Un API es un conjunto de funcionalidades o recursos que nos expone un sistema para poder interactuar con él desde otro sistema.app web que realiza peticiones a diferentes APIs de diferentes servicios de streaming de música.

La comunicación entre apps es asíncrona, de ahi que se usen peticiones ajax. Los dominios tienen que ser iguales pero a veces no es posible porque las apps pertenecen a empresas diferentes y por ende tienen diferentes dominios. Salta la política de cors. Es necesario ajustar para permitir la comunicación entre dominios diferentes.

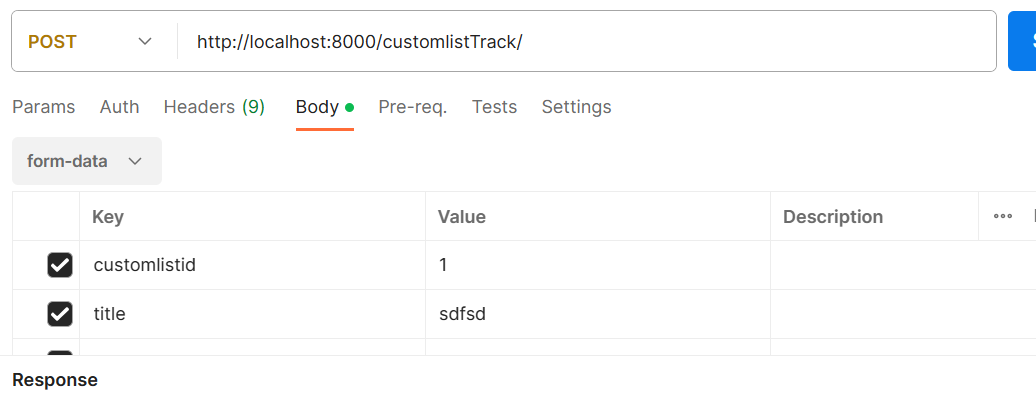
Nginx se suele usar no solo para desplegar sevidores web sino para proxys inversos

Django Rest Framework es una herramienta para la creación de API REST Web y está basado en el framework de desarrollo web de backend Django, el cual está escrito en Python. Su metodología de trabajo se basa en lo siguiente: En el servidor almacena la lógica y en el cliente estaría la interfaz, por lo que, a diferencia de Django, en Django Rest no vamos a usar plantillas con CSS. Entre sus características principales destaca la utilización de unas políticas de autenticación especiales (incluidos paquetes para OAuth1a y OAuth2), el sistema de serialización que admite orígenes de datos ORM y no ORM, el uso de vistas basadas en funciones personalizables, etc. REST es un acrónimo que significa *Representational State Transfer*, y se refiere a una **arquitectura de software para crear servicios web**. Para que todos los componentes de una API Web se comuniquen sin ningún problema, establece algunas reglas. Entre ellas se encuentra que utilicen el mismo tipo de dato para la comunicación de la información. Así, el estándar para devolver la información será por lo general o en formato xml o en formato JSON, este último el más usado.Los **servicios RESTful** emplean el protocolo HTTP(como GET, POST, PUT, DELETE) para realizar operaciones en los recursos que se encuentran en el servidor. REST es una buena opción cuando se quiere crear una API para un sistema que será accedido por diferentes tipos de dispositivos o aplicaciones, como móviles, web o de escritorio. Si deseas permitir que tus servicios web sean consumidos con facilidad por otras aplicaciones y que sean escalablesAdemás, Django Rest Framework toma como base el protocolo http y ofrece diversos métodos que puede usar el cliente para mandar peticiones: GET, POST, PUT y DELETE. Por último, define que la petición no tiene que tener estado (no tiene que depender de otra petición para poder responder) pero el servidor tiene que indicarle al cliente unos códigos de estado junto con la información a devolver, los cuales se clasifican en 4 grupos: 20X (Éxito), 30X (Redirección), 40X (No encontrado) y 50X (Error de servidor)

