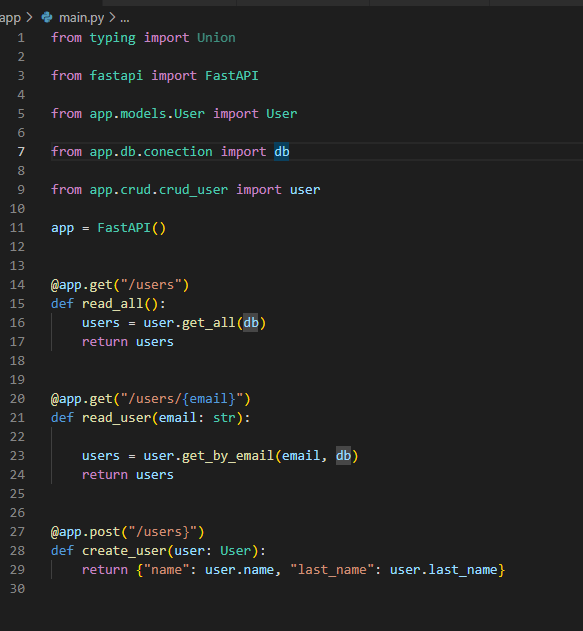
Documentación

1. Elegí el lenguaje de programación Python y utilice el framework fastApi ya que ha sido con lo que he trabajado para crear api REST.
2. Definí como formato de entrada y salida JSON ya que es más ligero y sencillo que XML y además es con el que más he trabajado. Para base de datos elegí la de Firebase ya que es NoSQL y es sencillo de configurar.
   1. Instale fastApi y uvicorn con los siguientes comandos.

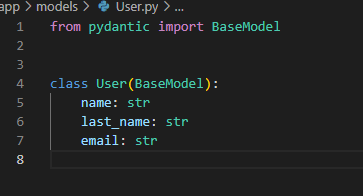


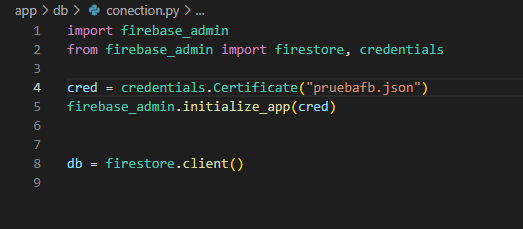


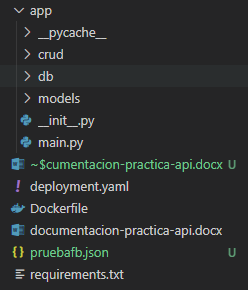
* 1. Cree la estructura base del api con sus diferentes endpoints en la clase main.



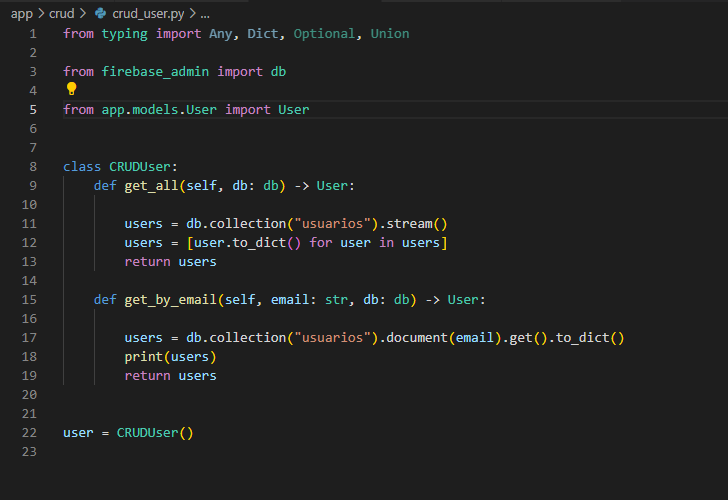
* 1. Cree la estructura para el modelo de datos que iba a necesitar, en este caso cree un modelo sencillo para un usuario el cual tiene “name”, “last\_name” y “email”.



