**ESTRUCTURA DE DATOS**

**INGENIERÍA MECATRÓNICA**

**ANDERSON JOSEPH OCHOA TRUJILLO**

**JUAN FELIPE VALENCIA SANCHEZ**

**PROYECTO FINAL**

**TO DO LIST CRUD APP**

**LORENA PAOLA VARGAS QUINTERO**

**Introducción**

Durante nuestra trayectoria universitaria, hemos podido observar de cerca una problemática recurrente entre los estudiantes: la dificultad para planificar y administrar su tiempo de manera eficiente. A menudo, se presentan situaciones en las que los estudiantes se sienten abrumados por las múltiples tareas y compromisos académicos que deben cumplir, lo que puede resultar en una disminución de la productividad y un aumento del estrés.

Ante esta situación, hemos decidido emprender un proyecto que pueda ofrecer una solución efectiva a este problema. El proyecto final consistirá en la creación de una aplicación web de lista de tareas Utilizando Django y una librería para la creación de la API llamada rest\_framework, La aplicación permitirá a los usuarios crear nuevas tareas, editar las existentes y eliminar las tareas que ya han completado, tal que podrán .hacer un seguimiento de sus tareas diarias.

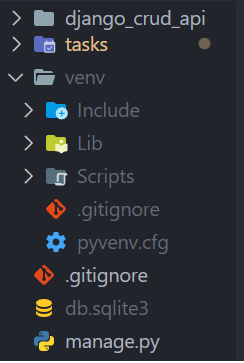
La base de datos usada es SQlite, que es la más recomendada para proyectos pequeños desarrollados en Django.

**Metodología**

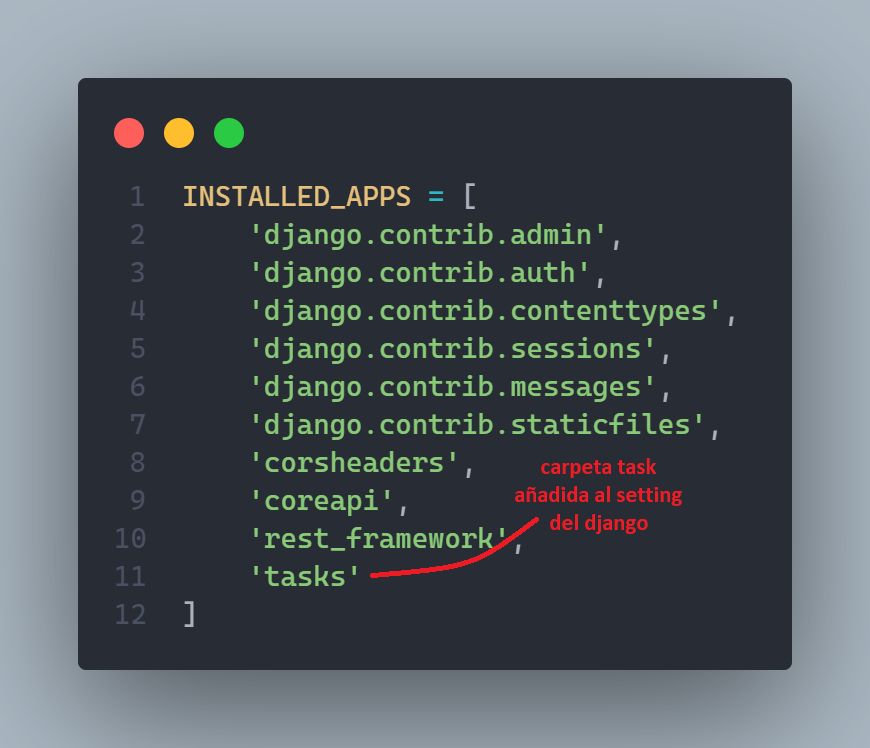
Durante este apartado solo mostraremos el desarrollo del backend, el cual incluye la programación en Django (framework de python basado en la web) y en rest\_framework (framework de Django).

**Inicialización del proyecto:**

1. Creamos nuestro entorno de desarrollo, usamos la tecnología virtualenv, usando el siguiente comando: **python -m venv (nombre que deseemos para nuestra carpeta de entorno).** En nuestro caso luego podremos inicializar el intérprete desde visual studio code, algunos editores de código también tienen esta opción.
2. Instalamos el framework Django con el siguiente comando: **pip install django.**
3. Creamos el proyecto base de nuestra aplicación con el comando: django-admin startproject (nombre del backend, en nuestro caso se llama django\_crud\_api).
4. Creamos nuestra aplicación con el comando: python manage.py startapp (nombre de la app, en nuestro le dimos de nombre tasks)



4.1. Debemos añadir nuestra app tasks al proyecto, para ello añadiremos el nombre de la carpeta en el settings del django\_crud\_api.