El marco REST proporciona varios esquemas de autenticación listos para usar y también le permite implementar esquemas personalizados.

request.userrequest.auth

Autenticación, la cual no es mas que un conjunto de mecanismos para asociar la solicitud entrante con un conjunto de credenciales de identificación que puede contener el usuario del cual proviene la petición o el token perteneciente a un usuario, dependiendo los mecanismos que se hayan definido, se valida si la solicitud puede ser permitida o no. autenticación siempre se ejecuta al comienzo de la vista antes de que se produzcan las comprobaciones de permisos y limitaciones y antes de cualquier procesamiento de datos en alguna vista, por ello Django Rest Framework, luego de validar la petición, guardará la instancia de la clase User en request.user. Ej views de customlisttrack

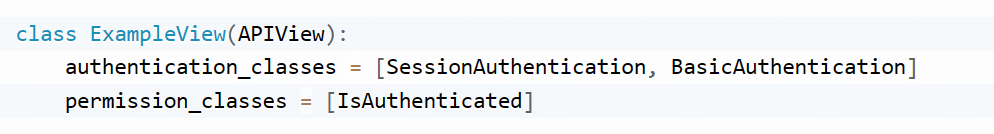


Además de ello, existe otra propiedad llamada request.auth, la cual se utiliza para cualquier información adicional de la autenticación, por ejemplo, representar el Token para la Autenticación.Los esquemas de autenticación predeterminados se pueden establecer globalmente, utilizando la configuración. Por ejemplo.DEFAULT\_AUTHENTICATION\_CLASSES

En settings.py



Esta configuración no sólo se puede indicar de forma global, podemos utilizarlas para que sólo se apliquen a una vista en específico.



Este tipo de autenticación utiliza un esquema HTTP simple basado en Token, un Token es un secuencia de caracteres especiales encriptados que representan información del usuario al cual se asocian. Este tipo de autenticación es ideal para configuraciones a proyectos cliente-servidor, donde hay clientes de varios tipos, puede ser escritorio o móvil.

Esta clase, incorporada en Django Rest Framework puede utilizarse como Autenticación Global, para ello sólo debemos aplicarla de la misma forma que SessionAuthentication y BasicAuthentication, adicionalmente debemos agregar rest\_framework.authtoken a nuestro archivo de configuración.

Cuando se realiza este tipo de Autenticación, utilizando esta clase, la clave del token debe incluirse en el header de la petición HTTP, debe tener como prefijo la palabra Token seguido de un espacio y luego el Token en cuestión.

Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b

Si la autenticación es realizada correctamente, colocará las siguientes credenciales en el request:

request.user será una instancia del modelo User que se halla definido.

request.auth será será una instancia de rest\_framework.authtoken.models.Token.

Toda respuesta que no se ha autenticado se les negará el permiso y darán como resultado una respuesta HTTP 401 Unauthorized .

1. Autentificación básica

Se basa principalmente en un nombre de usuario y una contraseña para identificarte.Cualquiera que intercepte la trasmisión de datos puede decodificar fácilmente esta información. Esto se denomina [ataque Man-In-The-Middle (MiTM)](https://es.wikipedia.org/wiki/Ataque_de_intermediario).Para proteger tu API mediante la autentificación básica debes configurar que las conexiones entre los clientes y tu servicio API funcionen únicamente mediante una conexión TLS/HTTPS, nunca sobre HTTP.

1. Autentificación basada en token

usuario y contraseña. Pero en este caso, con la primera petición de autentificación, el servidor generará un token basado en esas credenciales.El servidor guarda en base de datos este registro y lo devuelve al usuario para que a partir de ese momento no envíe más credenciales de inicio de sesión en cada petición HTTP.  En lugar de las credenciales, simplemente se debe enviar el token codificado en cada petición HTTP.Por norma general, los tokens están codificados con la fecha y la hora para que en caso de que alguien intercepte el token con un ataque MiTM, no pueda utilizarlo pasado un tiempo establecido. Además de que el token se puede configurar para que caduque después de un tiempo definido, por lo que los usuarios deberán iniciar sesión de nuevo.

