Proyecto

Bolsa de Valores



**Índice**

Crear proyecto en Node………………………………………………………………………………………………….........................4

Instalación de librerias………………………...…………………………………………………………………………..........................4

Crear servidor web……………………….……...………………………………………………………………………….........................5

Fichero de entorno .env……………….……...………………………………………………………………………….........................6

Extensión REST-Client….……………….……...………………………………………………………………………….........................7

Definir rutas (API)...…..….……………….……...………………………………………………………………………….........................8

Crear proyecto en Heroku…...……….……...…………………………………………………………………………......................10

Crear punto de entrada a la app (mongoose)……...…………………………………………………………......................12

Crear modelo………………………………………………..……...…………………………………………………………......................13

Petición save()..……………………………………………..……...…………………………………………………………......................14

Configuración Robo3T con Heroku….…………..……...…………………………………………………………......................14

Peticiones GET, POST………………………..…………..……...…………………………………………………………......................16

Validaciones (Express-Validator)..........…………..……...…………………………………………………………......................20

Crear proyecto en Node.js

Nos situamos en el directorio en el que queremos guardar el proyecto

En caso de que no queramos rellenar los campos que nos aparecen más abajo, iniciaríamos el proyecto con



Si queremos crear los detalles:



Nos aparece el asistente para ponerle nombre al proyecto



Añadimos una descripción



El entry point o fichero de entrada siempre va a ser “index.js”



Si abrimos el directorio con VSC vemos que nos ha generado un fichero package.json

**Instalar librerias**

**EXPRESS JS**

Usamos –save para que lo guarde directamente como dependencia en el proyecto, de no hacerlo, no afectaría al funcionamiento pero lo guarda en el apartado “devDependendecies” y cuando queramos subirlo a alguna plataforma no la leerá correctamente.



**MONGOOSE**

Interfaz para interactuar con mongoDB



**VALIDATOR**

Nos permite hacer validaciones para dar el ok a los datos que recibo por parte del usuario.



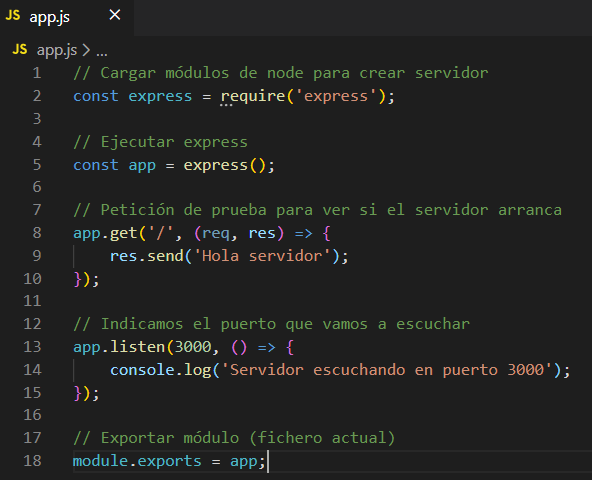
**Crear el servidor web**

Creamos fichero app.js

En él vamos a cargar express, recibimos las peticiones, cargamos las rutas, utilizamos los middlewares, etc.

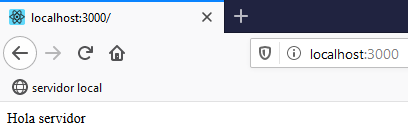


Empezamos por requerir Express y añadimos una petición junto con el puerto que vamos a usar



Probamos que funcione



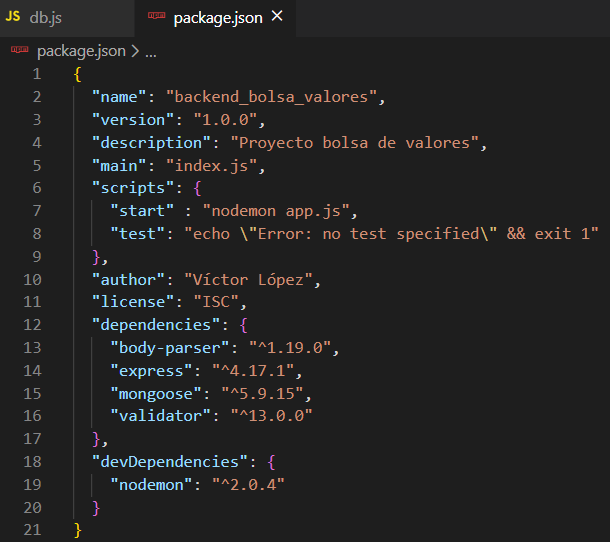


**NODEMON**

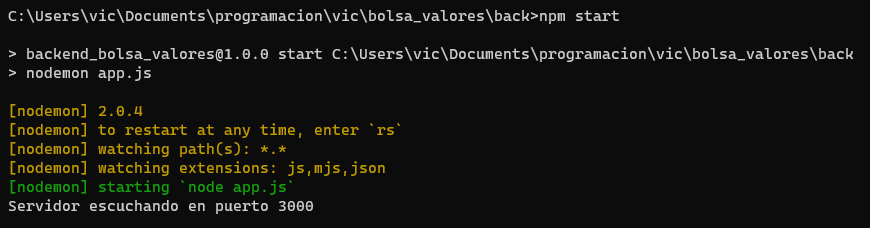
Nos permite que cada vez que se hace un cambio en el proyecto, reinicia el servidor automáticamente. Lo instalamos con “—save-dev” para que se agregue en “devDependencies” porque se instala a nivel local, si se sube a desarrollo nodemon no funcionará.



En package.json añadimos en “scripts” el punto de entrada para que se inicie automáticamente.



Probamos que arranca

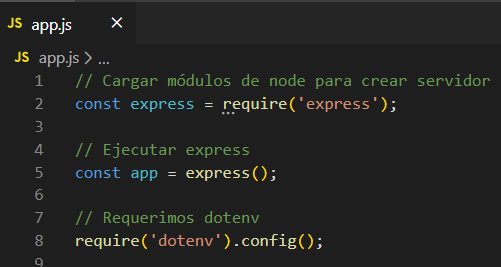


**Fichero de entorno**

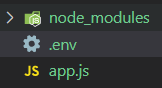
Instalamos la librería **dotenv** para leer los ficheros **.env**



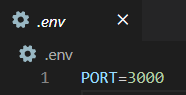
Lo requerimos en **app.js**



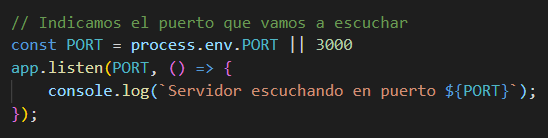
Creamos el fichero **.env** que se usa para guardar unas variables y desde la app acceder a ellas