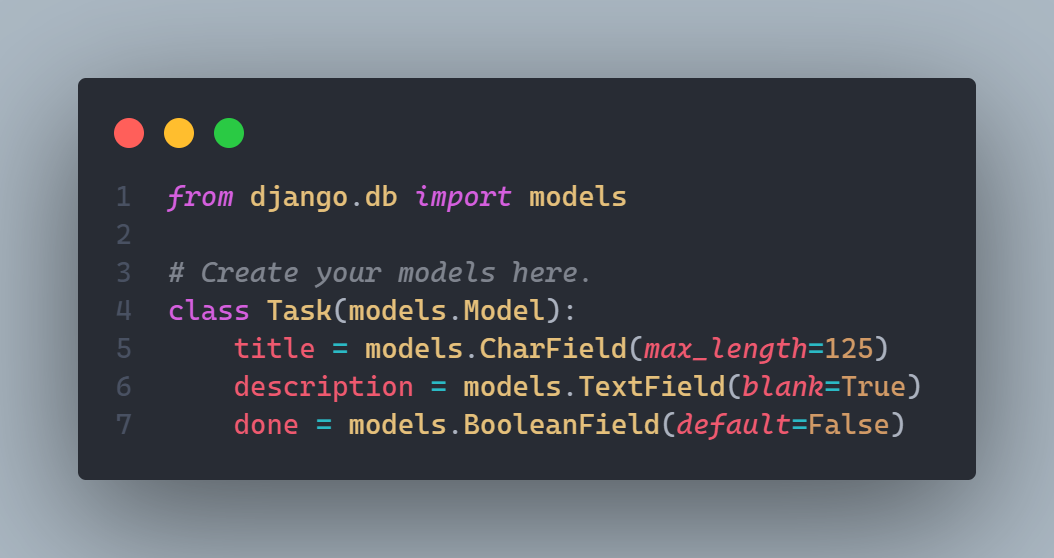
1. Ejecutamos el comando: **python manage.py migrate.** Esto lo que hace es crear todas las tablas para que el proyecto funcione. Usamos esto para evitarnos aprender usar otra tecnología en este caso para manejar las bases de datos, puesto que esto solo extendería el tiempo requerido para desarrollar el proyecto.
2. Ahora debemos instalar todos los módulos o frameworks necesarios para el desarrollo de nuestro proyecto.

6.1. Instalamos el framework de Django llamado rest\_framework con el siguiente comando: **pip install djangorestframework.** La razón por la que debemos descargar este framework de Django es porque con él podemos crear nuestra API, la cual será el puente entre el frontend y el backend de nuestra app.

6.2. El último módulo necesario en nuestro proyecto es django-cors-headers, lo instalamos con el siguiente comando: **pip install django-cors-headers**. Este nos brinda los headers, que son básicamente strings que indican quién puede conectarse a nuestro backend. Más adelante lo usaremos para que nuestro localhost del frontend tenga permitido obtener los datos de la API.

**Modelo de las tareas:**

En nuestra aplicación llamada tasks, hay un archivo llamado models.py, es en este archivo es donde crearemos la clase con nuestra tabla, la cual en este caso contendrá el título, la descripción y un input que nos dará la opción de marcarlo como completado y por defecto tendrá el valor de false (no completado)



Le pasamos el models para que al momento de crear el título, la descripción o el input, podamos indicar que queremos heredar de otra clase (establecida en el framework django) los valores, estos significan los siguiente:

* **CharField:** Para indicar que la propiedad tendrá como valor un string, como parámetro tenemos la opción de indicarle el número de caracteres máximos que pueden tener los títulos, en nuestro caso decidimos indicar como máximo 125 caracteres.
* **TextField:** Para indicar que la propiedad tendrá como valor un espacio para que el usuario escriba describa la tarea. El parámetro blank, es para indicar si el campo es necesario o no, si se deja en True, no será necesario entonces indicar una descripción para las tareas.
* **BooleanField:** Para indicar que la propiedad tiene como valor un input de tipo checkbox, el cual indica que si la tarea está o no está completa. Como parámetro se le indica si viene marcada o no, en este caso el valor fue False, para que inicie desmarcada.