1. Autentificación basada en clave API

en este caso primero debes configurar el acceso a los recursos de tu API. Tu sistema API debe generar una clave (*key*) y un *secret key* para cada cliente que requiera acceso a tus servicios. Cada vez que una aplicación necesite consumir los datos de tu API, deberás enviar tanto la *key* como la *secret key*.Este sistema es más seguro que los métodos anteriores, pero la generación de credenciales debe ser manual y esto dificulta la escalabilidad de tu API.Puede ser complicado poder almacenar y administrar estas credenciales. Es por ello que es imprescindible contar con una [API Gateway](https://www.itdo.com/blog/api-gateway-en-tu-arquitectura-de-microservicios/). Es un sistema intermediario que proporciona una [interfaz API REST o WebSocket](https://www.itdo.com/blog/rest-vs-websocket-que-diferencia-hay/) para hacer de enrutador desde un único punto de entrada, el API Gateway, hacia un grupo de microservicios y/o API de terceros definidos. Interactúa como puerta de enlace “Gateway”.

1. OAuth 2.0 (Autorización abierta)

Su propósito es permitir a otros proveedores, servicios o aplicaciones, el acceso a la información sin facilitar directamente las credenciales de los usuarios.accederán bajo la confirmación del usuario

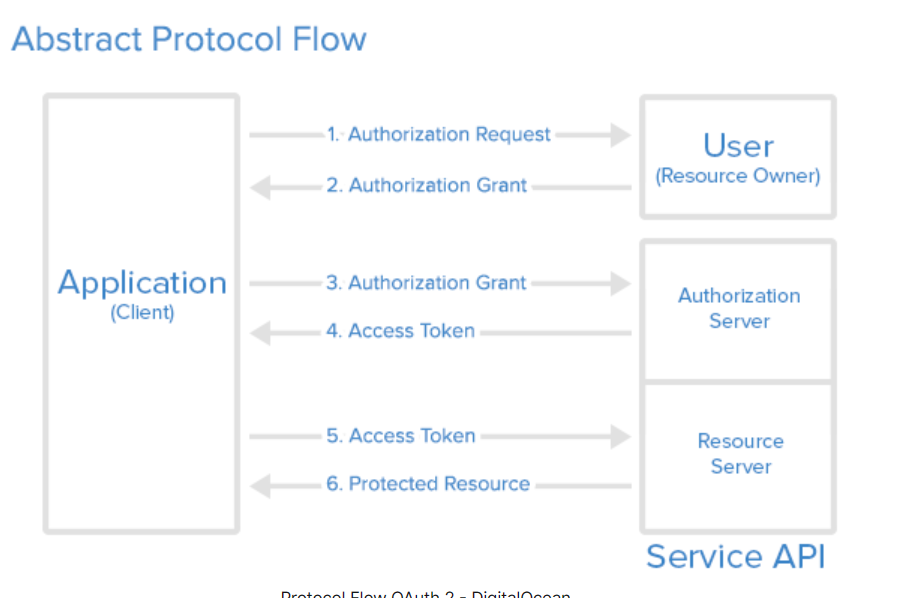
si configuro OAuth como autenticación en mi API, eso le permitiría delegar a un tercero para que use sus datos.

App x quiere acceder a los datos de tu cuenta de google

Le permitirá a la app...allow o cancel

solicitando acceso a la cuenta de usuario de Google.

se solicita el acceso a la gestión de Google Calendar. Una vez aprobada la autorización, esta aplicación de terceros podrá acceder a la información permitida mediante una autentificación con un token de acceso.



