GameSide

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

La idea es desarrollar una **API** que proporcione datos en formato JSON para que sea consumida por algún "frontend" del lado del cliente.

1. Requerimientos

Para que puedas trabajar con normalidad en este ejercicio debes tener instalado en tu máquina:

1. `uv` → Gestor de paquetería y proyectos Python.
2. `just` → Lanzador de comandos como recetas.

2. Puesta en marcha

Se proporciona una *receta* `just` para la puesta en marcha del proyecto:

```
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`
- Se ha establecido el *timezone* a `Atlantic/Canary` en `settings.py`
- Se han establecido las configuraciones *media* en `settings.py`

3. Aplicaciones

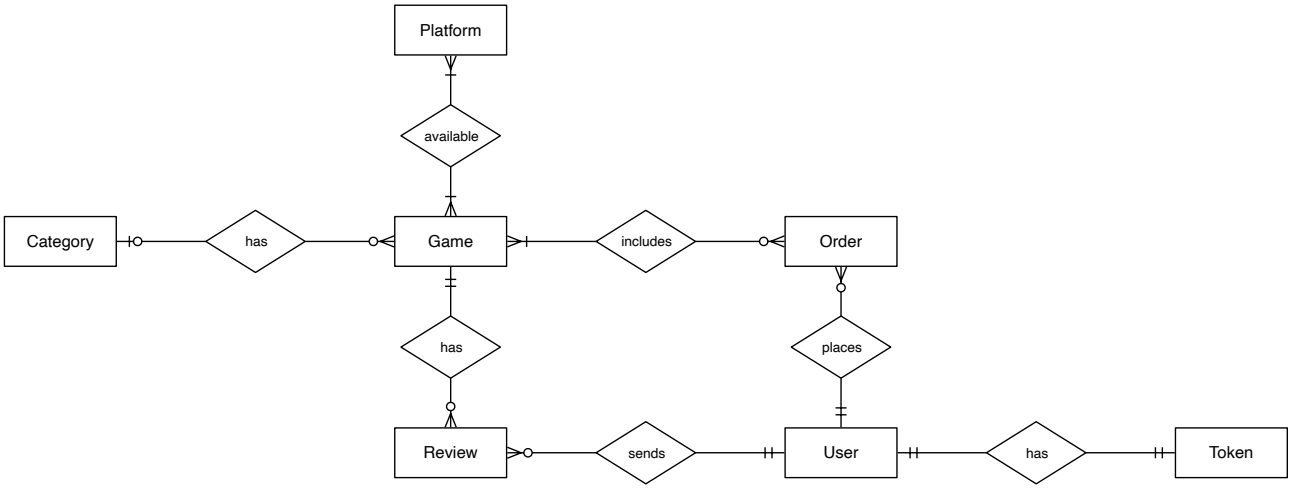
Habrá que añadir las siguientes aplicaciones:
Se proporciona una *receta* `just` para añadir una aplicación:

```
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`.

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

4. Modelos



4.1. `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`.
- El valor por defecto `cover`$^{(\Delta)}$ debe ser `covers/default.jpg`

4.2. games.Review

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

Campo	Tipo
rating$^{(*\ v_{min}=1,\ v_{max}=5)}$	*int*
comment$^{(*)}$	*str*
game$^{(*)}$	*fk* → Game
author$^{(*)}$	*fk* → User
created_at$^{(*)}$	*datetime*
updated_at$^{(*)}$	*datetime*

4.3. 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$^{(\varnothing\Delta=\#ffffff)}$	ColorField

4.4. 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$^{(\varnothing\Delta=platforms/logos/default.png)}$	*image*

4.5. orders.Order

Modelo que representa un pedido de un usuario.

Campo	Tipo
status$^{(*\Delta)}$	*enum (int)*
key$^{(*u\Delta=uuid.uuid4)}$	*UUID*
user$^{(*)}$	*fk* → User
games$^{(\varnothing)}$	*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`

- `key` representa la "clave de licencia" del pedido. Su valor por defecto debe ser `uuid.uuid4`
- Implementa una `@property` llamada `price` que devuelva el **precio total del pedido** en función de los juegos añadidos.

4.6. `users.Token`

Modelo que representa un token de autenticación de un usuario.

`user`$^{(*)}$	*o2o* → `User`
`key`$^{(*u\Delta=\texttt{uuid.uuid4})}$	*UUID*
`created_at`$^{(*)}$	*datetime*

- `key` representa la clave de autenticación del *token* y su valor por defecto debe ser `uuid.uuid4`

4.7. Carga de datos

Una vez que hayas **creado los modelos** asegúrate de que tu proyecto pasa los *tests core*. Se proporciona una *receta* `just` para ello:

```
just test tests/test_core.py
```

Ahora **crea y aplica las migraciones** y, si todo ha ido bien, procede a la carga de datos iniciales. Se proporciona una *receta* `just` para ello:

```
just load-data
```

Puedes consultar los usuarios creados en cualquier momento con la receta: `just show-users`

5. URLs

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

5.1. `games.urls`

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

Listado de los juegos disponibles en el sistema.

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

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

- `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)

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

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)
	`review`$^{(\circlearrowleft)}$ `review`$^{(\circlearrowleft)}$ `review`$^{(\circlearrowleft)}$ ...

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`$^{(\circlearrowleft)}$
	`author`$^{(\circlearrowleft)}$
	`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/` ⇒ `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

5.2. `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` 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/` ⇒ `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` 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

5.3. `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` 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/` ⇒ `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` 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

5.4. `orders.urls`

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

Añade un nuevo pedido (vacío).

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

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)

`token`$^{(bearer)}$		`id`
		`status`
		`key`$^{(!)}$
		`games`$^{(\circlearrowleft)}$
		`created_at`
		`updated_at`
		`price`

- `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` *Status*	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` *request*	`JSON` *response* (200)
`token`$^{(bearer)}$		`game`$^{(\circlearrowleft)}$ `game`$^{(\circlearrowleft)}$ `game`$^{(\circlearrowleft)}$ ...

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
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` *request*	`JSON` *response* (200)
`token`$^{(bearer)}$	`game-slug`	`num-games-in-order`

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	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/` ⇒ `orders.views.change_order_status()`

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

Headers	`POST` *request*	`JSON` *response* (200)
`token`$^{(bearer)}$	`status`$^{(value)}$	`status`$^{(label)}$

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` *Status*	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/` ⇒ `orders.views.pay_order()`

Pago del pedido con `pk=53`.

Headers	`POST` *request*	`JSON` *response* (200)
`token`$^{(bearer)}$	`card-number`	`status`
	`exp-date`	`key`
	`cvc`	

- `card-number` ⇒ Número de tarjeta de crédito en formato `DDDD-DDDD-DDDD-DDDD` (*dígitos*).
- `exp-date` ⇒ Fecha de caducidad de la tarjeta de crédito en formato `MM/YYYY` (*mes/año*).
- `cvc` ⇒ 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` *Status*	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

5.5. `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` y un *mensaje informativo* atendiendo a los siguientes casos (el orden de los errores es importante):

`HTTP` *Status*	Error
400	Invalid JSON body
400	Missing required fields
401	Invalid credentials
405	Method not allowed

6. 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`

7. 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`