Definimos la variable que vamos a utilizar

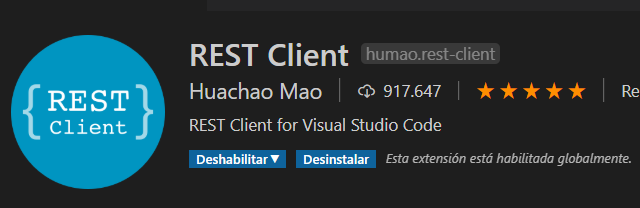


Ahora en el puerto que utilizamos podemos obtenerlo desde el fichero **.env**

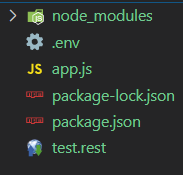


**Extensión REST-Client**

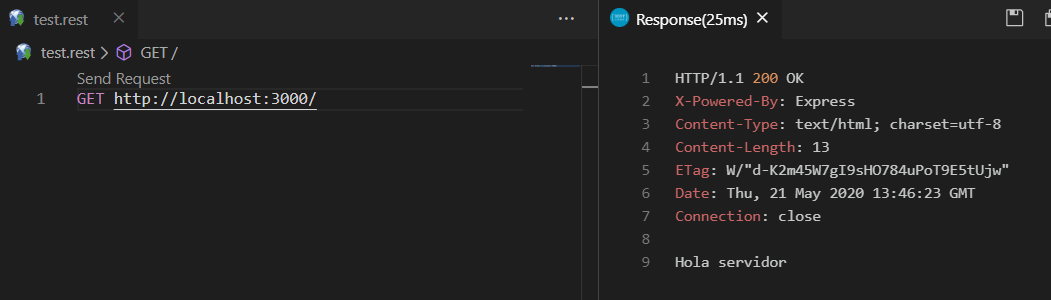
Para poder hacer peticiones y visualizarlas desde VS Code, instalamos la extensión REST-Client



Creamos el fichero **test.rest**

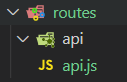


Definimos las peticiones



**Definir las rutas**

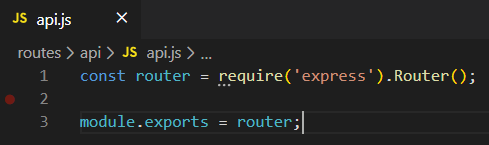
Creamos la carpeta **routes** dentro la carpeta **api** y el fichero **api.js**



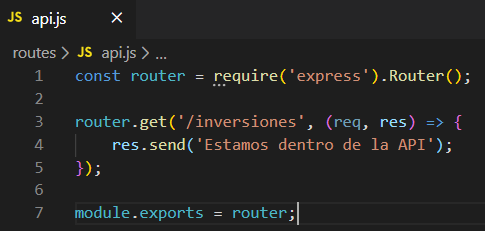
En **app.js** requerimos el **apiRouter** para redirigir todas las rutas que contengan **/api**



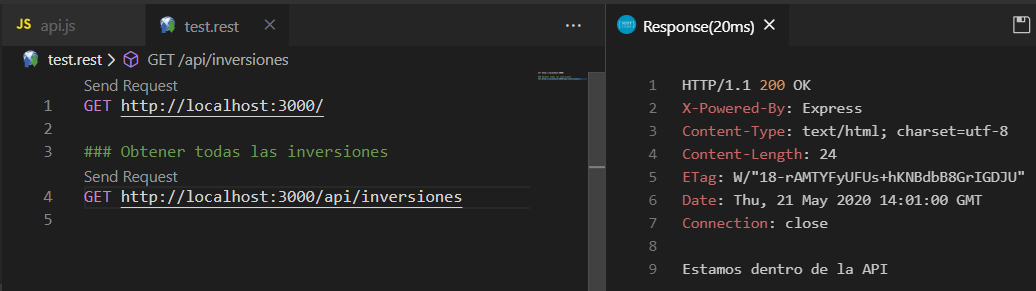
Definimos los manejadores



Dentro de los manejadores podemos definir las rutas

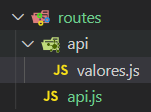


Probamos que funcione

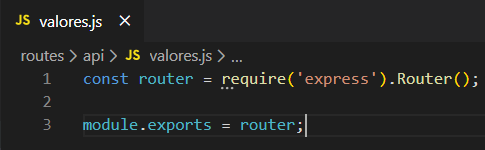


Para separar las rutas que usaremos, en la carpeta **api** dentro de **routes** definimos los diferentes grupos.

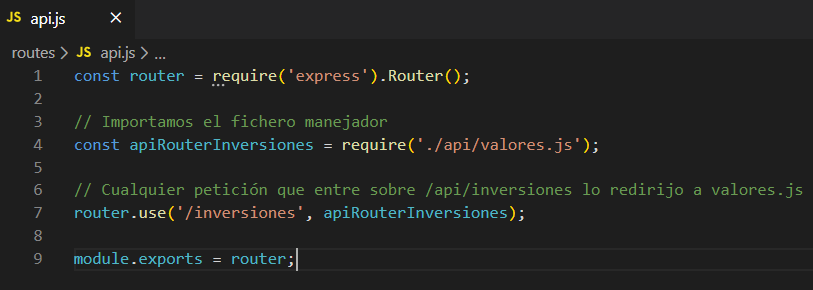
Creamos el fichero **valores.js**



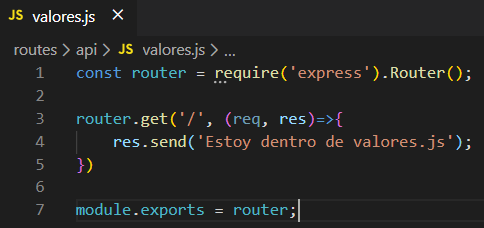
Como es un manejador de ruta o definimos igual que el **api.js**



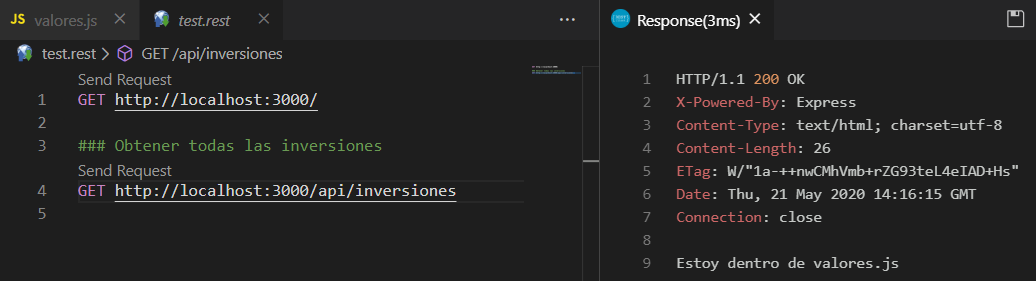
Ahora desde **api.js** delegamos la gestión de las rutas **/api/inversiones** a **valores.js**



