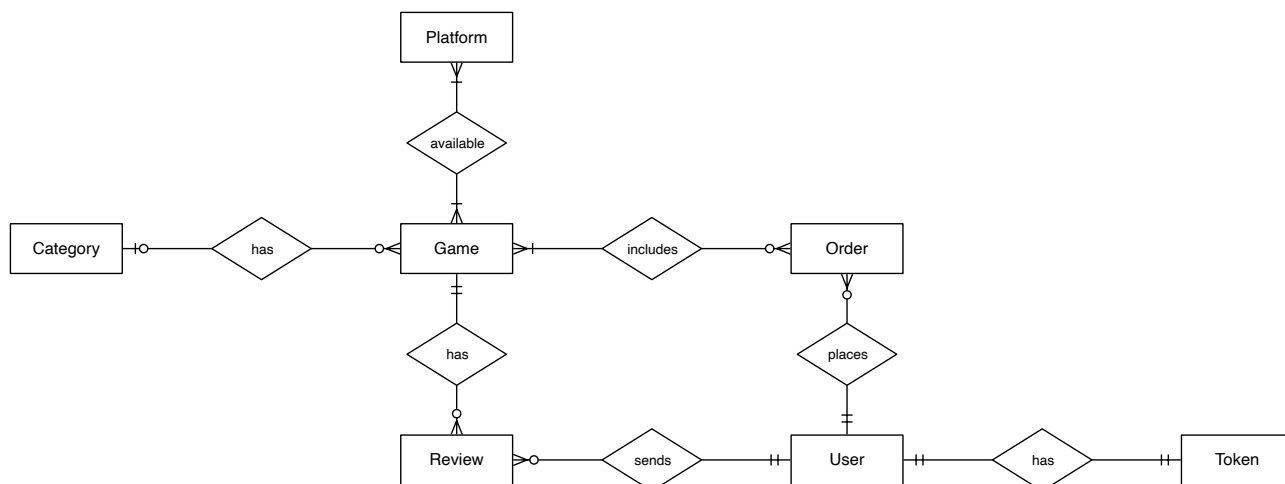
<code>shared</code>	Artefactos compartidos.
<code>games</code>	Gestión de juegos.
<code>platforms</code>	Gestión de plataformas.
<code>categories</code>	Gestión de categorías.
<code>orders</code>	Gestión de pedidos.
<code>users</code>	Gestión de usuarios.

Se proporciona una *receta* `just` para añadir una aplicación:

```
just startapp <app>
```

Esta receta no sólo crea la carpeta de la aplicación sino que añade la línea correspondiente de configuración en la variable `INSTALLED_APPS` del fichero `settings.py`.

3. Modelos



3.1. games.Game

Modelo que representa un videojuego.

Campo	Tipo
<code>title^(*)</code>	<code>str</code>
<code>slug^(*)</code>	<code>str</code>
<code>description^(∅)</code>	<code>str</code>
<code>cover^(∅Δ)</code>	<code>image</code>
<code>price^(*)</code>	<code>float</code>
<code>stock^(*)</code>	<code>int</code>
<code>released_at^(*)</code>	<code>date</code>
<code>pegi^(*)</code>	<code>enum (int)</code>
<code>category^(∅)</code>	<code>fk → Category</code>
<code>platforms^(*)</code>	<code>m2m → Platform</code>

- El **PEGI** (pegi) será un **enumerado entero** con valores:

- PEGI3 = 3
- PEGI7 = 7
- PEGI12 = 12
- PEGI16 = 16
- PEGI18 = 18

- Aplica `models.SET_NULL` en la clave ajena con `Category`.
- El valor por defecto `cover(Δ)` debe ser `covers/default.jpg`

3.2. games.Review

Modelo que representa una reseña de un usuario sobre un juego.

Campo	Tipo
<code>rating^(*)</code>	<code>int</code>
<code>comment^(*)</code>	<code>str</code>
<code>game^(*)</code>	<code>fk → Game</code>
<code>author^(*)</code>	<code>fk → User</code>
<code>created_at^(*)</code>	<code>datetime</code>
<code>updated_at^(*)</code>	<code>datetime</code>

- Aplica validadores para el campo `rating` que comprueben que el valor esté en el rango `[1, 5]`.

3.3. categories.Category

Modelo que representa una categoría de videojuegos: *Aventura, Acción, Estrategia, etc.*

Campo	Tipo
name ^(*u)	str
slug ^(*u)	str
description ^(∅)	str
color ^(∅Δ)	ColorField

- El valor por defecto color^(Δ) debe ser #ffffff

3.4. platforms.Platform

Modelo que representa una plataforma de videojuegos: *PC, PS4, Xbox, etc.*

Campo	Tipo
name ^(*u)	str
slug ^(*u)	str
description ^(∅)	str
logo ^(∅Δ)	image

- El valor por defecto logo^(Δ) debe ser logos/default.jpg

3.5. orders.Order

Modelo que representa un pedido de un usuario.