Adecuado a mis necesidades

Para editar las cors proxys inversos

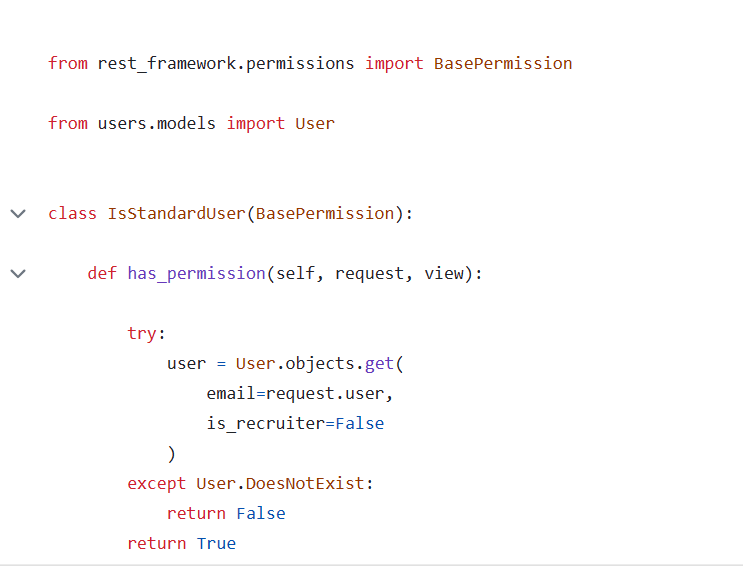
Dockers, herramienta de creación de contenedores generados a través de imágenes en donde se incluyen solamente los recursos necesarios para el servidor, hemos elegido esta alternativa debido a su portabilidad, escalabilidad, portabilidad y eficiencia tamaño respecto a otras alternativas

curl -X GET http://127.0.0.1:8000/api/example/ -H 'Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b'

No es un navegador

Además de PHP + Nginx para el balanceo de cargas

Django Rest Framework tiene por defecto algunos paquetes de permisos que permiten saber por ejemplo si el usuario está autenticado, mediante la clase IsAuthenticated, pero además en esta aplicación se añade una restricción nueva para evitar que el administrador modifique o añada accidentalmente datos de los usuarios desde la url y pueda hacerlo sólo desde el panel de administrador para mayor seguridad. Se creó un archivo permissions.py en el que se añadió el siguiente código

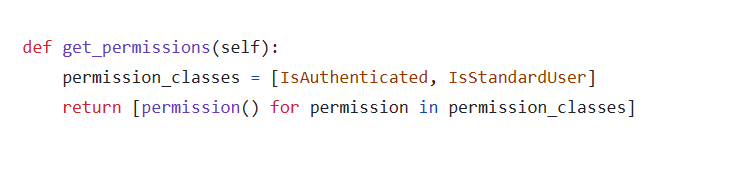


el método authenticate que necesitará como parámetros obligatorios username y password

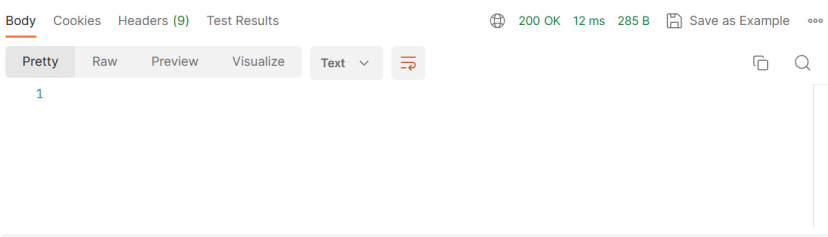
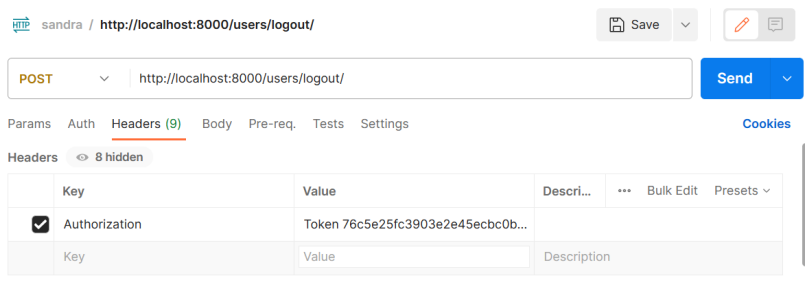


