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

Habrá que añadir las siguientes aplicaciones:

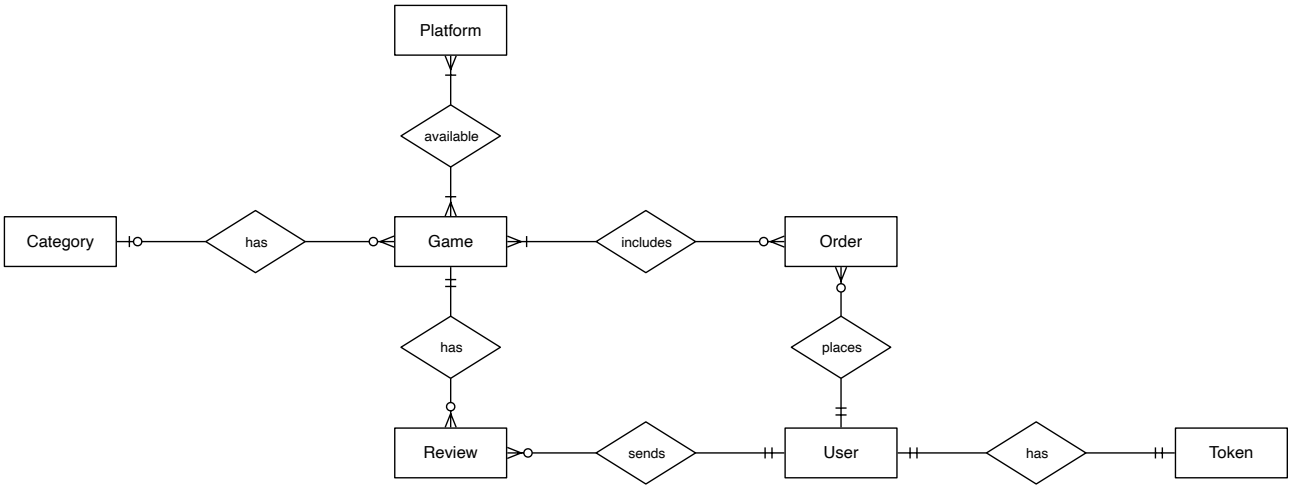
`shared`	Artefactos compartidos.
`games`	Gestión de juegos.
`platforms`	Gestión de plataformas.
`categories`	Gestión de categorías.
`orders`	Gestión de pedidos.
`users`	Gestión de usuarios.

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
`title`$^{(*u)}$	*str*
`slug`$^{(*u)}$	*str*
`description`$^{(\varnothing)}$	*str*
`cover`$^{(\varnothing\Delta)}$	*image*
`price`$^{(*)}$	*float*
`stock`$^{(*)}$	*int*
`released_at`$^{(*)}$	*date*
`pegi`$^{(*)}$	*enum (int)*
`category`$^{(\varnothing)}$	*fk* $\rightarrow$ `Category`
`platforms`$^{(*)}$	*m2m* $\rightarrow$ `Platform`

- 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`.

3.2. `games.Review`

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

Campo	Tipo
`rating`$^{(*)}$	*int*
`comment`$^{(*)}$	*str*
`game`$^{(*)}$	*fk* $\rightarrow$ `Game`
`author`$^{(*)}$	*fk* $\rightarrow$ `User`
`created_at`$^{(*)}$	*datetime*
`updated_at`$^{(*)}$	*datetime*

- 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`$^{(\varnothing)}$	*str*
`color`$^{(\varnothing\Delta)}$	`ColorField`

3.4. `platforms.Platform`

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

Campo	Tipo
`name`$^{(*u)}$	*str*
`slug`$^{(*u)}$	*str*
`description`$^{(\varnothing)}$	*str*
`logo`$^{(\varnothing\Delta)}$	*image*

3.5. `orders.Order`

Modelo que representa un pedido de un usuario.

Campo	Tipo
`status`$^{(*\Delta)}$	*enum (int)*
`key`$^{(*u\Delta)}$	*UUID*
`user`$^{(*)}$	*fk* $\rightarrow$ `User`
`games`$^{(\varnothing)}$	*m2m* $\rightarrow$ `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`

3.6. `users.Token`

Modelo que representa un token de autenticación de un usuario.

`user`$^{(*)}$	*o2o* $\rightarrow$ `User`
`key`$^{(*u)}$	*UUID*
`created_at`$^{(*)}$	*datetime*

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)
	`game`(↻) `game`(↻) `game`(↻) ...

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
405	

`/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)
`category`(↻) `platform`(↻)	`game`(↻) `game`(↻) `game`(↻) ...

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
405	

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

Detalle del juego “Elden Ring”.

`GET` *request*	`JSON` *response* (200)

	`id`
	`title`
	`slug`
	`description`
	`cover`
	`price`
	`stock`
	`released_at`
	`pegi`
	`category`(⟳)
	`platforms`(⟳)

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
404	Game not found
405	

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

Reseñas del juego "Elden Ring".

`GET` *request*	`JSON` *response* (200)
	`review`(⟳) `review`(⟳) `review`(⟳) ...