En **valores.js** hacemos la petición

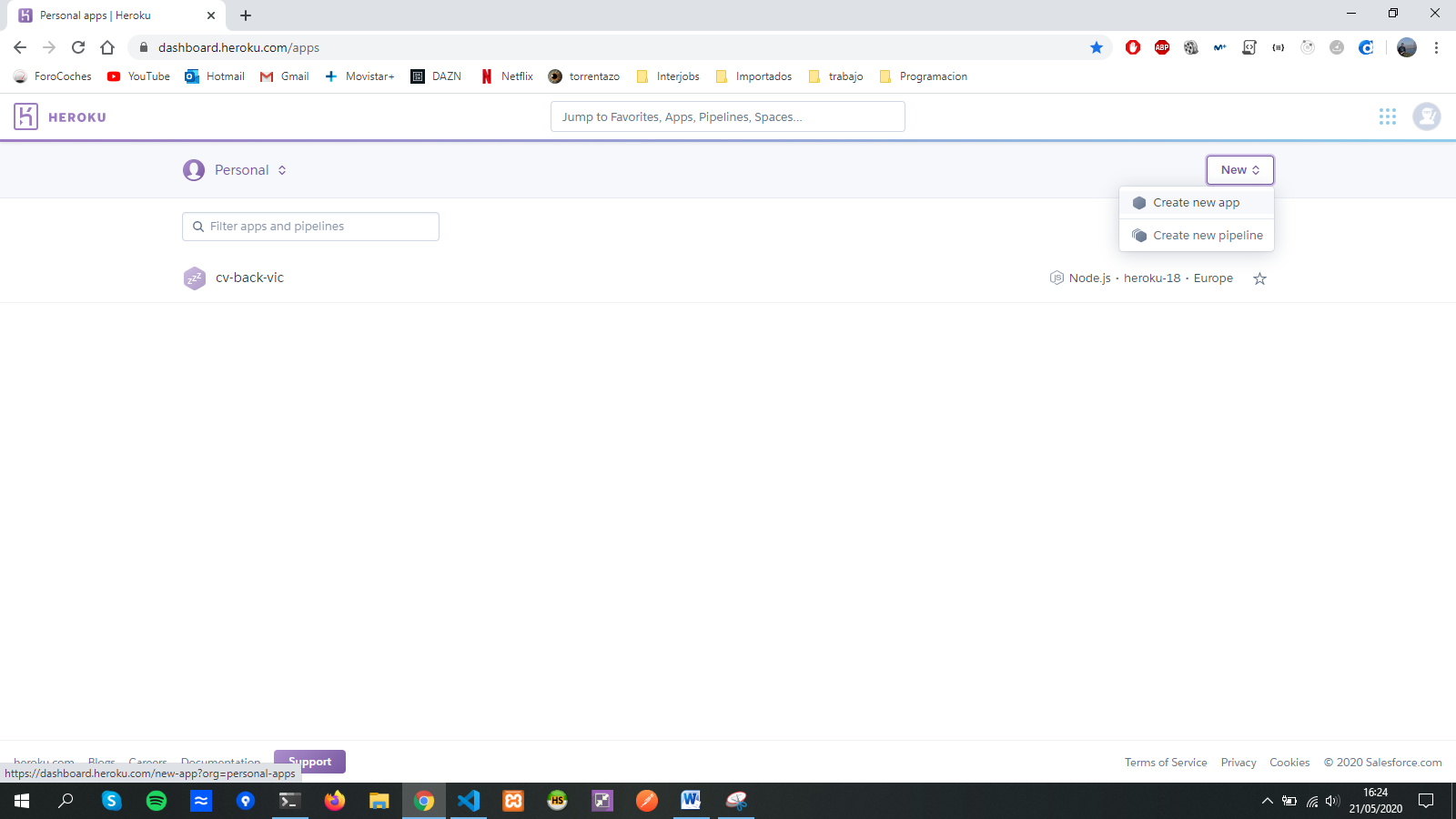


Probamos que funcione haciendo la petición a **/api/inversiones**

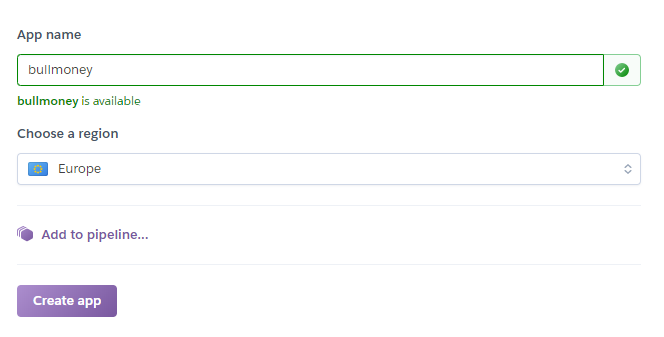


**Crear proyecto en Heroku**

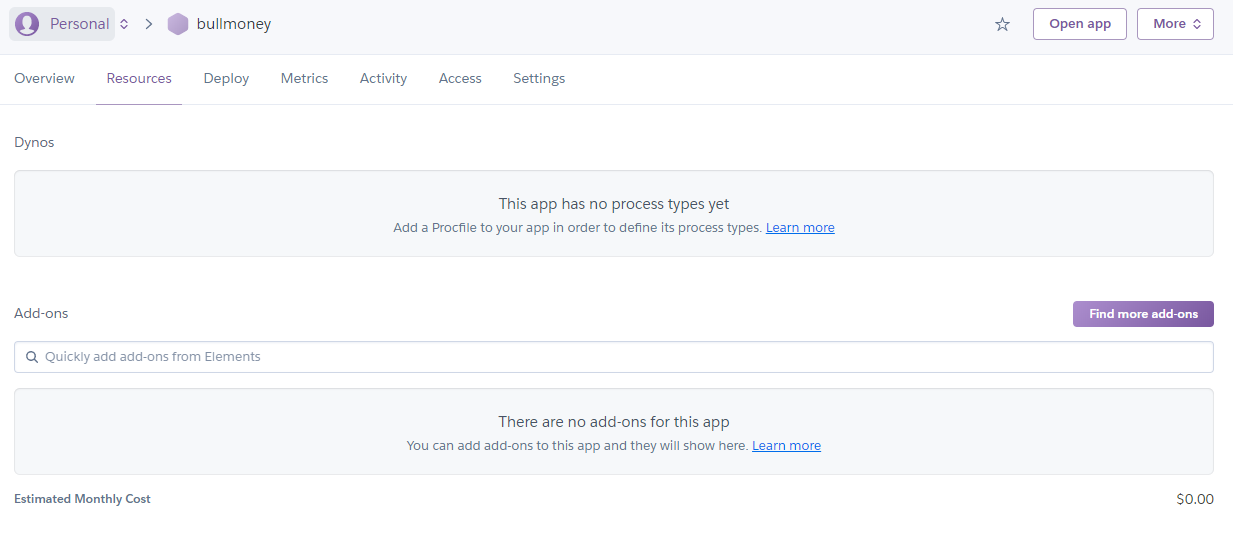
