Manual Técnico de FutPro

Introducción

FutPro es una aplicación web diseñada para gestionar un mercado de jugadores de fútbol, permitiendo a los usuarios comprar, vender y gestionar sus equipos virtuales. Esta plataforma ofrece funcionalidades tanto para usuarios regulares como para administradores, permitiendo la gestión completa de jugadores, equipos, transacciones y monedas virtuales (FutCoins).

Herramientas Utilizadas:

- **Backend (Django)**
  - **Django REST Framework:** Para la creación de APIs robustas y seguras.
  - **PostgreSQL:** Base de datos relacional utilizada para almacenar toda la información de la aplicación.
  - **Docker:** Para la contener en un contenedor facilitando la configuración y despliegue en un contenedor sin la necesidad de tener que instalar nada en su máquina .
  - **Adminer:** Herramienta de gestión de bases de datos de manera simple e interactiva accesible a través de Docker.
- **Frontend (Angular)**
  - **Angular CLI:** Herramienta de línea de comandos para inicializar, desarrollar y mantener aplicaciones Angular.
  - **Angular Material:** Para la implementación de una interfaz de usuario atractiva y responsiva. Con muchas herramientas para formularios y modales.
  - **Tailwind CSS:** Un framework CSS para diseñar interfaces de usuario de manera eficiente y sencilla.

Instalación y despliegue:

Requisitos Previos

- Docker
- Docker Compose
- Git
- Node.js y npm
- Angular CLI

Si tienes todas estas herramientas instaladas en tu maquina, podrá ejecutar y lanzar mi aplicación, para simplificarlo sigue estos pasos.

Empecemos con la ejecución de la API

1. Primero clonamos el repositorio y nos descargará todo el proyecto:
    a. git clone https://github.com/Mohaek10/FutPro-Backend.git
2. Tras la descarga del repositorio podremos iniciar el docker mediante este comando, pero antes asegurate de que estas en el directorio que contiene el fichero docker-compose.yml
    a. Con el comando docker-compose up-d se lanza el docker y empieza a construir las imagenes y el contenedor
3. Tras la instalación de todo y la ejecución de los contenedores, debemos de hacer las migraciones a la bbdd, con este comando:
    a. Docker exec futpro-backend-web-1 bash –c “python manage.py makemigrations”
4. Tras las crear los archivos de migracion ya podemos migrar a la bbdd mediante este comando:
    a. Docker exec futpro-backend-web-1 bash –c “python manage.py migrate”
5. Tras las migraciones debemos de cargar los datos para tener algunos datos ya cargados, mediante este comando, muy importante seguirlos en orden:
    a. Docker exec futpro-backend-web-1 bash –c “python manage.py loaddata ./database/users.json”
    b. Docker exec futpro-backend-web-1 bash –c “python manage.py loaddata ./database/data.json”
    c. Docker exec futpro-backend-web-1 bash –c “python manage.py loaddata ./database/ventas.json”

Tras seguir estas instrucciones ya debe de estar en funcionamiento el servidor, en caso de que haya habidor un problema con la importación de los datos, puede usted crear un superuser que es lo único necesario para el funcionamiento de la api y que haya un usuario inicial.