* 1. Cree la configuración para la conexión con la base de datos de Firebase, para esto dentro de mi consola de Firebase seguí los pasos para crear una nueva base de datos Firestore y descargue el archivo JSON con las credenciales para realizar la conexión. (Por Seguridad se omitió el archivo al subirse a Git, este va en la carpeta raíz)

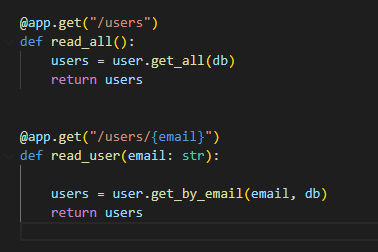


* 1. Una vez ya teniendo la conexión cree el archivo con los diferentes métodos para el crud de la aplicación, el cual era para leer todos los documentos de una colección de Firebase y otro el cual lee un documento en especifico proporcionando el correo con el que este registrado dicho elemento.



* 1. Una vez ya teniendo la estructura solo consumí los métodos del crud en el archivo main dependiendo del enpoint consumido y corrí el proyecto con el siguiente comando desde fuera de la carpeta app ya que si lo corremos desde dentro nos daría problemas con los imports de los diferentes modulos. 

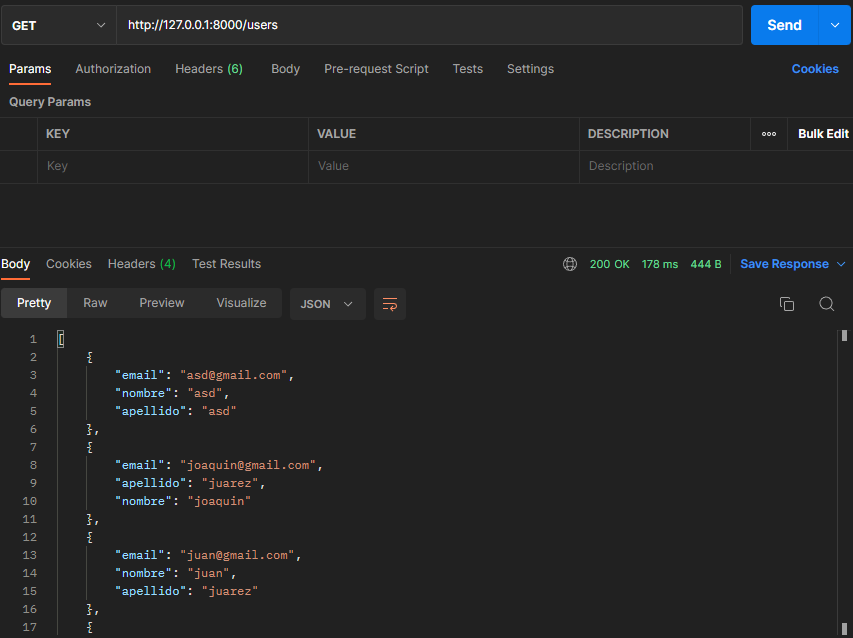
1. Implementamos la característica de hipermedia a la API lo cual nos permitirá obtener la información de un usuario en especifico si se define en el URL de la llamada.