En el login se opta por sustituir el username obligatorio por email, este cambio se refleja en UserLoginSerializer y además hay que especificarle en models.py que el campo username sería el equivalente al email:

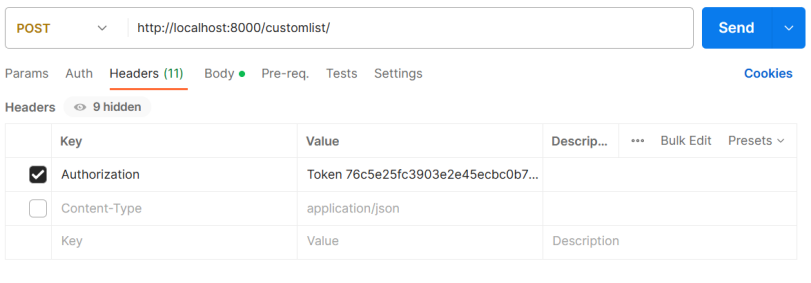
método get\_permissions para verificar que quien requiera una acción de listado, modificación,creación o borrado desde la url sea el usuario estándar y no el administrador, que sólo podrá realizar esta acción desde el panel de administración. En views.py de customlisttrack



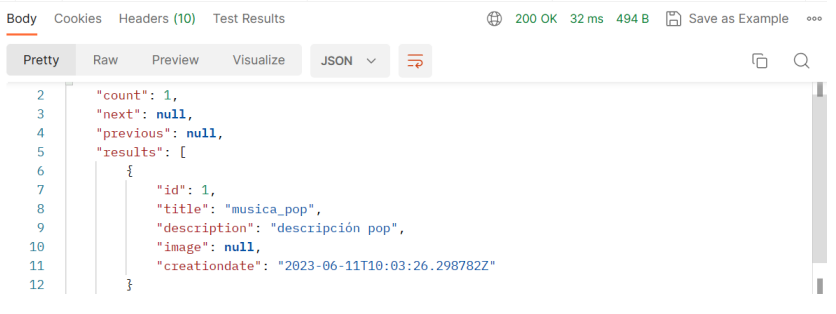
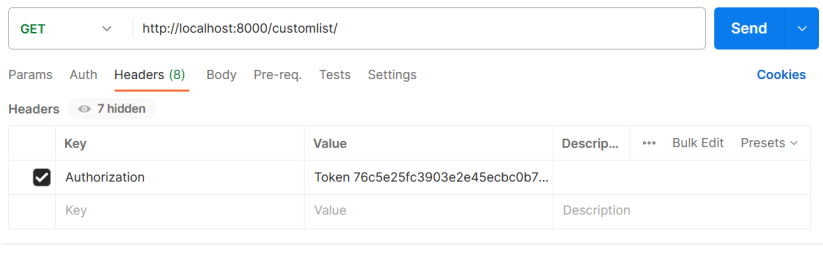
Para cerrar la sesión de usuario hay que pasarle por cabecera a http://localhost:8000/users/logout/ la variable Authentication en la que especificamos Token seguido de un espacio y el token asociado al usuario que quiere cerrar sesión.



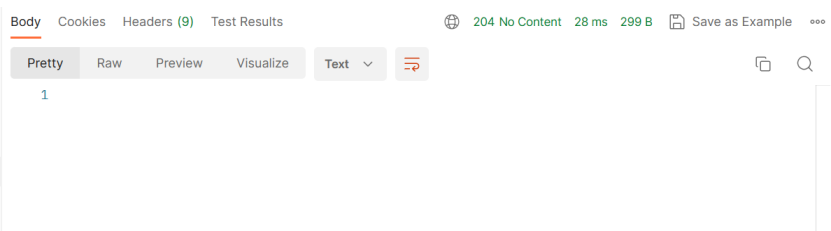
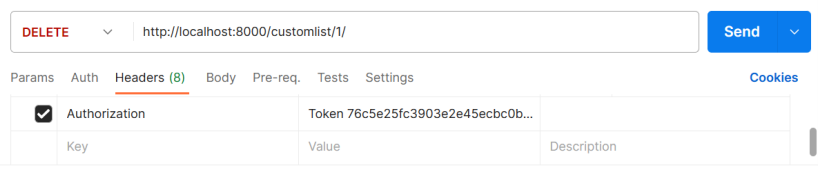
Para la creación de una lista de usuario a través de http://localhost:8000/customlist/ necesitamos pasarle como cabecera la clave authorization junto con el token del usuario.