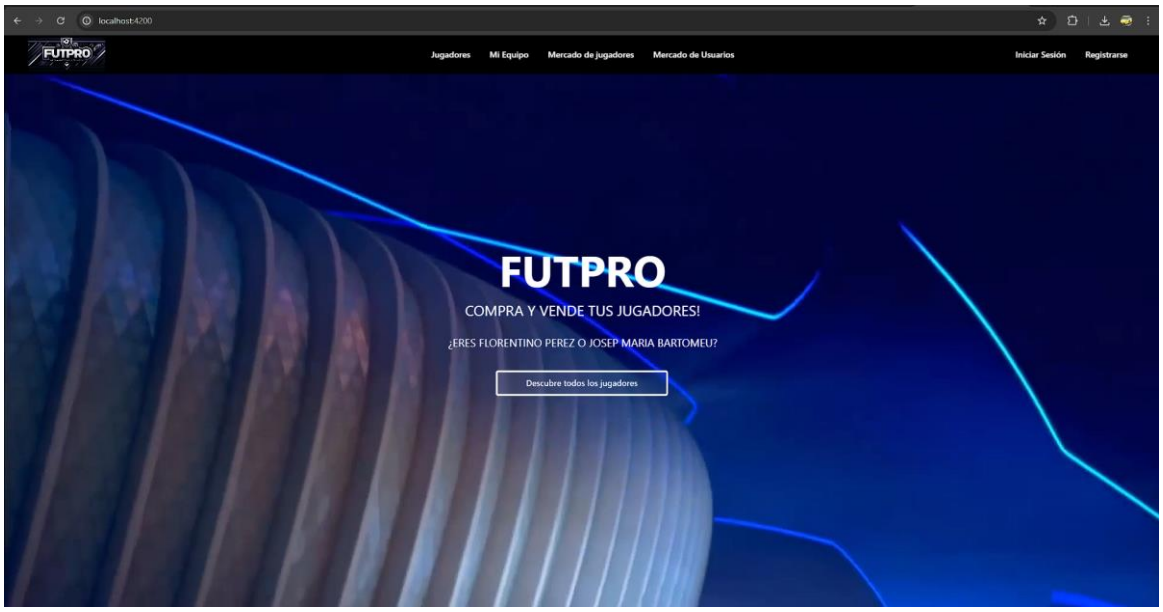
Para ello el comando es el siguiente: Docker exec futpro-backend-web-1 bash –c “python manage.py createsuperuser”

Tras ejecutar este comando se le pedirán algunos datos desde la consola.

Finalmente revise en el puerto localhost:8000/admin, si le carga el panel de django administration eso es que está todo funcionando correctamente.

Ejecución del frontend y despliegue

Tras la puesta en marcha del servidor ya podemos ejecutar el frontend siguiendo estos pasos:

1. Clonamos el repositorio que contiene mi frontend de angular:
    a. Git clone https://github.com/Mohaek10/FutPro-Frontend.git
    b. Nos movemos a la carpeta que contiene mi aplicación: cd futpro-tfg
2. Si tiene usted instalado node con una versión superior a la 16 no tendrás problemas con la instalacion, si no es asi por favor actualice su node.
3. Tras esa comprobación ya podemos instalar los packetes necesarios con este comando:
    a. npm install
4. Tras la instalación de todas las dependencias arrancamos el servidor de angular, con la ayuda del CLI de angular. Usamos este comando:
    a. ng seve
5. Tras ejecutar ese comando procedemos a comprobar en el navegador localhost:4200, si todo está en funcionamiento le aparecerá el dashboard:
6. 

Librerías usadas en el servidor:

1. asgiref==3.8.1

- **Descripción**: asgiref (ASGI Reference) es una librería de referencia para el protocolo ASGI (Asynchronous Server Gateway Interface). ASGI es el estándar para aplicaciones web asincrónicas en Python, y asgiref proporciona herramientas y utilidades para trabajar con ASGI en Django.
- **Uso**: Es esencial para la compatibilidad de Django con ASGI, permitiendo el manejo de solicitudes asíncronas.

2. cffi==1.16.0

- **Descripción**: cffi (C Foreign Function Interface for Python) es una librería para llamar a código C desde Python. Proporciona una forma simple y eficiente de interactuar con bibliotecas escritas en C.
- **Uso**: Utilizada por cryptography para interactuar con librerías criptográficas de bajo nivel.

3. cryptography==42.0.7

- **Descripción**: cryptography es una librería para criptografía en Python. Proporciona herramientas para realizar operaciones criptográficas como cifrado, firmas, hashing, etc.
- **Uso**: Utilizada para asegurar datos sensibles, como contraseñas y tokens.

4. Django==5.0.6

- **Descripción**: Django es un framework web de alto nivel para desarrollar aplicaciones web rápidas, seguras y escalables. Proporciona una estructura de trabajo completa que incluye ORM, autenticación, administración, y más.
- **Uso**: Es el framework principal para desarrollar tu aplicación web.

5. django-cors-headers==4.3.1

- **Descripción**: django-cors-headers es una aplicación Django para manejar encabezados CORS (Cross-Origin Resource Sharing). Permite que tu API sea accesible desde diferentes dominios.
- **Uso**: Configura y permite CORS para que la API pueda ser accedida desde clientes externos.

