Pro Trade

Implementa un proyecto web Django para gestionar **traspasos de futbolistas en el mercado de fichajes**.

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:

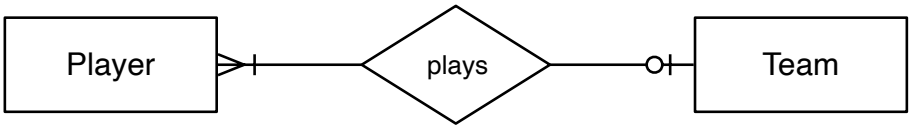
`shared`	Artefactos compartidos.
`players`	Gestión de jugadores.
`teams`	Gestión de equipos.
`users`	Gestión de usuarios.

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. `teams.Team`

Modelo que representa un equipo de fútbol.

Campo	Tipo
`name`$^{(*u)}$	*str*
`slug`$^{(*u)}$	*str*
`league`$^{(*)}$	*enum*
`shield`$^{(\varnothing\Delta)}$	*image*

- La liga (`league`) será un enumerado (*como cadena de texto*) con valores:
  - `LALIGA = 'L'`
  - `PREMIER = 'P'`
  - `CALCIO = 'C'`
  - `BUNDESLIGA = 'B'`
- El valor por defecto `shield`$^{(\Delta)}$ debe ser `teams/shields/default.png`

3.2. `players.Player`

Modelo que representa un/a jugador/a.

Campo	Tipo
`name`$^{(*u)}$	*str*
`slug`$^{(*u)}$	*str*
`position`$^{(*)}$	*enum*
`birth_date`$^{(*)}$	*date*
`market_value`$^{(*)}$	*float*
`photo`$^{(\varnothing\Delta)}$	*image*
`team`$^{(\varnothing)}$	*fk* $\rightarrow$ `Team`

- La posición (`position`) será un enumerado (*como cadena de texto*) con valores:
  - `GOALKEEPER = 'G'`
  - `DEFENDER = 'D'`
  - `MIDFIELDER = 'M'`
  - `FORWARD = 'F'`
- Utiliza `DecimalField` para el valor de mercado `market_value` (se expresará en *millones de euros*).
- El valor por defecto `photo`$^{(\Delta)}$ debe ser `players/photos/default.png`

3.3. `users.Token`

Modelo que representa un token de autenticación de un usuario.

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

- El valor por defecto `key`$^{(\Delta)}$ debe ser `uuid.uuid4`

3.4. `User`

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

3.5. 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. `players.urls`

`/api/players/` ⇒ `players.views.player_list()`

Listado de los/las jugadores/as disponibles en el sistema.

`GET` *request*	`JSON` *response* (200)
	`player`(↻) `player`(↻) `player`(↻) ...

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/players/?position=M&team=real-sociedad` ⇒ `players.views.player_list()`

Listado de los/las jugadores/as disponibles en el sistema filtrando por los parámetros de la petición *querystring*.

`GET` *request*	`JSON` *response* (200)
`position`(⏲) `team`(⏲)	`player`(⏲) `player`(⏲) `player`(⏲) ...

- `position` y `team` son parámetros de la petición *querystring*.
- El valor de `position` es el *código del enumerado*. Es decir `M` haría referencia a `MIDFIELDER`.
- El valor de `team` es el *slug*. Es decir `real-sociedad` haría referencia al equipo `Real Sociedad`.
- 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/players/lamine-yamal/` ⇒ `players.views.player_detail()`

Detalle del jugador "Lamine Yamal".

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

	`id`
	`name`
	`slug`
	`position`
	`birth_date`
	`market_value`
	`photo`
	`team`(⏲)

Serializador `Team`
`id`
`name`
`slug`
`league`
`shield`

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	Player not found

`/api/players/add/` ⇒ `players.views.add_player()`

Añade un/a nuevo/a jugador/a al sistema.

Headers	`POST` *request*	`JSON` *response* (200)

`token`$^{(bearer)}$	`name`
	`slug`
	`position`
	`birth-date`
	`market-value`
	`team-slug`
		`id`$^{(pk-player)}$

- `token` es el *token de autenticación* que se enviará en la cabecera de la petición.
- `position` se enviará como *valor* del enumerado.
- `birth-date` se enviará en formato `ISO 8601`. El método `fromisoformat()` te puede ayudar a convertirlo.
- `market-value` se enviará en *millones de euros*.

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
400	Invalid position
400	Invalid birth date
400	Team not found
400	Player already exists

`/api/players/transfer/` ⇒ `players.views.transfer_player()`

Realiza un traspaso de un/a jugador/a a otro equipo.

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

⊛ En el caso de que el/la jugador/a se traspase a **un equipo de otra liga**, el valor de mercado **se incrementará un 10 %**.

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
400	Player not found
400	Team not found

5. Administración

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

- `teams.Team`
- `players.Player`
- `users.Token`