Campo	Tipo
status ^(*Δ)	enum (int)
key ^(*uΔ)	UUID
user ^(*)	fk → User
games ^(∅)	m2m → Game
created_at ^(*)	datetime
updated_at ^(*)	datetime

- El estado de un pedido (status) será un **enumerado entero** con valores:
 - INITIATED = 1 (por defecto)
 - CONFIRMED = 2
 - PAID = 3
 - CANCELLED = -1
- El valor por defecto key^(Δ) debe ser uuid.uuid4

⊛ Implementa una **@property** llamada price que devuelva el **precio total del pedido** en función de los juegos añadidos.

3.6. users.Token

Modelo que representa un token de autenticación de un usuario.

user ^(*)	o2o → User
key ^(*uΔ)	UUID
created_at ^(*)	datetime

- El valor por defecto key^(Δ) debe ser uuid.uuid4

3.7. User

No hay que implementar este modelo. Se usará el modelo **User** que ofrece Django.

3.8. Carga de datos

Una vez que hayas creado los modelos y aplicado las migraciones, puedes cargar datos de prueba con la siguiente *receta* [just](#):

```
just load-data
```

4. URLs

Dado que estamos implementando una **API** prácticamente todas las URLs devolverán una respuesta en formato **JSON**.

4.1. games.urls

</api/games/> ⇒ `games.views.game_list()`

Listado de los juegos disponibles en el sistema.

GET request	JSON response (200)
	<code>game^(🕹)</code> <code>game^(🕹)</code> <code>game^(🕹)</code> ...

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed

</api/games/?category=sports&platform=ps5> ⇒ `games.views.game_list()`

Listado de los juegos disponibles en el sistema filtrando por los parámetros de la petición *querystring*.

GET request	JSON response (200)
<code>category^(🕹)</code> <code>platform^(🕹)</code>	<code>game^(🕹)</code> <code>game^(🕹)</code> <code>game^(🕹)</code> ...

- `category` y `platform` son parámetros de la petición *querystring*.
- Se puede filtrar por uno, por otro o por ambos.

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed

`/api/games/eldenring/` ⇒ `games.views.game_detail()`

Detalle del juego “Elden Ring”.

GET request	JSON response (200)
	<pre>id title slug description cover price stock released_at pegi category^(v) platforms^(v)</pre>

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
404	Game not found

`/api/games/eldenring/reviews/` ⇒ `games.views.review_list()`

Reseñas del juego “Elden Ring”.

GET request	JSON response (200)
	<pre>review^(v) review^(v) review^(v) ...</pre>

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
404	Game not found

`/api/games/reviews/21/` ⇒ `games.views.review_detail()`

Detalle de la reseña con pk=21.

GET request	JSON response (200)
	rating comment game ^(v) author ^(v) created_at updated_at

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
404	Review not found

`/api/games/eldenring/reviews/add/` \Rightarrow `games.views.add_review()`

Añade una nueva reseña al juego “Elden Ring”.

Headers	POST request	JSON response (200)
token ^(bearer)	rating comment	id ^(pk-review)

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
400	Invalid JSON body
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
404	Game not found
400	Rating is out of range

4.2. categories.urls

`/api/categories/` \Rightarrow `categories.views.category_list()`

Listado de las categorías disponibles.

GET request	JSON response (200)
	category ^(v) category ^(v) category ^(v) ...

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los

siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed

`/api/categories/sports/` \Rightarrow `categories.views.category_detail()`

Detalle de la categoría *Deportes*.

GET request	JSON response (200)
	<code>id^(pk-category)</code> <code>name</code> <code>slug</code> <code>description</code> <code>color</code>

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
404	Category not found

4.3. platforms.urls

`/api/platforms/` \Rightarrow `categories.views.platform_list()`

Listado de las plataformas disponibles.

GET request	JSON response (200)
	<code>platform^(Ⓢ)</code> <code>platform^(Ⓢ)</code> <code>platform^(Ⓢ)</code> <code>...</code>

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed

`/api/platforms/ps5/` \Rightarrow `categories.views.platform_detail()`

Detalle de la plataforma *PlayStation 5*.

GET request	JSON response (200)
	<code>id^(pk-platform)</code> <code>name</code> <code>slug</code> <code>description</code> <code>logo</code>

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
404	Platform not found

4.4. orders.urls

</api/orders/add/> ⇒ `orders.views.add_order()`

Añade un nuevo pedido (vacío).