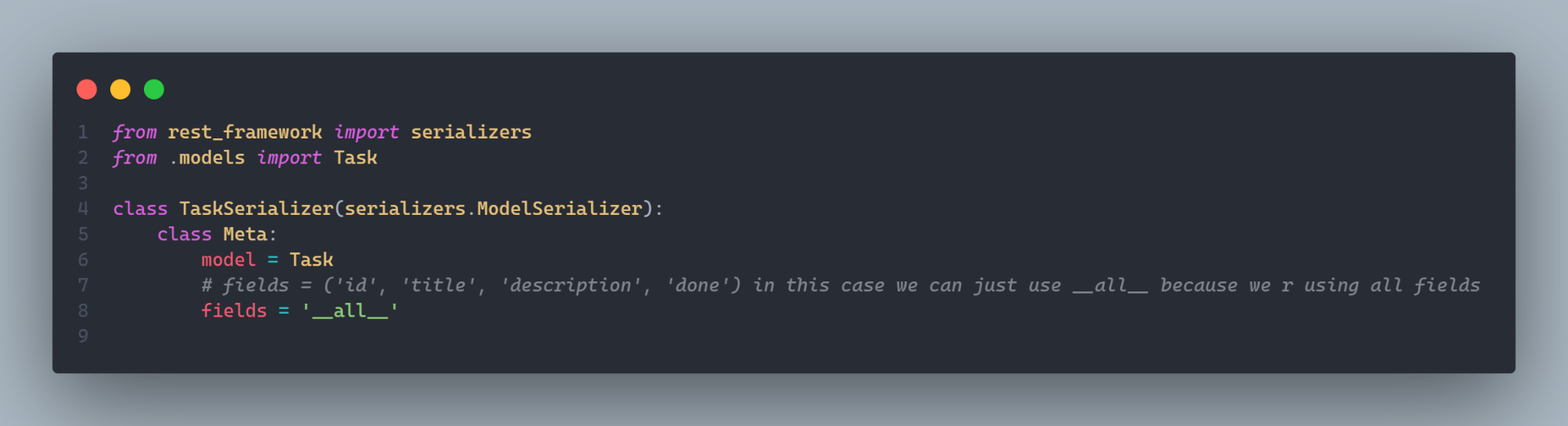
Una vez creado el modelo de tareas, se debe migrar el código para que se cree el código que ejecutará python para añadir los datos a la base de datos. Ejecutamos el comando **python manage.py makemigrations tasks.** Indicamos el tasks para indicarle que queremos hacer la migración sólo de nuestra aplicación. El código creado es el siguiente:



Este código será el que ejecutará Python y lo inyectará en la base de datos con el comando **python manage.py migrate tasks**.

**Creación de la API:**

1. Debemos indicar qué es lo que queremos que se convierta en nuestra API, para ello creamos un archivo llamado serializer.py, en dicho archivo importamos serializers desde el rest\_framework, serializers es un paquete que nos permite seleccionar los campos.

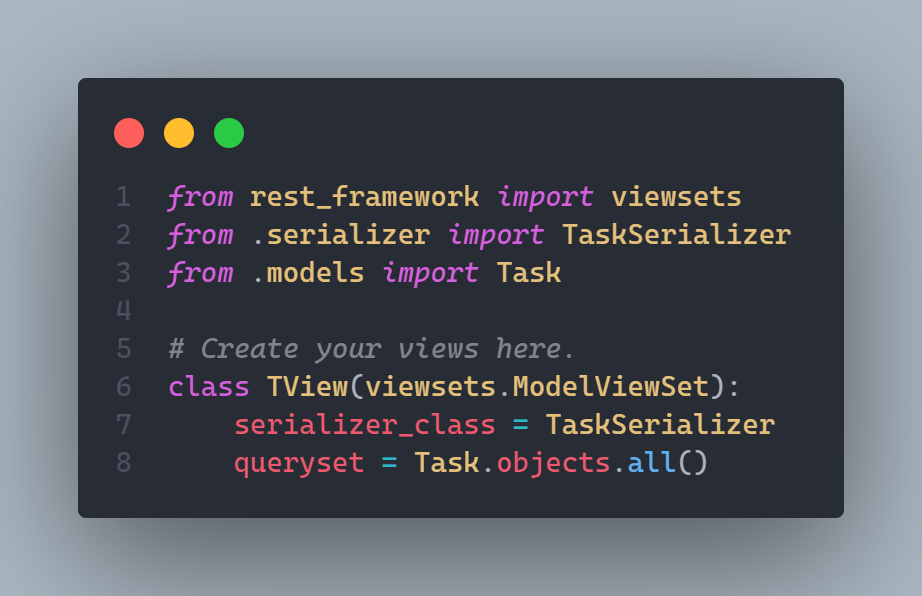


Creamos una función llamada TaskSerializer a la cual llamamos serializers.ModelSerializer.