6. django-filter==24.2

- **Descripción**: django-filter es una librería que facilita la creación de filtros complejos para las consultas en Django. Proporciona una forma declarativa de filtrar consultas basadas en los parámetros de la URL.
- **Uso**: Facilita la implementación de filtros en las vistas de la API.

7. djangorestframework==3.15.1

- **Descripción**: djangorestframework (DRF) es una librería poderosa y flexible para construir APIs web en Django. Proporciona herramientas para la serialización, vistas basadas en clases, autenticación, y más.
- **Uso**: Esencial para crear y gestionar APIs RESTful en tu aplicación Django.

8. djangorestframework-simplejwt==5.3.1

- **Descripción**: djangorestframework-simplejwt es una librería para manejar la autenticación basada en JSON Web Tokens (JWT) en DRF. Proporciona una forma segura de autenticar usuarios usando tokens.
- **Uso**: Implementa la autenticación basada en JWT en tu aplicación.

9. pillow==10.3.0

- **Descripción**: Pillow es una librería de procesamiento de imágenes en Python. Permite abrir, manipular y guardar varios formatos de archivos de imagen.
- **Uso**: Utilizada para manejar imágenes en tu aplicación, como el procesamiento de fotos de perfil o imágenes de productos.

10. psycopg2-binary==2.9.9

- **Descripción**: psycopg2-binary es un adaptador de base de datos PostgreSQL para Python. Permite que las aplicaciones Python se conecten y operen con bases de datos PostgreSQL.
- **Uso**: Es el adaptador principal utilizado para conectar Django con la base de datos PostgreSQL.

11. pycparser==2.22

- **Descripción**: pycparser es un parser en C puro para Python. Es una dependencia de cffi y es utilizada para analizar el código C.
- **Uso**: Proporciona la funcionalidad de parsing necesaria para cffi.

12. PyJWT==2.8.0

- **Descripción**: PyJWT es una librería para trabajar con JSON Web Tokens en Python. Permite codificar y decodificar JWTs.
- **Uso**: Utilizada en djangorestframework-simplejwt para manejar la autenticación JWT.

13. sqlparse==0.5.0

- **Descripción**: sqlparse es una herramienta de análisis y formateo de SQL escrita en Python.
- **Uso**: Utilizada internamente por Django para analizar y formatear consultas SQL.

14. tzdata==2024.1

- **Descripción**: tzdata proporciona información sobre las zonas horarias. Es una base de datos de las zonas horarias utilizadas por muchas aplicaciones.
- **Uso**: Utilizada por Django para manejar las zonas horarias en las aplicaciones.

Librerías de Pruebas

15. pytest==8.2.2

- **Descripción**: pytest es una herramienta de testing para Python que facilita la escritura de tests simples y escalables.
- **Uso**: Utilizada para escribir y ejecutar tests automatizados en tu aplicación Django

Librerías y dependencias usadas en Angular

1. animate.css

- **Descripción**: `animate.css` es una biblioteca de animaciones CSS cross-browser. Es una colección de animaciones CSS listas para usar para cualquier proyecto web.
- **Uso**: Se utiliza para agregar animaciones predefinidas a los elementos de la interfaz de usuario, mejorando la experiencia visual y la interactividad de la aplicación.

2. jwt-decode

- **Descripción**: `jwt-decode` es una biblioteca que permite decodificar JSON Web Tokens (JWT) sin necesidad de una clave secreta. Es útil para extraer información del token JWT.
- **Uso**: Se utiliza para decodificar los JWT y extraer información del usuario, como roles y permisos, sin necesidad de enviarlo al servidor.

3. ngx-pagination

- **Descripción**: `ngx-pagination` es una biblioteca de Angular para paginación. Proporciona componentes y directivas para agregar paginación a cualquier lista o tabla en Angular.
- **Uso**: Se utiliza para implementar la funcionalidad de paginación en listas o tablas de datos, mejorando la navegación y la experiencia del usuario al manejar grandes cantidades de datos.