Creamos un proyecto nuevo



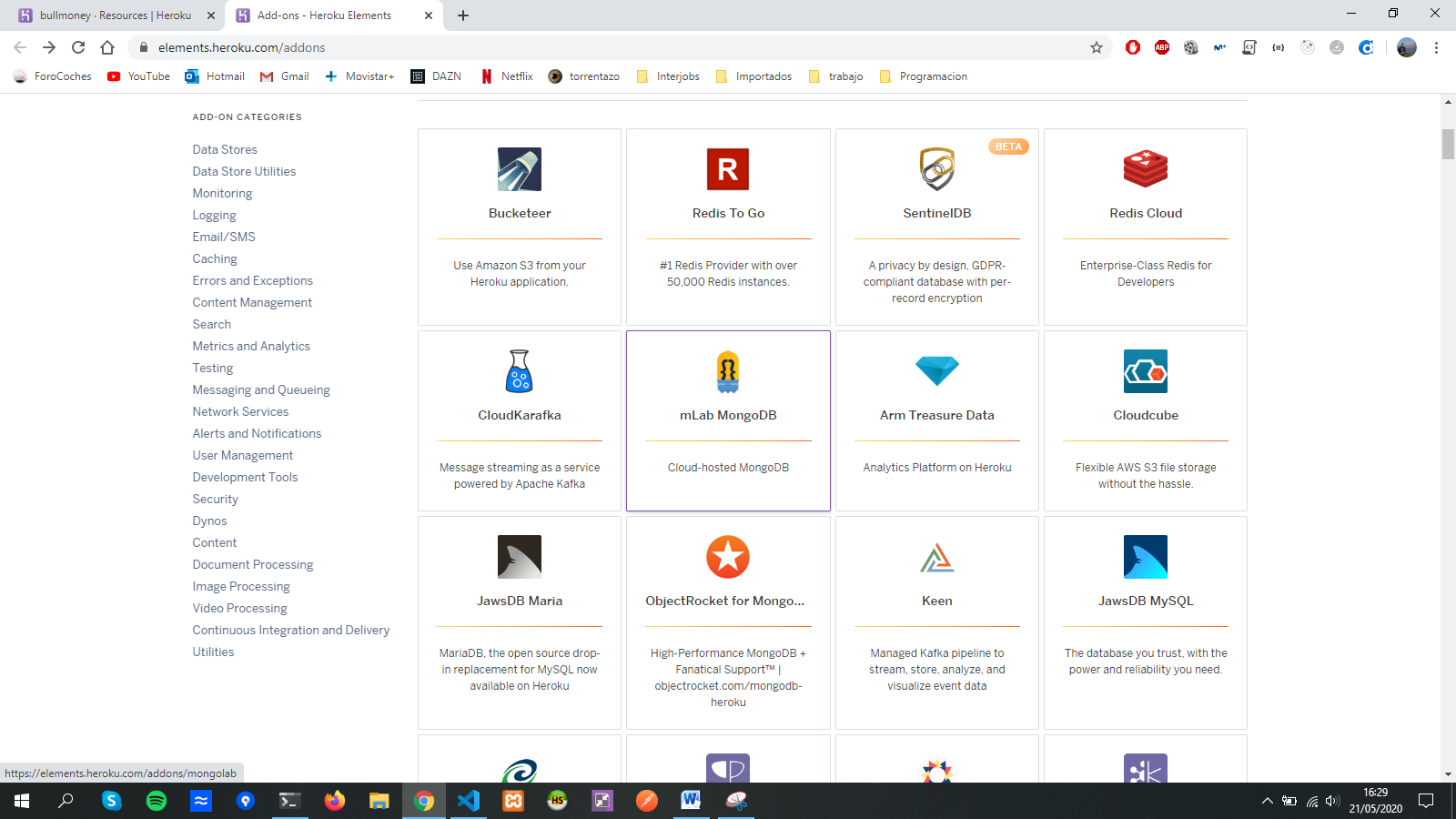
Aplicamos un nombre al proyecto



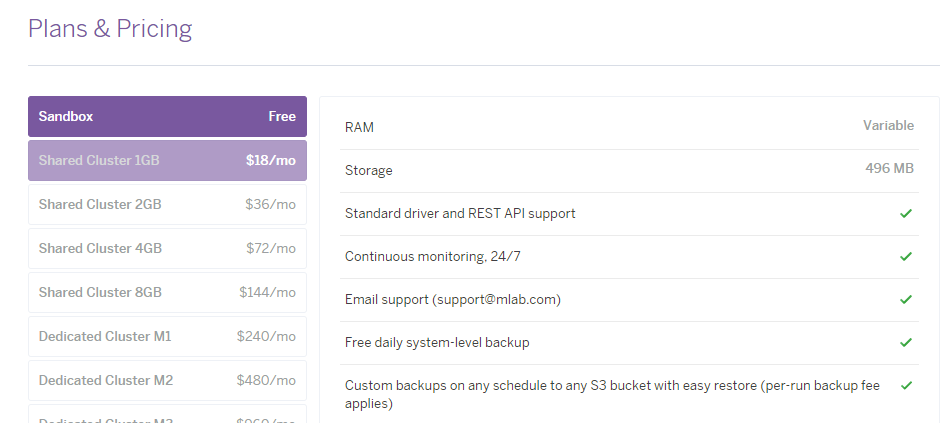
Una vez creada la app vamos a **Resources** para añadir un **addon**

