Introducción

Con este proyecto final intentamos demostrar todo lo que sabemos y hemos aprendido durante la duración del curso de Técnico Superior de Desarrollo de Aplicaciones Web pero durante estos dos años lo único que tengo claro es que en el sector tecnológico nunca se sabe lo suficiente. Debemos tener una visión más allá e ir aprendiendo durante toda la carrera profesional que desarrollemos. Este es solo el primer paso, la primera piedra para poder dedicarnos a aquello que nos apasiona. El desarrollo de este proyecto nos permite recopilar los conocimientos que hemos adquirido y enfrentarnos a problemas que surgen durante el proceso. Es muy importante en nuestro comienzo asumir que estos desafíos van a llegar día sí día también y es una parte que hace este mundo de la programación más interesante.

El hecho de asumir que no controlamos todo nos hará más dispuestos a resolver los problemas y seguir el camino. O simplemente, el tener una idea e intentar plasmarla en el código te permite superar y adquirir nuevos conocimientos.

Personalmente al realizar esta memoria me ha llevado a superarme y afrontar situaciones en las que la idea superaba con creces los conocimientos que tenía y eso no ha sido impedimento para afrontarlo y aprender.

* Descripción general del producto:

‘artHUB’ surge de una idea de poder conectar a los artistas que empiezan o no tienen los medios o esa pizca de suerte necesaria para llegar al público. Se trata de una plataforma en la que los usuarios pueden registrarse y subir como si una biblioteca se tratase, sus obras. No importa el tipo de obra, si es una pintura como una fotografía. Además, esta aplicación permite, además de exponer el producto, la posibilidad de adquirir dichas obras.

En un mundo cada vez más digitalizado, la creatividad y el arte se convierten en un medio poderoso para expresar emociones, contar historias y conectar con otras personas. Sin embargo, para muchos artistas que empiezan, la falta de recursos, de oportunidades o simplemente la falta de esa pizca de suerte necesaria puede dificultar el camino hasta llegar al público.

Es en este contexto que surge ‘artHUB’, una plataforma diseñada para conectar a los artistas y con la gente. El objetivo es simple: brindar una aplicación donde los artistas puedan dar a conocer sus obras, sin importar su nivel de experiencia o los medios a su disposición.

En artHUB, los artistas tienen la libertad de compartir sus creaciones con una amplia audiencia, creando así una especie de biblioteca donde cada obra tiene su lugar. No importa si son pinturas o fotografías, se trata de arte digital, todas las formas de expresión artística son bienvenidas en la plataforma.

artHUB intenta ir más allá de ser solo un escaparate para el arte. También se ofrecerá a los usuarios la posibilidad de adquirir las obras que les gusten, facilitando así la conexión directa entre artistas y potenciales compradores.

La idea principal es construir una comunidad, donde los artistas encuentren inspiración, apoyo y oportunidades para crecer y desarrollarse. artHUB quiere ser el puente que conecta el talento con el mundo, eliminando barreras y abriendo puertas hacia un futuro donde el arte sea accesible para todos.

En resumen, artHUB es más que una plataforma, es un movimiento, basado en una experiencia a personal, que busca promover el arte y la creatividad de cualquier persona que se atreva a representar cualquier idea o sentimiento sobre un lienzo o mediante una cámara.

Estudio de tecnologías.

Para la redacción de este apartado separaremos entre el backend, el frontend y el deployment.

* Backend:

Para la realización de la API encargada de tratar y manejar la información y la gran mayoría de la lógica de la aplicación he decidido utilizar el framework Django basado en Python. Al realizar las prácticas tuve mi primer contacto tanto con Python como son su framework estrella Django Rest Framework.

La primera impresión fue muy buena, tanto por su facilidad de compresión y aprendizaje de su uso básico. Esto y las posibilades que me ofreció su propio ORM para llegar a los datos de la BD y filtrarlos fue lo que me impulsó a elegirlo para realizar la aplicación.

En este proyecto he utilizado la parte de los modelos (models), controladores de vista (viewsets) donde se agrupa toda la lógica de la vista relacionada y el enrutamiento hacia diferentes endpoints para poder suminstrar la información y la manera en la que la necesitaba.

Además, he utilizado los serializers proporcionados por Django Rest Framework. Los serializers se utilizan en la transformación de los datos de los modelos de Django en formatos que sean fáciles de manejar y transmitir, como JSON.

Con los serializers, he podido definir cómo se serializan y deserializan los datos de mis modelos al interactuar con la API.

Esto incluye la validación de los datos de entrada que llegan desde el frontend, la gestión de relaciones entre modelos y la personalización de los datos para adaptarse a las necesidades que requerían las peticiones realizadas.

Django Rest Framework proporciona un Router que gestiona automáticamente las urls para mostrar la información de los modelos, tanto para crear, modificar como borrar, pero en mi caso he necesitado crear casi todos los enpoints manualmente por la relación entre modelos que existía. Por ejemplo, al crear un Artist, necesitaba crear primero un User con el que relacionarlo.

class ArtistViewSet(viewsets.ModelViewSet):  
 queryset = Artist.objects.all()  
 serializer\_class = ArtistSerializer  
  
 def create(self, request, \*args, \*\*kwargs):  
 user = dict()  
 artist = dict()  
  
 user['username'] = self.request.data.get('username')  
 user['first\_name'] = self.request.data.get('first\_name')  
 user['last\_name'] = self.request.data.get('last\_name')  
 user['email'] = self.request.data.get('email')  
 user['password'] = self.request.data.get('password')  
 user['avatar'] = self.request.data.get('avatar')  
 artist['nickname'] = self.request.data.get('nickname')  
  
 user\_data\_response = CustomUserSerializer(data=user)  
  
 try:  
 user = CustomUser.objects.create\_user(\*\*user)  
 except IntegrityError:  
 return Response({'error': 'User already exists'},status=status.HTTP\_400\_BAD\_REQUEST)  
  
 artist = Artist(\*\*artist)  
 artist.user = user artist.save()  
  
 artist\_data\_response = self.serializer\_class(data=artist)  
  
 return Response({'message': 'User and artist created successfully'},  
 status=status.HTTP\_201\_CREATED)

En este ejemplo, he sobreescrito la función créate del propio Router y la he definido manualmente para crear un Artist. Primero, creo el User y con su ‘id’ me permite crear el Artist relacionado.