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
404	Game not found
405	

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

Detalle de la reseña con `pk=21`.

`GET` *request*	`JSON` *response* (200)

	`rating`
	`comment`
	`game`(⟳)
	`author`(⟳)
	`created_at`
	`updated_at`

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
404	Review not found
405	

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

Añade una nueva reseña al juego "Elden Ring".

POST *request*	JSON *response* (200)
`token`	`id`$^{(pk-review)}$
`rating`	
`comment`	

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
400	Invalid JSON body
400	Rating must be between 1 and 5
401	Invalid token
404	Game not found
405	

4.2. `categories.urls`

`/api/categories/` ⇒ `categories.views.category_list()`

Listado de las categorías disponibles.

GET *request*	JSON *response* (200)
	`category`$^{(\circlearrowleft)}$
	`category`$^{(\circlearrowleft)}$
	`category`$^{(\circlearrowleft)}$
	...

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
405	

`/api/categories/sports/` ⇒ `categories.views.category_detail()`

Detalle de la categoría *Deportes*.

GET *request*	JSON *response* (200)
	id$^{(pk-category)}$ name slug description color

Devuelve una respuesta JSON con una clave error (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
404 405	Category not found

4.3. platforms.urls

/api/platforms/ ⇒ categories.views.platform_list()

Listado de las plataformas disponibles.

GET *request*	JSON *response* (200)
	platform$^{(\circlearrowleft)}$ platform$^{(\circlearrowleft)}$ platform$^{(\circlearrowleft)}$ ...

Devuelve una respuesta JSON con una clave error (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
405	

/api/platforms/ps5/ ⇒ categories.views.platform_detail()

Detalle de la plataforma *PlayStation 5*.

GET *request*	JSON *response* (200)
	id$^{(pk-platform)}$ name slug description logo

Devuelve una respuesta JSON con una clave error (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
404 405	Platform not found

4.4. orders.urls

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

Añade un nuevo pedido (vacío).

POST *request*	JSON *response* (200)
token	id$^{(pk-order)}$

Devuelve una respuesta JSON con una clave error (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
400	Invalid JSON body
400	Missing required fields
401	Invalid token
405	

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

Detalle del pedido con pk=53.

POST *request*	JSON *response* (200)
token	status key$^{(!)}$ games$^{(↻)}$ created_at updated_at price

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

Devuelve una respuesta JSON con una clave error (si procede) atendiendo a los siguientes casos:

HTTP *Status*	Error
400	Invalid JSON body
400	Missing required fields
401	Invalid token
403	User is not the owner of requested order
404	Order not found
405	

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

Listado con los juegos del pedido con pk=53.

POST *request*	JSON *response* (200)
token	game$^{(↻)}$ game$^{(↻)}$ game$^{(↻)}$ ...

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
401	Invalid token
403	User is not the owner of requested order
404	Order not found
405	

`/api/orders/53/games/add/eldenring/` ⇒ `orders.views.add_game_to_order()`

Añade el juego *Elden Ring* al pedido con `pk=53`.

`POST` *request*	`JSON` *response* (200)
`token`	`num-games-in-order`

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
400	Game out of stock
401	Invalid token
403	User is not the owner of requested order
404	Order not found
404	Game not found
405	

⊛ Se debe actualizar el stock del juego añadido.

`/api/orders/53/confirm/` ⇒ `orders.views.confirm_order()`

Confirmación del pedido con `pk=53`.

`POST` *request*	`JSON` *response* (200)
`token`	`status`

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
400	Orders can only be confirmed when initiated
401	Invalid token
403	User is not the owner of requested order
404	Order not found
405	

`/api/orders/53/cancel/` ⇒ `orders.views.cancel_order()`

Cancelación del pedido con `pk=53`.

`POST` *request*	`JSON` *response* (200)
`token`	`status`

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
400	Orders can only be cancelled when initiated
401	Invalid token
403	User is not the owner of requested order
404	Order not found
405	

⊛ Se debe actualizar el stock de los juegos añadidos al pedido.

`/api/orders/53/pay/` ⇒ `orders.views.pay_order()`

Pago del pedido con `pk=53`.

`POST` *request*	`JSON` *response* (200)
`token`	`status`
`card-number`	`key`
`cvc`	

- `card-number` ⇒ Número de tarjeta de crédito en formato `XXXX-XXXX-XXXX-XXXX`.
- `cvc` ⇒ Código de verificación de la tarjeta de crédito en formato `XXX`.

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
400	Orders can only be payed when confirmed
400	Invalid card number
400	Invalid CVC
401	Invalid token
403	User is not the owner of requested order
404	Order not found
405	

4.5. `users.urls`

`/api/auth/` ⇒ `users.views.auth()`

Autenticar credenciales de usuario.

`POST` *request*	`JSON` *response* (200)
`username` `password`	`token`

Devuelve una respuesta `JSON` con una clave `error` (si procede) atendiendo a los siguientes casos:

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
401	Invalid credentials
405	

5. 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`