4. rxjs

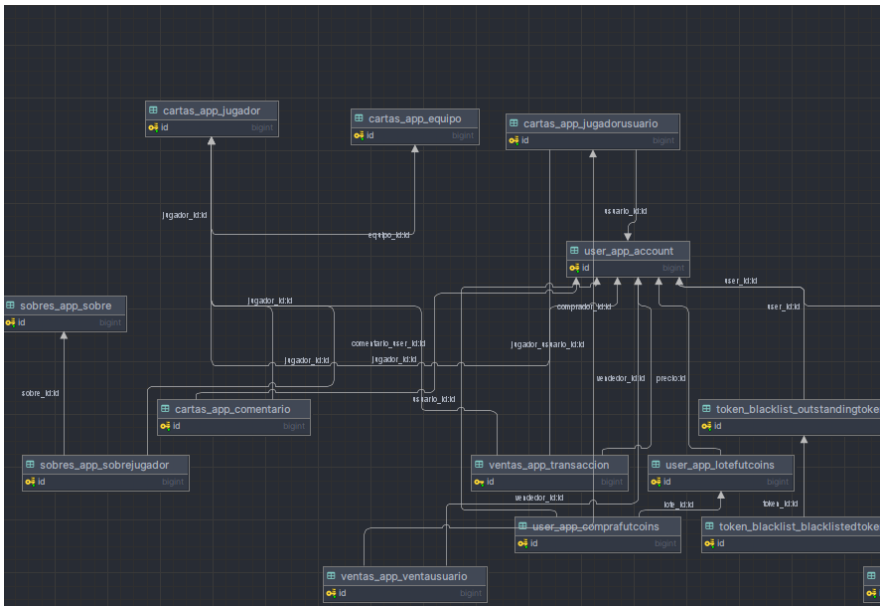
- **Descripción**: `rxjs` (Reactive Extensions for JavaScript) es una biblioteca para la programación reactiva usando observables. Proporciona herramientas para componer programas asíncronos y basados en eventos utilizando secuencias observables.
- **Uso**: Se utiliza para manejar la programación reactiva en Angular, especialmente para la gestión de datos asíncronos, eventos y flujos de datos en tiempo real.

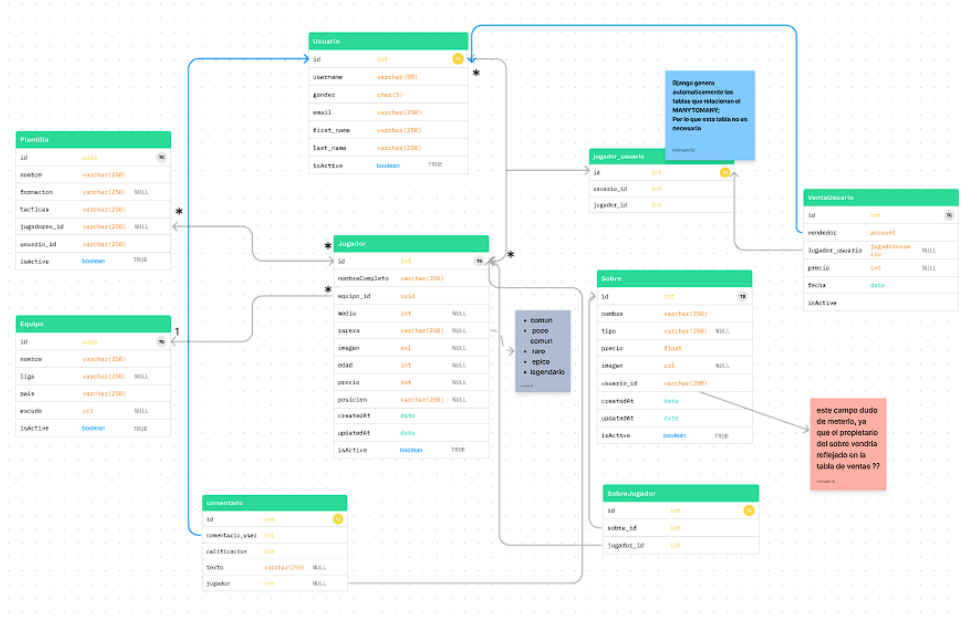
5. sweetalert2

- **Descripción**: `sweetalert2` es una biblioteca de alertas personalizables y fáciles de usar para la web. Proporciona alertas modales que son visualmente atractivas y funcionales.

- **Uso**: Se utiliza para mostrar alertas modales interactivas y personalizadas en la aplicación, mejorando la comunicación con el usuario y proporcionando feedback inmediato y claro.