Con el método GET y especificando en la cabecera el token del usuario, el usuario accederá a <http://localhost:8000/customlist>



para poder borrarla se hace de la siguiente manera:



Cuando se realizan peticiones de tipo REST entre orígenes y destinos existe una política de seguridad encargada de proteger este método de comunicación conocido como CORS(Cross Origin Resource Sharing) el cual se encarga de impedir el acceso o la comunicación entre terminales con diferente dominio y dirección IP y puerto. Cuando hago peticiones se hacen desde ip y puerto porque un cliente hace muchas petis, se identifican por ip. Por ejemplo mi movil accede a google o al navegador para hacer varias cosas. Yt, spoty...todas las peticiones misma ip porque mismo equipo. Para que el servidor sepa en que momento enviarle los paquetes de spoty o yt lo sabe por el puerto. Es como la banda de comunicacion. Mediante los puertos identificas los recursos porque si me meto en chrome y abro varias pestañas todas tienen la misma identificacion. Telefono y ordenador misma ip. Que info envio puerto. A continuación, se detallan los ajustes a realizar y sus correspondientes apartados:

1. Instalación de las cabeceras CORS en el backend, a través del comando pip install django-cors-headers

2. Ajustes de la aplicación settings.py. Se registran las cabeceras recién instaladas en el apartado INSTALLED\_APPS. Después, en el apartado MIDDLEWARE se configura un middleware con preferencia, el cual se encargará de procesar las peticiones CORS



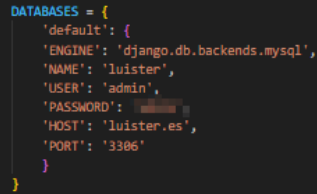
Se añaden los dominios que se quieran permitir, en nuestro caso el cliente 3000 y el servidor desplegado en Docker Luister-website



Se añade como orígenes o hosts permitidos, la IP del servidor y su correspondiente dominio.



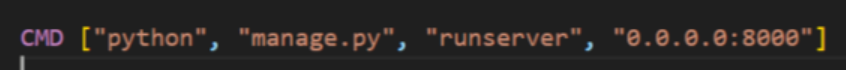
En este mismo fichero, se procede a modificar la configuración por defecto de la base de datos, por la del servidor de bases de datos de producción de la aplicación. Se indica el motor a utilizar (Engine), nombre de la base de datos, el usuario, la contraseña, el host o dirección IP del servidor y el puerto de escucha



Por último, se añade un apartado para gestionar las credenciales de la sesión en las cabeceras de las peticiones CORS



Dockerfile Fue necesario añadir una línea con los CMD ya que es la encargada de ejecutar los comandos necesarios para, en este caso, habilitar el servicio principal del servidor(Python).



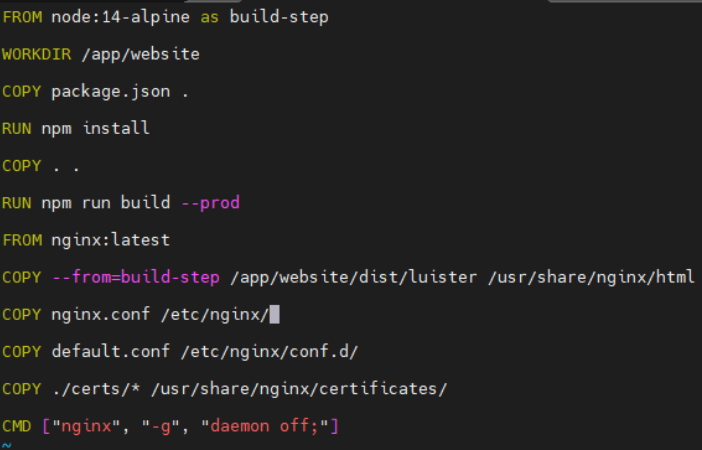
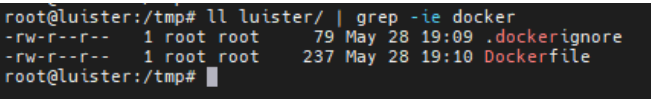
A la hora de instalar django saltaba el siguiente error:



así pues, tuvimos que añadir mysqlient en el archivo requirements.

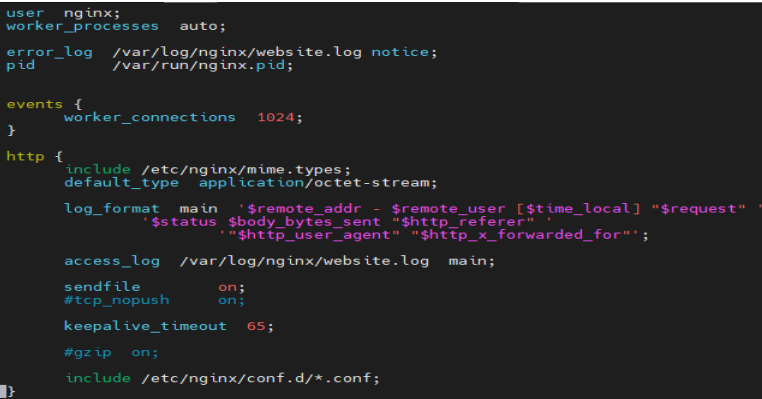


Los archivos Dockerfile son ficheros de texto utilizados para configurar una imagen de Docker. A partir de las imágenes, se crearán los servidores en los que se encontrarán en ejecución las aplicaciones y servicios. Es un componente fundamental en el despliegue de aplicaciones en Docker, y consta de un conjunto de instrucciones que se detallarán posteriormente. Se recomienda crear estos ficheros en la raíz o directorio padre/madre del proyecto o aplicación a desplegar



FROM: Especifica la imagen base a partir de la cual se construirá el sistema. En este caso, se hará uso de una imagen de Node, ya que la aplicación está desarrollada con el Framework Angular, el cual hace uso de node y el gestor de paquetes npm. ● WORKDIR: Especifica el directorio de trabajo de la aplicación. En este caso, se hará uso de la ruta /app/website/ creada para la aplicación cliente. ● COPY: Copia ficheros y directorios desde el sistema de archivos local al contenedor. En este caso, se copia el fichero package.json de la aplicación de angular, al directorio del contenedor. ● RUN: Con esta instrucción, se ejecuta un comando en dentro del contenedor durante el proceso de creación de la imagen. En la cuarta instrucción, se encarga de ejecutar el comando npm install, mediante el cual se sincroniza las dependencias de la aplicación y se instalan los recursos necesarios. ● En la quinta instrucción, nuevamente se hace uso del COPY para copiar todos los ficheros del proyecto, al directorio de trabajo. ● En la sexta instrucción, se realiza el empaquetado de la aplicación mediante la sentencia RUN npm run build --prod. El resultado de este comando es un directorio de nombre /dist/ que vendría a ser la compilación de la aplicación. El despliegue de la aplicación se realizará mediante este último recurso. ● En la séptima sentencia se hace uso de la instrucción FROM, para indicar que se hará uso de una imagen adicional para ejecutar las instrucciones posteriores. En este caso, se hace uso de la imagen de nginx, un servidor web ligero de alto rendimiento, proxy inverso y servidor de correo electrónico, ya que será la encargada de poner en marcha la aplicación cliente. ● En la octava sentencia, se hace uso de la instrucción COPY, para copiar los ficheros generados en la imagen anterior, resultado de la compilación, en el directorio de nginx reservado para los recursos, HTML, CSS, JS y demás, necesarios para el correcto funcionamiento de la aplicación web cliente. ● En la novena sentencia, se copia el fichero personalizado de configuración de nginx al directorio de la aplicación, para que ésta funcione con las configuraciones establecidas en dicho fichero y con los ajustes por defecto. Más adelante se mostrarán los detalles designados en este fichero de configuración. ● En la décima sentencia, se copian los certificados SSL de la aplicación, a la ruta del contenedor, desde la cual se hará uso de dichos ficheros. Éstos son necesarios para poder acceder al servidor mediante el protocolo HTTPS. ● CMD: Mediante esta instrucción, se especifica el comando a ejecutarse al iniciarse el contenedor. En este caso, se ejecuta el comando nginx para poner en marcha el servidor

