**IBISA Integrator**

Concepto:

IBISA Integrator es un componente que expone datos del negocio bajo una interfaz SQL para garantizar la compatibilidad con las herramientas tradicionales de BI, análisis de datos ad-hoc, integraciones, etc.

Integrator tiene algunas primitivas importantes a tener en cuenta para entender el componente:

* Los suscriptores pueden definir sus modelos de datos a su conveniencia y según sus casos de uso. Dichos modelos de datos se denominan **Business Models**.
* Los suscriptores suben a integrator esos modelos de datos en formato .sql e Integrator se encarga de crear la Base de Datos con sus tablas y el usuario con privilegios de acceso únicamente a esa Base de Datos
* Integrator ejecutará de forma periódica (según job\_frequency) un ETL (job) asociado a ese modelo de datos
* Los suscriptores deben construir y subir a Integrator el job capaz de alimentar ese modelo de datos bajo el esquema de ejecución periódico descrito, dándole la inteligencia necesaria para hacer UPDATEs e INSERTs donde corresponda, recibir el token para consultar a los componentes de IBISA, recibir fecha de inicio y de fin y cualquier otra medida para garantizar la veracidad, disponibilidad e integridad de los datos que se escriban en el Business Model

Estructura del proyecto:

La estructura de archivos es en un 90% lo default de un proyecto de django. Cada archivo tiene sus respectivos docstrings que describen el detalle de lo que hace cada función. De manera general, se estructura de la siguiente manera:

|  |  |
| --- | --- |
|  | * En la carpeta integrator/ se encuentran los archivos del proyecto (según django). Aquí se configuran la URL base (urls.py) y la configuración general de cómo debe correr el proyecto (settings.py) * En integratorapi/ se encuentran los archivos específicos de la aplicación (según django) * El archivo integratorapi/urls.py contiene las rutas completas de los recursos RESTful que expone la aplicación * En integratorapi/views.py se encuentran las funciones APIView que manejan la lógica de las rutas * En integratorapi/serializers.py se encuentran los serializadores que son utilizados por los APIViews para serializar los datos de las instancias de los modelos * En integratorapi/models.py se encuentran definidos los 2 modelos de datos que usa Integrator para su funcionamiento. BusinessModels y BusinessModelExecutionHistory (histórico de ejecución) * En integratorapi/utils.py se encuentran diferentes funciones auxiliares que son invocadas en diferentes partes el proyecto para uso específico. * En integratorapi/sqlfileparser.py se encuentran las funcionalidades necesarias para parsear/recorrer los archivos .sql que suben los suscriptores * En requirements.txt se encuentran los requerimientos base del proyecto para poder correr. * En el archivo .env se encuentran las variables de configuración que necesita la aplicación para funcionar correctamente |

Componentes:

A screenshot of a cell phone

Description automatically generated

1. API:

Set de recursos RESTful construidos sobre django-rest-framework para interactuar con las capacidades del componente

1. Scheduler (ver documento específico del Scheduler para más info):

Daemon que está verificando constantemente los Jobs que tiene que ejecutar. Se encarga de ejecutarlos según el job\_execution\_command cuando corresponda y de almacenar en la BD el output de ejecución del mismo

1. Motor de Base de Datos MySQL:

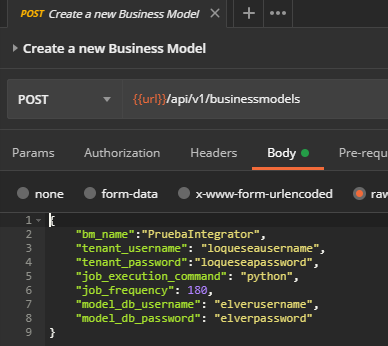
Instancia de MySQL en donde se almacena la base de datos de administración así como las bases de datos de los modelos de negocio que se vayan creando

1. ETLs:

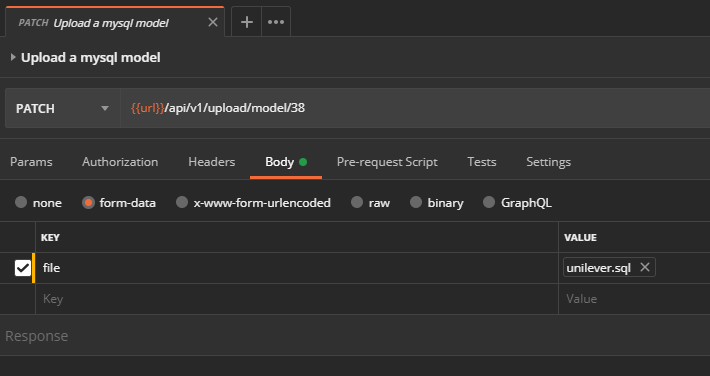
Jobs que se ejecutan cada job\_frequency minutos. Deben estar construidos recibiendo el token, fecha desde, fecha hasta y el string de conexión a la Base de Datos del Business Model.