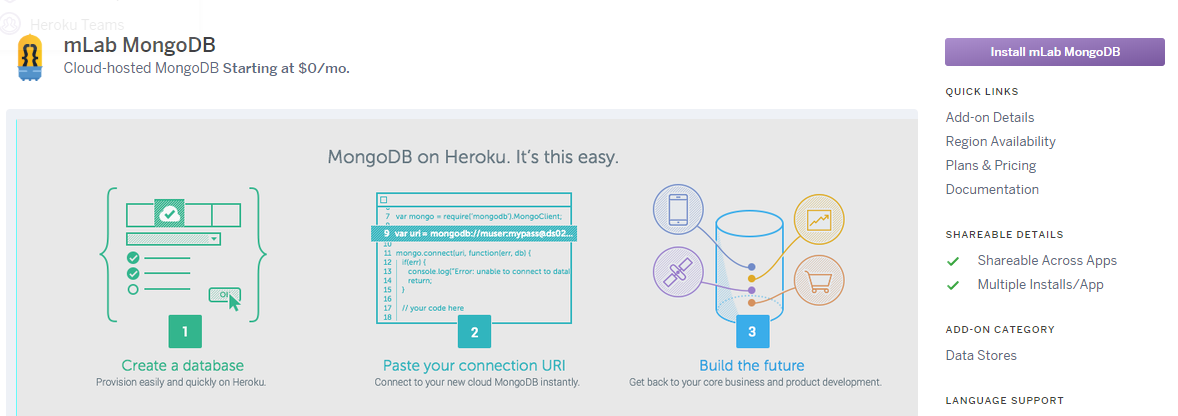


Seleccionamos **mLab MondoDB**

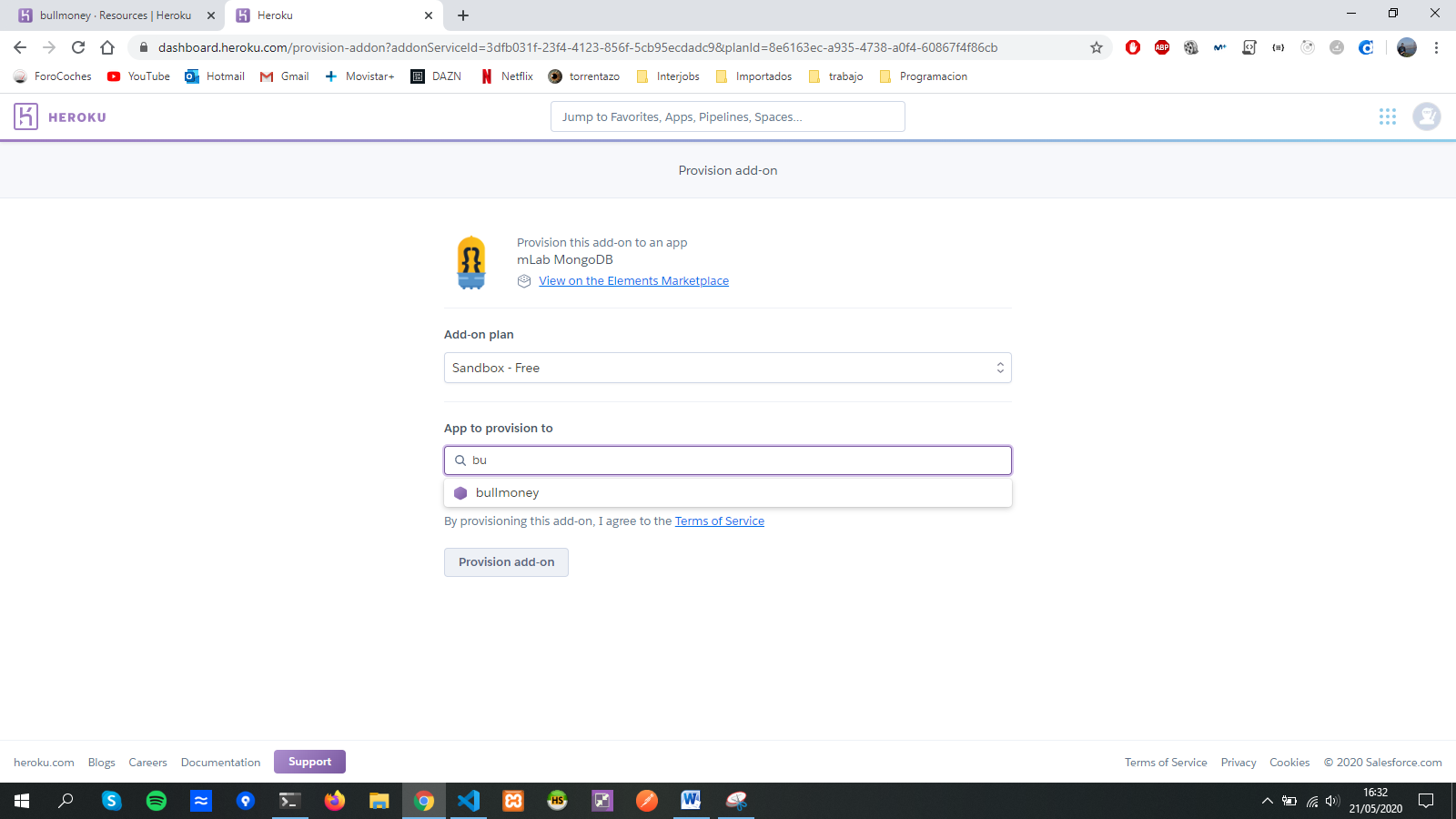


La añadimos a nuestro proyecto, cuidado al seleccionar el plan porque puede tener coste

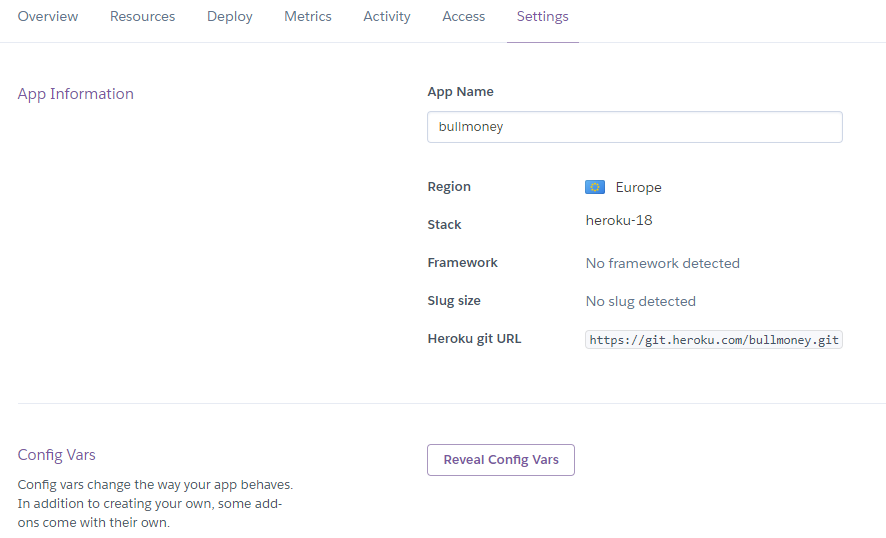




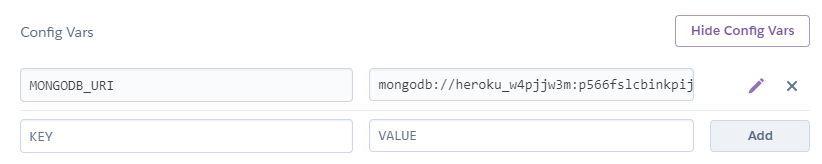
Seleccionamos el proyecto en el que queremos añadirlo



En **settings** podemos ver la configuración y en **Reveal Config Vars** accedemos a la BBDD

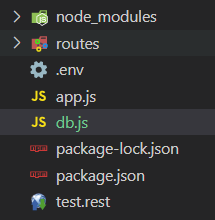


Aquí vemos la configuración junto con el link para conectarnos a ella

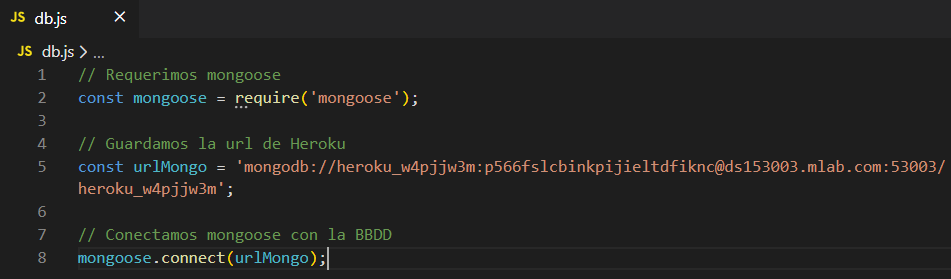


**Crear punto de entrada a la app**

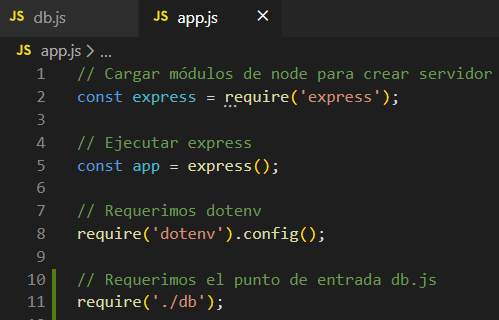
Creamos el fichero “db.js” para tener la conexión con la base de datos.



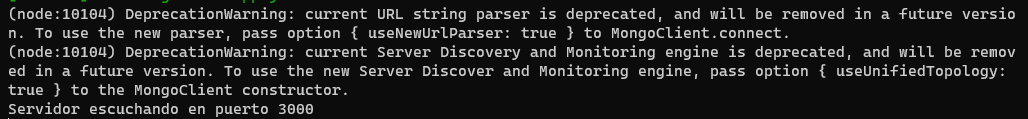
Para hacer la conexión con la base de datos, mongoose va a recibir por parámetro la url que la obtendremos de Heroku



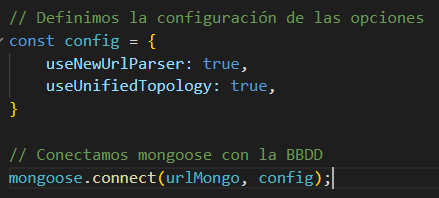
En **app.js** requerimos el fichero **db.js**



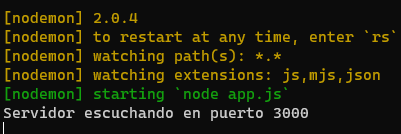
Probamos que funciona y nos devuelve este error, esto se debe a que hay funciones de mongoose que están deprecadas, por lo que hay que pasarle unas opciones para usar las nuevas funcionalidades que suplen a las antiguas, en el mismo error nos indica que debemos hacer para arreglarlos.