Headers	POST request	JSON response (200)
<code>token^(bearer)</code>		<code>id^(pk-order)</code>

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP Status	Error
405	Method not allowed
400	Invalid authentication token
401	Unregistered authentication token

</api/orders/53/> ⇒ `orders.views.order_detail()`

Detalle del pedido con `pk=53`.

Headers	GET request	JSON response (200)
<code>token^(bearer)</code>		<code>id</code> <code>status</code> <code>key^(!)</code> <code>games^(♡)</code> <code>created_at</code> <code>updated_at</code> <code>price</code>

- `key=None` si el pedido **no** está en estado PAID.

Devuelve una respuesta JSON con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP <i>Status</i>	Error
405	Method not allowed
400	Invalid authentication token
401	Unregistered authentication token
404	Order not found
403	User is not the owner of requested order

[/api/orders/53/games/](#) ⇒ `orders.views.order_game_list()`

Listado con los juegos del pedido con `pk=53`.

Headers	GET <i>request</i>	JSON <i>response</i> (200)
<code>token^(bearer)</code>		<code>game^(⌚)</code> <code>game^(⌚)</code> <code>game^(⌚)</code> ...

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP <i>Status</i>	Error
405	Method not allowed
400	Invalid authentication token
401	Unregistered authentication token
404	Order not found
403	User is not the owner of requested order

[/api/orders/53/games/add/](#) ⇒ `orders.views.add_game_to_order()`

Añade un juego al pedido con `pk=53`.

Headers	POST <i>request</i>	JSON <i>response</i> (200)
<code>token^(bearer)</code>	<code>game-slug</code>	<code>num-games-in-order</code>

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP <i>Status</i>	Error
405	Method not allowed
400	Invalid JSON body
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
404	Order not found
404	Game not found
403	User is not the owner of requested order
400	Game out of stock

⊛ Al **añadir un juego al pedido** se debe actualizar el stock del juego añadido.

`/api/orders/53/status/` \Rightarrow `orders.views.change_order_status()`

Confirmación/cancelación del pedido con `pk=53`.

Headers	POST <i>request</i>	JSON <i>response</i> (200)
<code>token</code> ^(bearer)	<code>status</code> ^(value)	<code>status</code> ^(label)

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP <i>Status</i>	Error
405	Method not allowed
400	Invalid JSON body
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
404	Order not found
403	User is not the owner of requested order
400	Invalid status
400	Orders can only be confirmed/cancelled when initiated

⊗ Al **cancelar un pedido** se debe actualizar el stock de los juegos añadidos al pedido.

`/api/orders/53/pay/` \Rightarrow `orders.views.pay_order()`

Pago del pedido con `pk=53`.

Headers	POST <i>request</i>	JSON <i>response</i> (200)
<code>token</code> ^(bearer)	<code>card-number</code> <code>exp-date</code> <code>cvc</code>	<code>status</code> <code>key</code>

- `card-number` \Rightarrow Número de tarjeta de crédito en formato DDDD-DDDD-DDDD-DDDD (*dígitos*).
- `exp-date` \Rightarrow Fecha de caducidad de la tarjeta de crédito en formato MM/YYYY (*mes/año*).
- `cvc` \Rightarrow Código de verificación de la tarjeta de crédito en formato DDD (*dígitos*).

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP <i>Status</i>	Error
400	Invalid JSON body
400	Invalid authentication token
400	Missing required fields
400	Orders can only be paid when confirmed
400	Invalid card number
400	Invalid expiration date
400	Invalid CVC
400	Card expired
401	Unregistered authentication token
403	User is not the owner of requested order
404	Order not found
405	Method not allowed

4.5. `users.urls`

`/api/auth/` \Rightarrow `users.views.auth()`

Autenticar credenciales de usuario.

POST <i>request</i>	JSON <i>response</i> (200)
username	token
password	

Devuelve una respuesta **JSON** con una clave **error** y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP <i>Status</i>	Error
400	Invalid JSON body
400	Missing required fields
401	Invalid credentials
405	Method not allowed

5. Serializadores

Además de los serializadores indicados en el apartado anterior en cada vista de detalle, habrá que serializar (para los correspondientes casos) el modelo **User** con los siguientes campos:

- `id` (*pk-user*)
- `username`
- `first_name`
- `last_name`
- `email`

6. Administración

Los siguientes modelos deben estar accesibles desde la **interfaz administrativa** de Django:

- `games.Game`
- `games.Review`
- `categories.Category`
- `platforms.Platform`
- `orders.Order`
- `users.Token`