Esquemas de las tablas y relaciones:





Arquitectura de angular:

Descripción de los Directorios y su Propósito

`src`

Contiene los archivos raíz de la aplicación Angular.

- `favicon.ico`: Ícono de la aplicación.
- `index.html`: Página HTML principal de la aplicación.
- `main.ts`: Punto de entrada principal de la aplicación Angular.
- `styles.css`: Archivo global de estilos.

`app`

Directorio principal de la aplicación Angular.

- `app.component.*`: Archivos relacionados con el componente raíz de la aplicación.
- `app.config.ts`: Configuración de la aplicación, incluyendo proveedores y rutas.
- `app.routes.ts`: Definición de las rutas de la aplicación.

`core`

Contiene los elementos centrales y servicios reutilizables.

- `guards`: Guardas de rutas para autenticación y autorización.
  - `admin.guard.ts`: Guarda que protege las rutas de administración.
  - `auth.guard.ts`: Guarda que protege las rutas que requieren autenticación.
- `interceptors`: Interceptores HTTP.
  - `auth.interceptor.ts`: Interceptor que agrega el token de autenticación a las solicitudes HTTP.
- `services`: Servicios para gestionar la lógica de negocio y la comunicación con la API.
  - `auth.service.ts`: Servicio de autenticación.
  - `equipos.service.ts`: Servicio para gestionar equipos.
  - `jugadores.service.ts`: Servicio para gestionar jugadores.
  - `mercado.service.ts`: Servicio para gestionar el mercado.
  - `user.service.ts`: Servicio para gestionar usuarios.

`errors`

Contiene componentes de manejo de errores.

- `forbidden`: Componente para la página de acceso denegado.

    - `forbidden.component.*`: Archivos del componente.
- `notfound`: Componente para la página de no encontrado.
    - `notfound.component.*`: Archivos del componente.

`features`

Contiene las funcionalidades principales de la aplicación, organizadas por áreas.

- `admin`: Funcionalidades de administración.
    - `admin-dashboard`: Dashboard de administración.
    - `equipo`: Gestión de equipos (crear, actualizar, ver).
    - `jugador`: Gestión de jugadores (crear, actualizar, ver).
    - `manage-equipo`: Componente para gestionar equipos.
    - `manage-jugadores`: Componente para gestionar jugadores.
    - `manage-users`: Componente para gestionar usuarios.
    - `transacciones`: Componente para gestionar transacciones.
- `auth`: Funcionalidades de autenticación.
    - `login`: Componente de inicio de sesión.
    - `register`: Componente de registro de usuarios.
- `dashboard`: Componente del dashboard principal.
- `futcoins`: Funcionalidades relacionadas con los futcoins.
    - `compra-futcoins-dialog`: Diálogo para comprar futcoins.
- `jugadores`: Funcionalidades relacionadas con jugadores.
    - `jugador-detail`: Detalle de un jugador.
    - `jugador-list`: Lista de jugadores.
- `mercado`: Funcionalidades relacionadas con el mercado.
    - `compra-dialog`: Diálogo de compra.
    - `compra-user-dialog`: Diálogo de compra a usuario.
    - `mercado-sis`: Mercado del sistema.
    - `mercado-use`: Mercado de usuarios.
    - `venta-dialog`: Diálogo de venta.
- `mi-equipo`: Funcionalidades relacionadas con el equipo del usuario.
    - `mi-equipo.component.*`: Archivos del componente.
- `user`: Funcionalidades relacionadas con el perfil del usuario.
    - `mis-transacciones`: Componente para ver las transacciones del usuario.
    - `ver-perfil`: Componente para ver el perfil del usuario.

`layout`

Contiene componentes comunes de la interfaz de usuario.

- `footer`: Componente para el pie de página.
    - `footer.component.*`: Archivos del componente.
- `header`: Componente para el encabezado.
    - `header.component.*`: Archivos del componente.

`shared`

Contiene componentes, modelos y pipes compartidos en la aplicación.

- `components`: Componentes reutilizables.
  - `jugador`: Componente de jugador reutilizable.
    - `jugador.component.*`: Archivos del componente.
- `models`: Modelos de datos.
  - `comentario.models.ts`: Modelo de comentario.
  - `equipo.models.ts`: Modelo de equipo.
  - `jugador.models.ts`: Modelo de jugador.
  - `lote-futcoins.models.ts`: Modelo de lote de futcoins.
  - `user.models.ts`: Modelo de usuario.
- `pipes`: Pipes personalizados.
  - `formato-numero.pipe.ts`: Pipe para formatear números.