Se crea el fichero nginx.conf y default.conf, los cual se copiarán al directorio de configuración de nginx para aplicar la configuración deseada al contenedor, lo cual incluye el proxy inverso para evitar el bloqueo, por CORS (Cross-Origin Resource Sharing), de las peticiones de la aplicación cliente a las APIs externas a las que necesita acceder. La configuración del fichero default.conf se encuentra incluida en una directiva server, ya que este fichero luego sería incluido en el archivo principal de configuración de nginx /etc/nginx/nginx.conf.



user nginx: Se especifica que el usuario encargado de ejecutar el proceso principal de NGINX es el usuario del mismo nombre. ● worker\_processes auto: Número de procesos de trabajo definidos para manejar las peticiones. El valor auto permite que NGINX ajuste el número de procesos en función de la cantidad de núcleos de la unidad central de procesamiento. ● error\_log /var/log/nginx/website.log notice: Establece la ruta y el nivel de registro de errores de NGINX. ● pid /var/run/nginx.pid: Especifica la ruta donde se almacenará el archivo PID (identificador de proceso) de NGINX. ● events: Ajustes referente a los eventos en NGINX, como las conexiones de los clientes. ● worker\_connections 1024: Número de conexiones simultáneas que un proceso de trabajo puede manejar. ● http: Define las configuraciones principales para el protocolo HTTP. ● include /etc/nginx/mime.types: Incluye el archivo de tipos MIME predefinidos en la configuración de NGINX. Este archivo proporciona mapeos entre extensiones de archivo y tipos MIME. ● default\_type application/octet-stream: Establece el tipo MIME predeterminado para las respuestas sin tipo MIME especificado. ● log\_format main: Define un formato de registro personalizado llamado "main". Esta directiva especifica cómo se registrará la información en los registros de acceso. ● access\_log /var/log/nginx/website.log main: Configura el registro de acceso de NGINX y especifica que se utilizará el formato "main". Los registros de acceso se guardarán en el archivo /var/log/nginx/website.log. ● sendfile on: Activa la funcionalidad de envío de archivos de NGINX, lo que permite la transmisión eficiente de archivos estáticos. ● keepalive\_timeout 65: Establece el tiempo máximo que una conexión se mantendrá abierta para permitir solicitudes posteriores (keep-alive). ● include /etc/nginx/conf.d/\*.conf: Incluye archivos de configuración adicionales ubicados en el directorio /etc/nginx/conf.d/. Ofrece modularidad y la organización de la configuración en archivos separados.

El paquete OAuth del marco REST [de Django](https://jpadilla.github.io/django-rest-framework-oauth/) proporciona compatibilidad con OAuth1 y OAuth2 para el marco REST.

pip install djangorestframework-oauth

JSON Web Token es un estándar bastante nuevo que se puede usar para la autenticación basada en tokens. A diferencia del esquema TokenAuthentication integrado, JWT Authentication no necesita usar una base de datos para validar un token

Un paquete para la autenticación JWT es [djangorestframework-simplejwt](https://github.com/davesque/django-rest-framework-simplejwt) que proporciona algunas características, así como una aplicación de lista negra de tokens conectables.