**Caso de Uso: Paso a paso de creación de un nuevo Business Model**

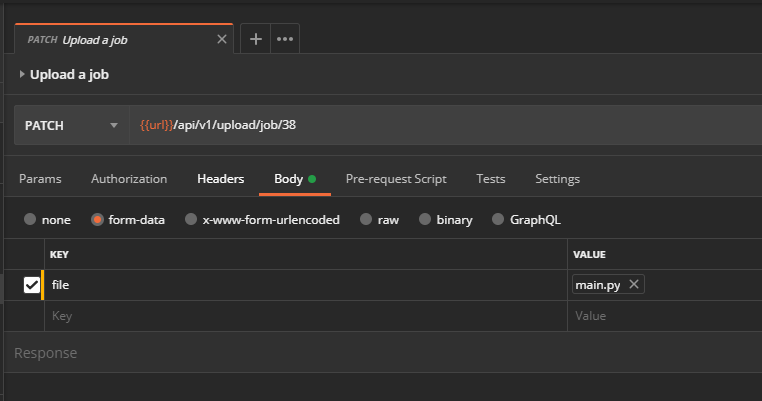
1. Hacer un POST a /api/v1/businessmodels del API para crear un nuevo registro del Business Model. Por ejemplo:



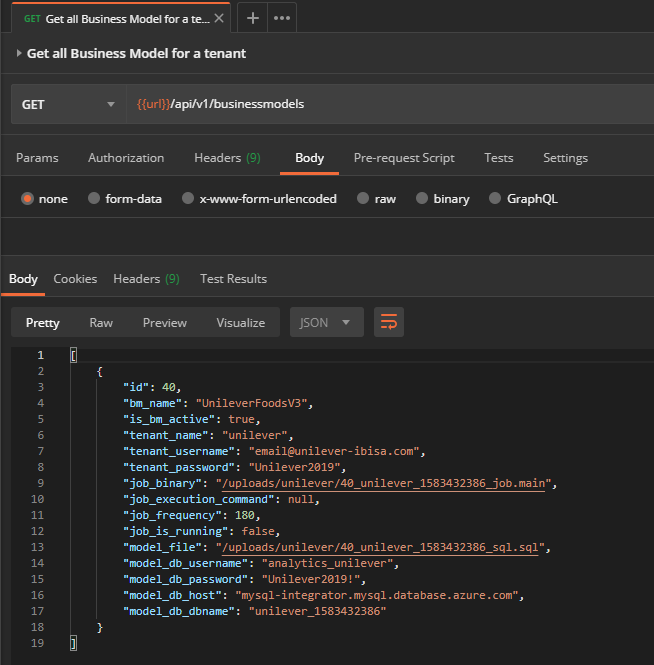
1. Después de tener el Business Model, tomar nota de su id para proceder a subir el .sql y el job correspondiente
2. Subir el .sql, hacer un PATCH a /api/v1/upload/model/:bm\_id con un el archivo correspondiente como form-data



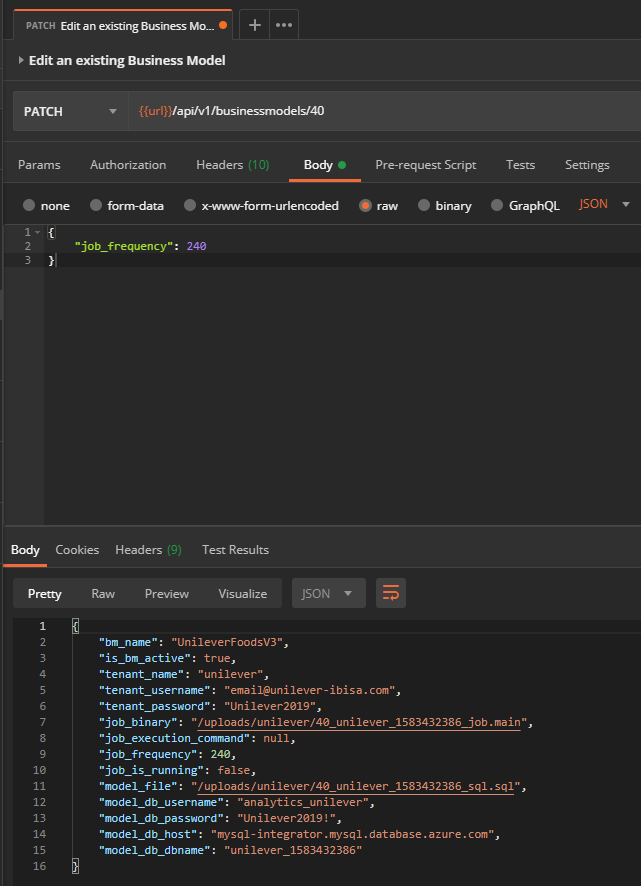
1. Subir el job que alimenta al modelo, haciendo un PATCH a a /api/v1/upload/job/:bm\_id con el archivo correspondiente como form-data:



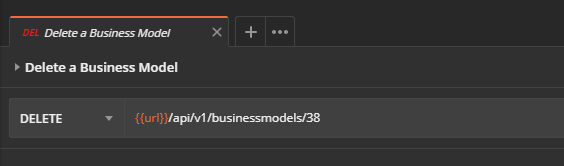
1. Hacer un GET a /api/v1/businessmodels para asegurarse que el Business Model que se está manipulando está activo y no le falta nada:



1. De faltarle algo, hacer un PATCH a /api/v1/businessmodels/:bm\_id y editar el valor del campo correspondiente:



1. Por último, para eliminar un Business Model, hacer un DELETE a /api/v1/businessmodels/:bm\_id:



El API se encargará de borrar la Base de Datos y el Usuario correspondiente.

**Referencias**

Las guías que se usaron para la configuración de la infraestructura y en la estructura de las pruebas fueron:

* <https://realpython.com/testing-in-django-part-1-best-practices-and-examples/>
* <https://www.django-rest-framework.org/api-guide/testing/>
* <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing>
* <https://certbot.eff.org/lets-encrypt/ubuntubionic-nginx>
* <https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-18-04>