Arquitectura de Django:

`.envs`

Contiene archivos de configuración de variables de entorno que no se deben subir al control de versiones.

`.venv`

Directorio del entorno virtual de Python que contiene las dependencias y paquetes instalados.

`cartas_app`

Aplicación encargada de gestionar las cartas o cromos.

- `api`: Lógica relacionada con la API de cartas.
- `migrations`: Migraciones de la base de datos para la aplicación de cartas.
- `__init__.py`: Archivo de inicialización del módulo.
- `admin.py`: Configuración del administrador de Django para las cartas.
- `apps.py`: Configuración de la aplicación.
- `middleware.py`: Middleware específico para la aplicación de cartas.
- `models.py`: Definición de los modelos de datos para las cartas.
- `tests.py`: Pruebas unitarias para la aplicación de cartas.

database

Directorio para archivos relacionados con los datos de pruebas y de inicio

futpro

Directorio principal del proyecto Django que contiene la configuración y el punto de entrada del proyecto.

- `__init__.py`: Archivo de inicialización del módulo.
- `asgi.py`: Configuración para ASGI.
- `settings.py`: Configuración global del proyecto Django.
- `urls.py`: Definición de las rutas y vistas globales del proyecto.
- `wsgi.py`: Configuración para WSGI.

media

Directorio para archivos multimedia subidos por los usuarios.

plantillas_app

Directorio para plantillas HTML que se usarán en las vistas de Django.

sobres_app

Aplicación encargada de gestionar los sobres (paquetes) de cartas.

- `api`: Lógica relacionada con la API de sobres.
- `migrations`: Migraciones de la base de datos para la aplicación de sobres.
- `__init__.py`: Archivo de inicialización del módulo.
- `admin.py`: Configuración del administrador de Django para los sobres.
- `apps.py`: Configuración de la aplicación.
- `models.py`: Definición de los modelos de datos para los sobres.
- `tests.py`: Pruebas unitarias para la aplicación de sobres.

user_app

Aplicación encargada de gestionar los usuarios y la autenticación.

- `api`: Lógica relacionada con la API de usuarios.
- `migrations`: Migraciones de la base de datos para la aplicación de usuarios.
- `__init__.py`: Archivo de inicialización del módulo.
- `admin.py`: Configuración del administrador de Django para los usuarios.
- `apps.py`: Configuración de la aplicación.
- `models.py`: Definición de los modelos de datos para los usuarios.

- `tests.py`: Pruebas unitarias para la aplicación de usuarios.

`ventas_app`

Aplicación encargada de gestionar las ventas de cartas.

- `api`: Lógica relacionada con la API de ventas.
- `migrations`: Migraciones de la base de datos para la aplicación de ventas.
- `__init__.py`: Archivo de inicialización del módulo.
- `admin.py`: Configuración del administrador de Django para las ventas.
- `apps.py`: Configuración de la aplicación.
- `models.py`: Definición de los modelos de datos para las ventas.
- `tests.py`: Pruebas unitarias para la aplicación de ventas.

`.gitignore`

Archivo que especifica los archivos y directorios que Git debe ignorar.

`comandos.txt`

Archivo de texto que contiene comandos útiles para la gestión del proyecto.

`docker-compose.yml`

Archivo de configuración para Docker Compose, que define los servicios, redes y volúmenes para la aplicación.

`Dockerfile`

Archivo de configuración para Docker, que define cómo construir la imagen del contenedor para la aplicación.

`manage.py`

Script de utilidad para la gestión del proyecto Django, permite ejecutar comandos como migraciones, iniciar el servidor de desarrollo, etc.

`pytest.ini`

Archivo de configuración para Pytest, una herramienta de pruebas para Python.

`requirements.txt`

Archivo que lista las dependencias y paquetes requeridos para el proyecto Django.

Fuentes:

- https://angular.dev/
- https://tailwindcss.com/docs/
- https://stackoverflow.com/
- https://www.django-rest-framework.org/
- Diversos blogs explicativos, o de resolución de algún fallo
- https://www.linkedin.com/learning
- https://cosasdedevs.com/