En **db.js** añadimos la configuración y también lo añadimos como parámetro a la conexión de mongoose

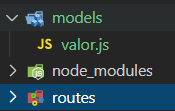


Comprobamos que ya no aparecen más errores

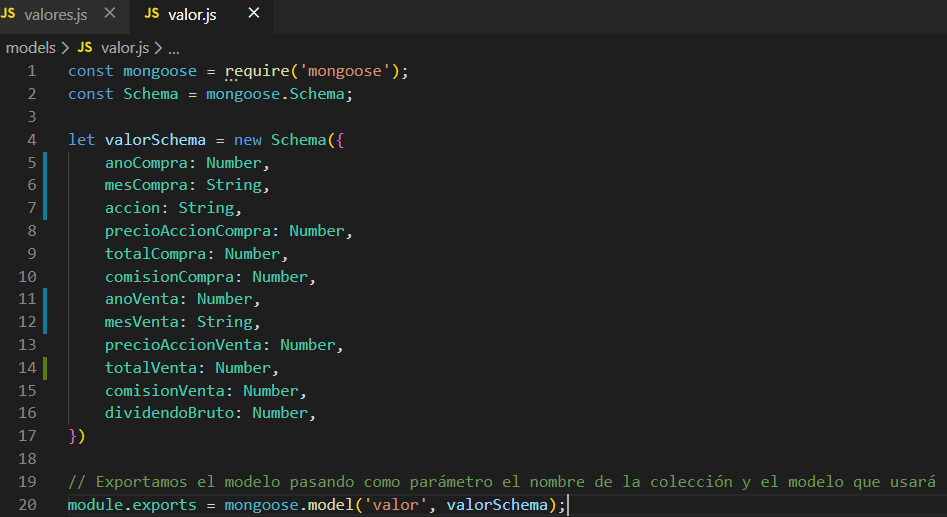


**Crear modelo**

Creamos la carpeta **models** y dentro el modelo de **valores.js**

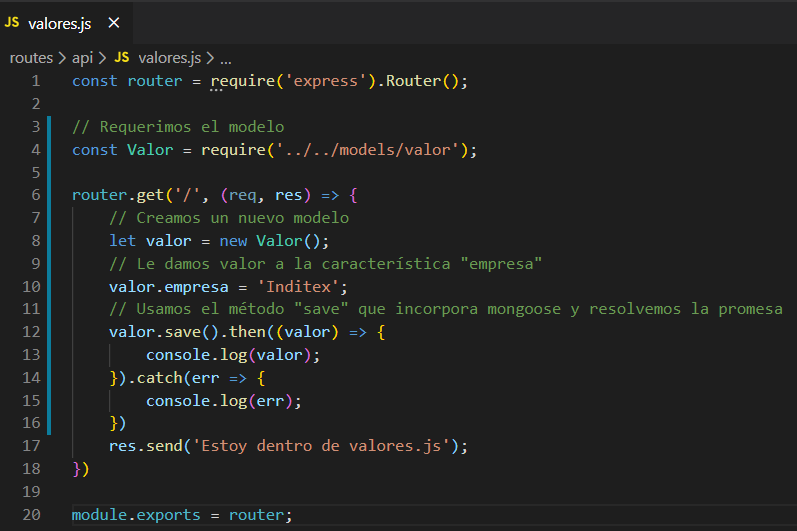


Definimos el **Schema** de mongoose para los datos que va a contener el modelo, al exportarlo indicaremos como parámetro el nombre de la colección (siempre en singular) y el modelo que usará para crearla

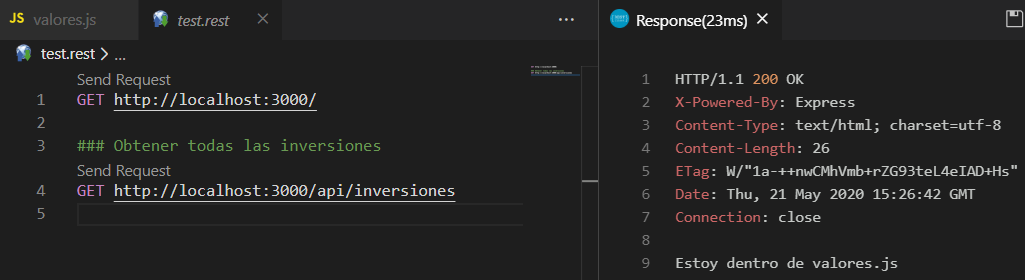


**Petición Save()**

En **valor.js** importamos el modelo e intentamos guardar una de las características que contiene



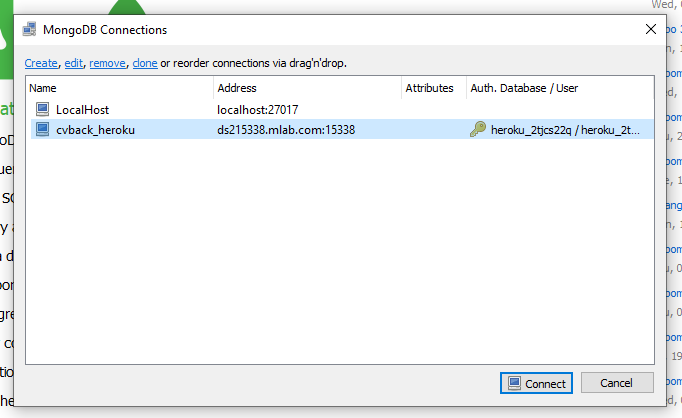
Probamos que funcione lanzando la petición



Como nos ha devuelto el mensaje que teniamos definido, comprobamos la base de datos para comprobar que el valor se haya guardado correctamente, para ello utilizamos **Robo3T**

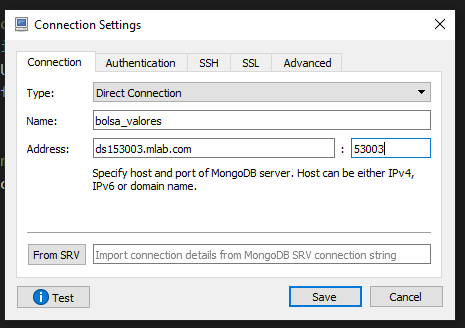
**Configuración Robo3T con Heroku**

Para conectar **Robo3T** con la bbdd de **Heroku** creamos una nueva conexión



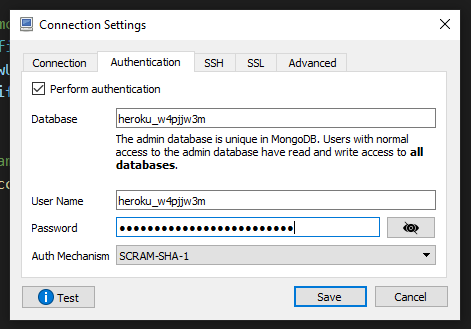
Usamos los datos de la url de **Heroku**