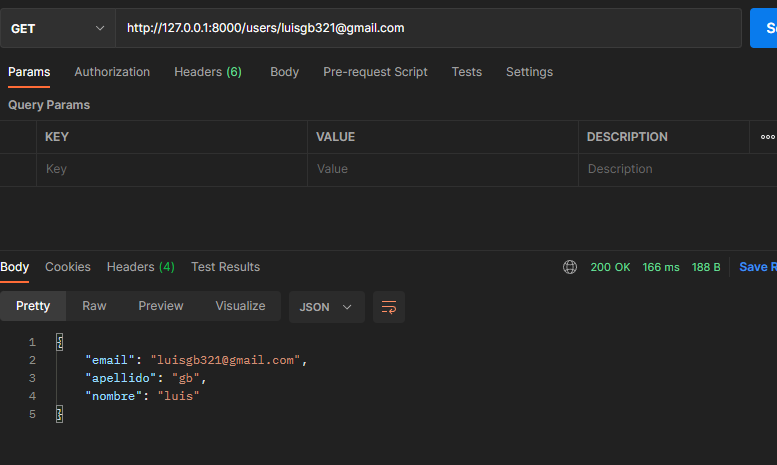


1. Una vez ya con la aplicación corriendo podemos ingresar a el siguiente URL para comprobar que nuestro servicio este corriendo <http://127.0.0.1:8000/users>, aquí nos debería de desplegar una lista de los usuarios que están guardados en la base de datos.

Si después del users en el URL colocamos alguno de los correos de los usuarios que nos despliega la lista ([127.0.0.1:8000/users/luisgb321@gmail.com](http://127.0.0.1:8000/users/luisgb321@gmail.com)) nos arrojara la información de ese usuario en específico.

1. Probamos en Postman para ver que nuestros endpoints funcionan como se espera.

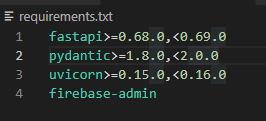




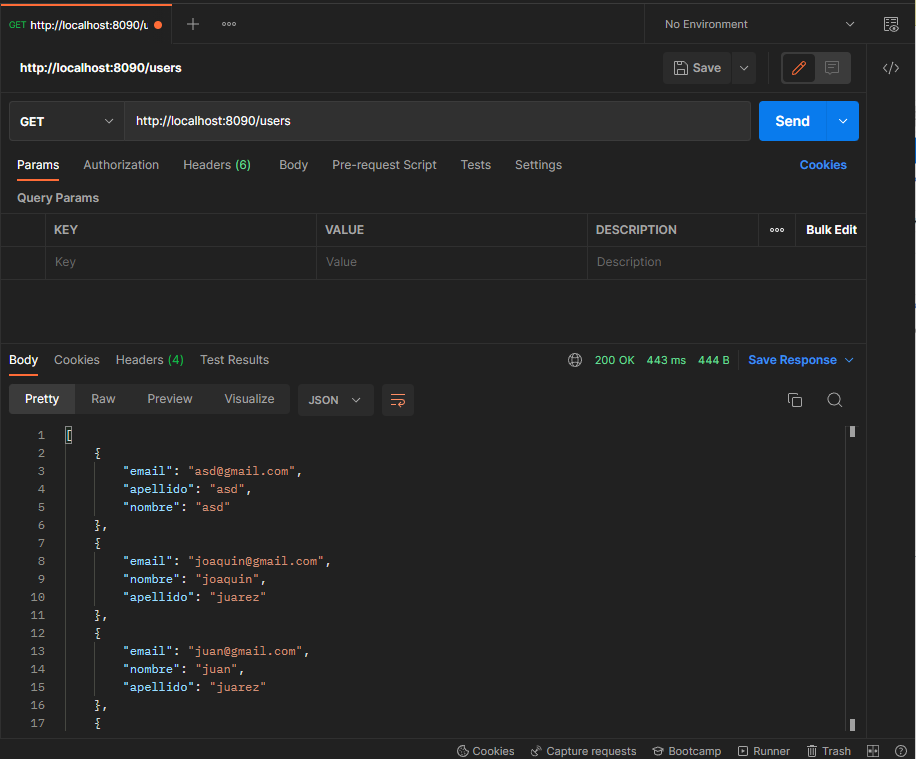
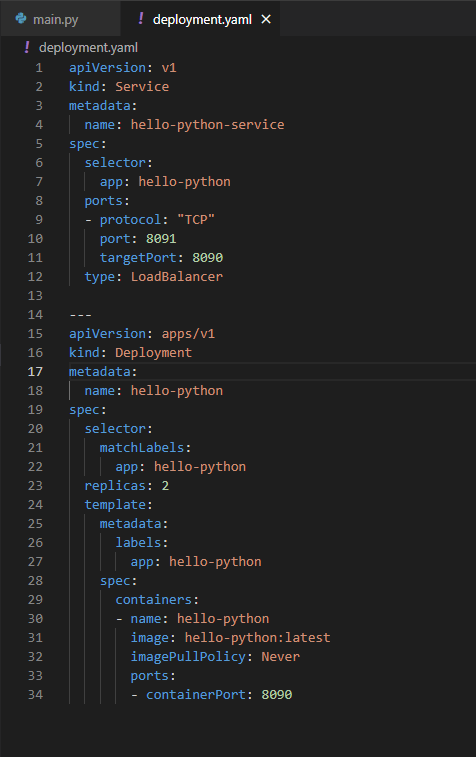
1. Realice la instalación de Docker para contenerizar la aplicación RESTful.
   1. Necesite activar la virtualización de CPU en mi BIOS para que funcionara Docker.
   2. Después de eso agregue el Dockerfile como se especifica en la documentación de fastApi y especificamos el puerto 8090.
   3. 
   4. Agregue el archivo JSON con las credenciales de Firebase a la imagen de Docker ya que caso contrario no se encontrará el archivo al intentar desplegarlo en el contenedor.