* El serializer permite convertir los models usados en la sección anterior en tipos de datos nativos de python, para que sean fácilmente renderizados como el JSON que nos interesa generar.
* Con el .ModelSerializer nos proveemos de una forma más fácil de crear una clase serializer con celdas que correspondan al model (en nuestro caso sería el Task, por lo cual lo importamos).

La class Meta es para indicar de dónde vienen los models que usaremos para nuestro crear nuestro JSON y especificar cuáles son.

Django nos permite que con una sola clase podamos generar todos los create, read, update y delete (CRUD) de forma automática.

****

Importamos el viewsets desde rest\_framework, el cual es una clase que va a ser heredada a nuestra clase la cual llamamos TView. Dicha clase heredada proporciona una funcionalidad completa de CRUD para una vista basada en modelos.

En la clase TView, se establecen dos atributos de clase: serializer\_class y queryset.

El atributo serializer\_class establece la clase de serializador que se usará para convertir las instancias del modelo Task en representaciones de datos que se pueden enviar a través de la red. La clase de serializador TaskSerializer debe ser una subclase de serializers.ModelSerializer también proporcionada por Django REST Framework.

El atributo queryset establece la consulta que se utilizará para recuperar los objetos Task del modelo. En este caso, se seleccionan todos los objetos de la tabla Task utilizando el método all().

* Primero, importamos los módulos necesarios path e include desde el módulo django.urls, include\_docs\_urls desde el módulo rest\_framework.documentation, routers y views desde el módulo tasks.
* Luego, creamos un objeto router de la clase routers.DefaultRouter() de Django REST Framework. Usamos este objeto para registrar la vista TView de la aplicación tasks con el nombre de tasks.
* A continuación, definimos las URLs para nuestra API. Primero, creamos una URL para la versión 1 de la API en la ruta api/v1/ que incluye las URLs registradas por el objeto router utilizando el método include(). Luego, creamos una URL para la documentación de la API en la ruta docs/ usando el método include\_docs\_urls().



**Resultados**

No pudimos obtener datos cuantitativos debido a que la aplicación no fue lanzada por lo que datos tales como descargas, número de listas creadas e incluso tiempo de uso promedio, por lo que recurrimos hacer una encuesta, mostrando cómo funcionaba la aplicación desde el usuario de admin, las preguntas y resultados fueron los siguientes:

* ¿Con qué frecuencia utilizas la aplicación para organizar tus tareas?
* ¿Recomendarías esta aplicación a otras personas?
* ¿Qué características te gustaría ver en la aplicación en el futuro?
* ¿Cómo describirías la velocidad y el rendimiento de la aplicación?

Después de realizar una encuesta a 10 estudiantes de nuestra aplicación de organización de tareas, hemos obtenido resultados muy positivos en cuanto a la satisfacción de los usuarios con la aplicación. Los usuarios han indicado que la aplicación les ha ayudado a ser más productivos y a gestionar sus tareas de manera más eficiente.En cuanto a las características que les gustaría ver en la aplicación en el futuro, los usuarios han mencionado la posibilidad de agregar más opciones de personalización, así como la integración con otras aplicaciones de productividad y la capacidad de compartir listas de tareas con otras personas.

**Conclusiones**

A pesar de que no se pudieron obtener datos cuantitativos debido a que la aplicación no había sido lanzada, los resultados de la encuesta que realizamos a los usuarios de nuestra aplicación de organización de tareas han sido muy positivos. Los usuarios han destacado la utilidad y eficacia de la aplicación para gestionar sus tareas y han manifestado su satisfacción con respecto a su funcionamiento y rendimiento. Además, los comentarios constructivos que han proporcionado los usuarios nos han permitido conocer sus necesidades y expectativas, lo que nos dará la oportunidad de mejorar la aplicación y hacerla aún más útil y atractiva para los usuarios. En general, los resultados de la encuesta nos han proporcionado una idea clara de la percepción de los usuarios hacia nuestra aplicación y nos han motivado a seguir trabajando para ofrecer una herramienta útil y eficaz para la gestión de tareas y la productividad.