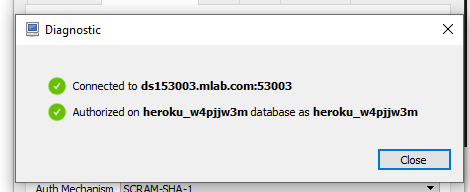


En la autenticación

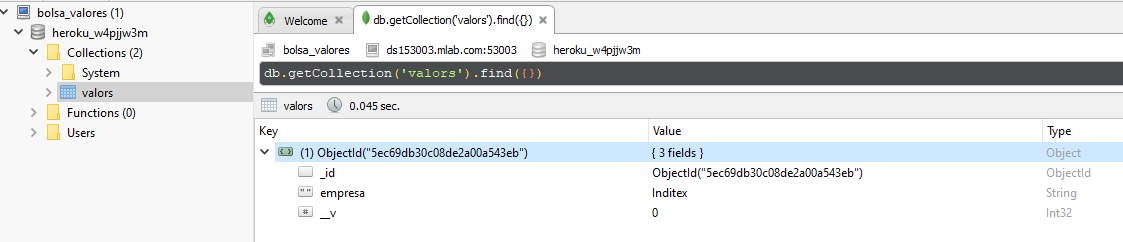




Hacemos el test para comprobarlo



Entramos en la bbdd y abrimos la colección **valors**

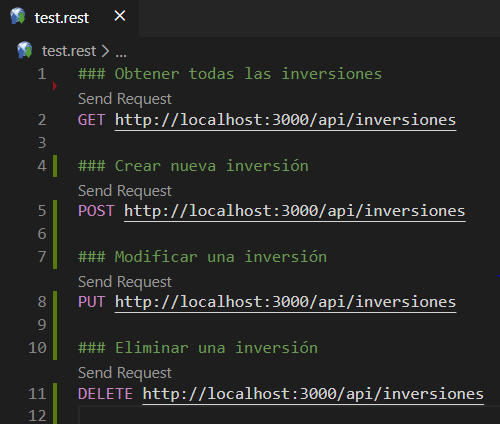


Además en el console.log que tenemos nos aparece así en la consola

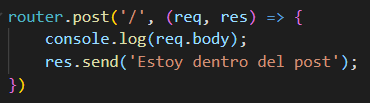


**Peticiones GET, POST, PUT y DELETE**

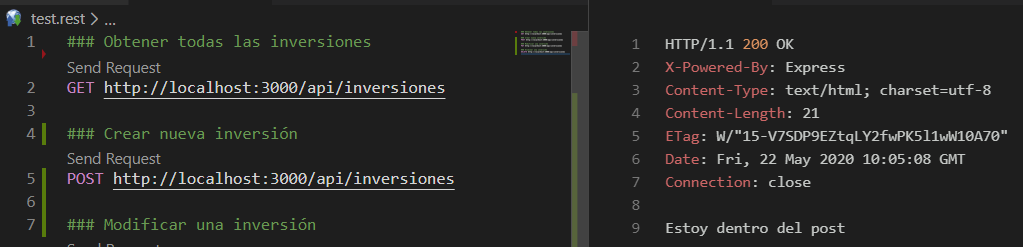
Preaparamos **test.rest** con las diferentes peticiones que vamos a realizar



En **valores.js** definimos la petición **POST**

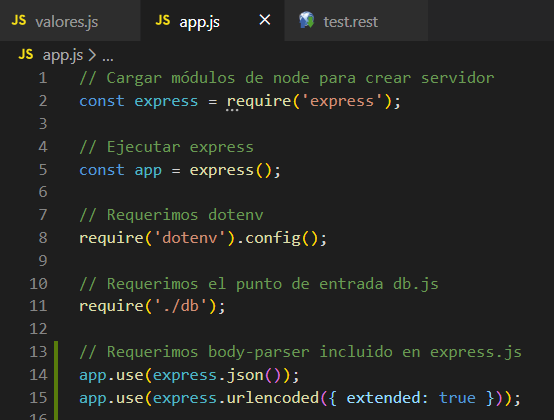


Probamos que funcione

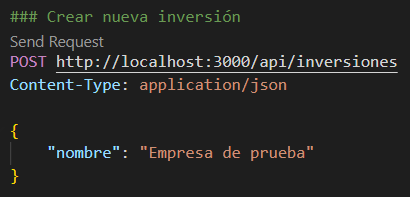


Para manejar la solicitud **POST** en **Express.js** version 4 y superior, necesitamos usar el middleware **body-parser**, lo que hace es extraer toda la parte del cuerpo de una secuencia y nos la almacena en **req.body**, a partir de **Express.js v.4** no hace falta instalar **body-parser** porque lo trae incorporado

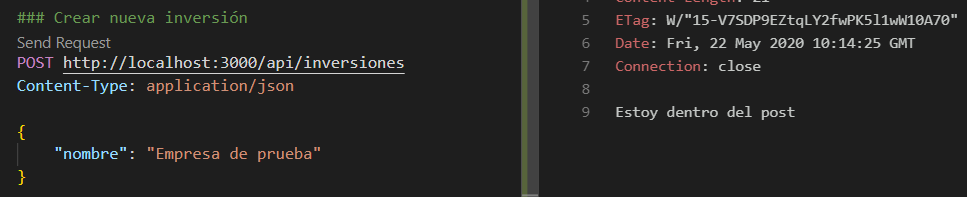
Para poder usarlo, lo configuramos en **app.js**



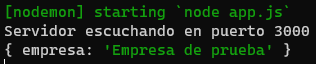
Adjuntamos una cabecera en la petición **POST**



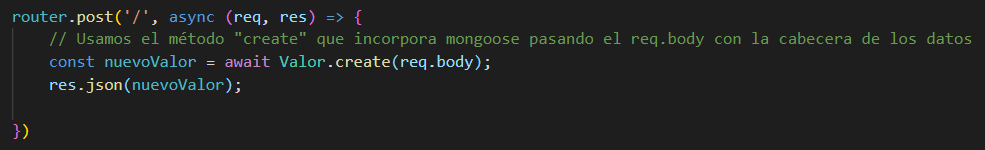
Probamos que envia correctamente la petición



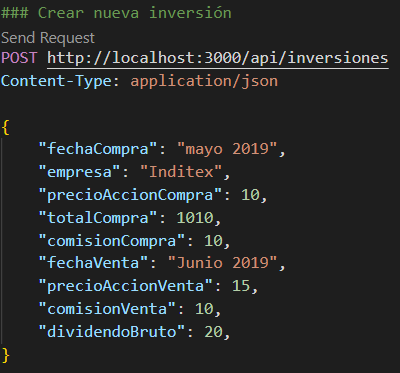
Como hicimos anteiormente un **console.log(req.body)**, en la consola nos debe aparecer la cabecera que hemos definido



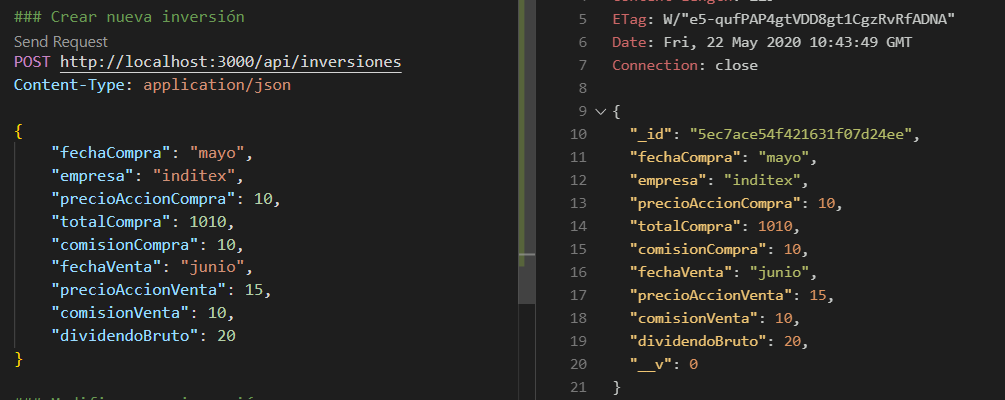
Para crear un nuevo **valor** usamos el método **create()** que incorpora **mongoose** en el que le pasamos como parámetro los datos de la cabecera