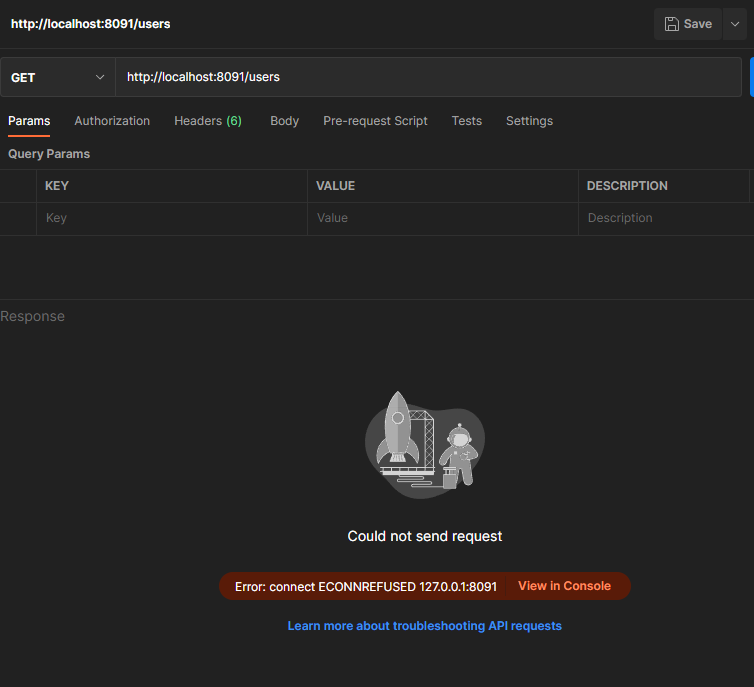
* 1. También se creó el archivo .txt llamado requirements el cual contiene las librerías necesarias para la aplicación.



* 1. Una vez ya teniendo la configuración se corre el comando para crear la imagen. 
  2. Después se corre el comando para iniciar el contenedor especificando el puerto 8090. 

1. Accedemos [localhost:8090/users](http://localhost:8090/users) para probar que funciona nuestra aplicación. 
2. Para la implementación de Kubernetes lo descargue desde su portal y también descargue minikube para poder crear el ambiente local.
   1. Una vez instalado lo necesario corrí el comando para correr el minikube. 
   2. Después cree el archivo yaml con la configuración para kubernetes. 
   3. Después mande el archivo yaml a Kubernetes corriendo el siguiente comando. 
   4. Y con el siguiente comando se comprobó que los pods estuvieran corriendo en kubernetes.



* 1. Al intentar acceder a la dirección a donde apunta el deployment de Kubernetes no da respuesta, aunque los pods estén corriendo. 
  2. Se corrio el comando minikube image load hello-python:latest para intentar cargar la imagen del contenedor a kubernetes y después se creo de nuevo el deployment apuntando directamente a esa imagen. 
  3. El problema siguió estando presente.