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:

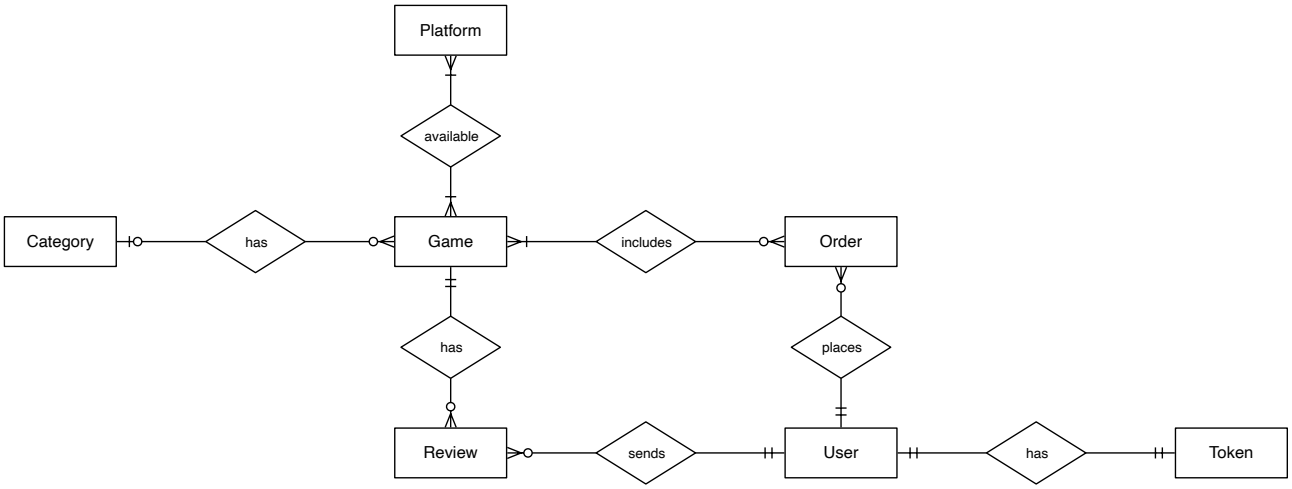
`shared`	Artefactos compartidos.
`games`	Gestión de juegos.
`platforms`	Gestión de plataformas.
`categories`	Gestión de categorías.
`orders`	Gestión de pedidos.
`users`	Gestión de usuarios.

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`$^{(\varnothing)}$	*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`
`user`$^{(*)}$	*fk* $\rightarrow$ `User`

- Aplica validadores para el campo `rating` que comprueben que el valor esté en el rango $[0, 5]$.

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`$^{(\varnothing)}$	`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`$^{(\varnothing)}$	*image*

3.5. `orders.Order`

Modelo que representa un pedido de un usuario.

Campo	Tipo
`status`$^{(*)}$	*enum (int)*
`created_at`$^{(*)}$	*datetime*
`updated_at`$^{(*)}$	*datetime*
`key`$^{(\varnothing)}$	*UUID*
`user`$^{(*)}$	*fk* → `User`
`games`$^{(*)}$	*m2m* → `Game`

- El estado de un pedido (`status`) será un enumerado **entero** con valores:
  - `INITIATED = 0`
  - `CONFIRMED = 1`
  - `CANCELLED = 2`
  - `PAID = 3`

3.6. `users.Token`

Modelo que representa un token de autenticación de un usuario.

`user`$^{(*)}$	*o2o* → `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 un `JSON`. Para poder hacer pruebas de forma sencilla, puedes utilizar la herramienta HoppScotch (gratuita y de código abierto).

4.1. `games.urls`

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

Listado de los juegos disponibles en el sistema.

GET *request*	JSON *response*
	`game`(↻) `game`(↻) `game`(↻) ...

`/api/games/filter?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*
`category`(↻) `platform`(↻)	`game`(↻) `game`(↻) `game`(↻) ...

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

Detalle del juego "Elden Ring".

GET *request*	JSON *response*

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

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

Reseñas del juego “Elden Ring”.

GET *request*	JSON *response*
	`review`(⏲) `review`(⏲) `review`(⏲) ...

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

Detalle de la reseña con `pk=21`.

GET *request*	JSON *response*

	`comment`
	`rating`
	`game`(⏲)
	`user`(⏲)

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

Añade una nueva reseña al juego “Elden Ring”.

POST *request*	JSON *response*

`token`
`comment`
`rating`
	`pk`(*review*)

4.2. `categories.urls`

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

Listado de las categorías disponibles.

GET *request*	JSON *response*
	`category`(⏲) `category`(⏲) `category`(⏲) ...

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

Detalle de la categoría *Deportes*.

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

4.3. `platforms.urls`

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

Listado de las plataformas disponibles.

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

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

Detalle de la plataforma *PlayStation 5*.

GET *request*	JSON *response*
	pk$^{(category)}$ name description logo

4.4. `orders.urls`

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

Añade un nuevo pedido.

POST *request*	JSON *response*
token	pk$^{(order)}$ status created_at

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

Detalle del pedido con `pk=53`.

POST *request*	JSON *response*

`token`	`status`
	`created_at`
	`updated_at`
	`key`
	`games`(⥁)

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

Confirmación del pedido con `pk=53`.

POST *request*	JSON *response*

`token`	`status`
	`created_at`
	`updated_at`

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

Cancelación del pedido con `pk=53`.

POST *request*	JSON *response*

`token`	`status`
	`created_at`
	`updated_at`

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

Pago del pedido con `pk=53`.

POST *request*	JSON *response*

`token`	`status`
	`created_at`
	`updated_at`
	`key`

4.5. `users.urls`

`/api/auth/` ⇒ `users.views.auth()`

Autenticar credenciales de usuario.

`POST` *request*	`JSON` *response*
`username` `password`	`token`

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`