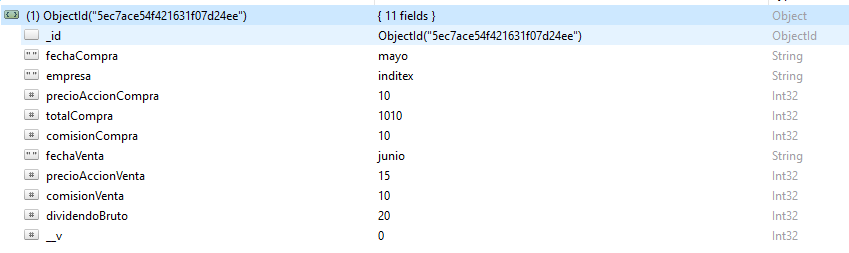
Creamos en la cabecera los campos de nuestro modelo



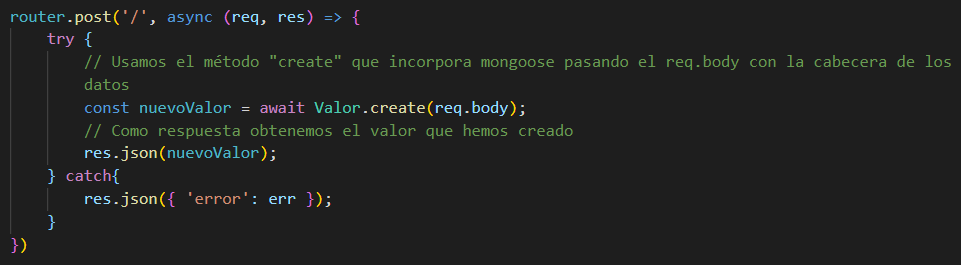
Probamos que nos crea el **valor** en la colección al hacer la petición



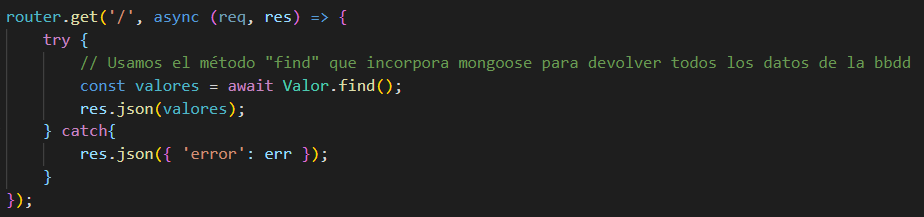
Comprobamos que se ha creado en la bbdd en **Robo3T**



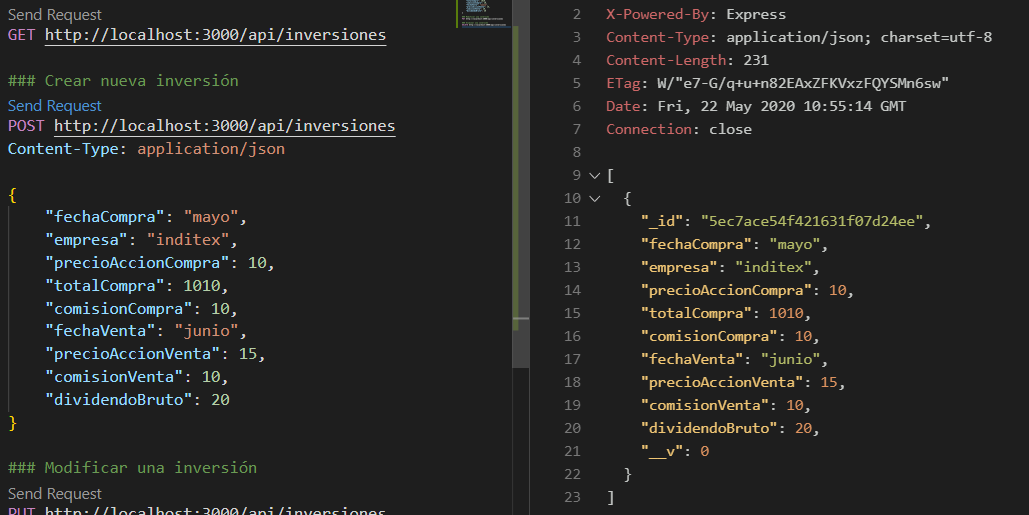
Terminamos de definir la petición **POST** con **try/catch** para recoger posibles errores



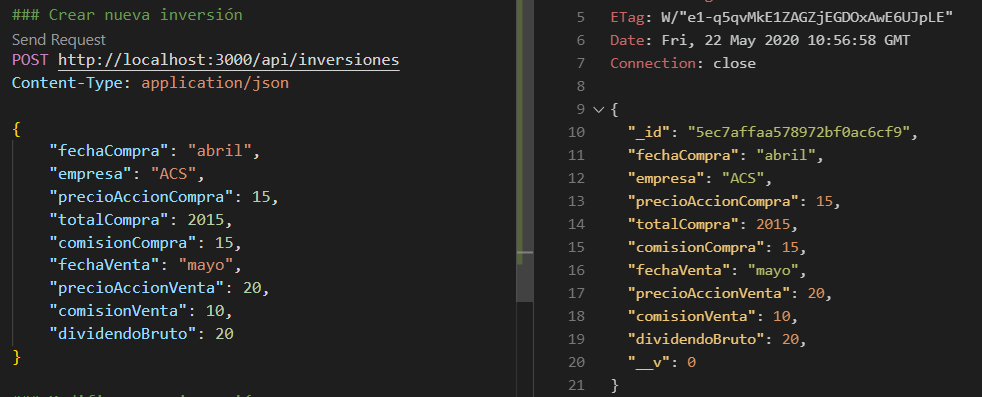
Definimos correctamente la petición **GET** para que nos devuelva todos los datos que tenemos en la bbdd



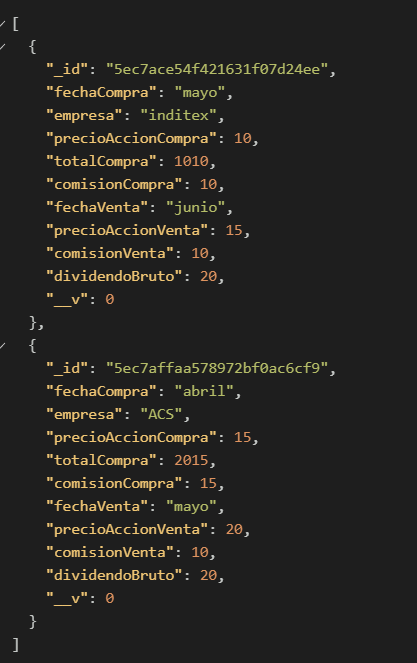
Probamos y nos aparece el único dato que tenemos guardado



Si añadimos otro más



Obtenemos el array con todos los valores

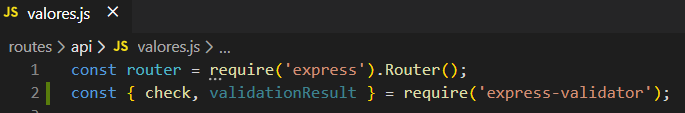


**Express Validator**

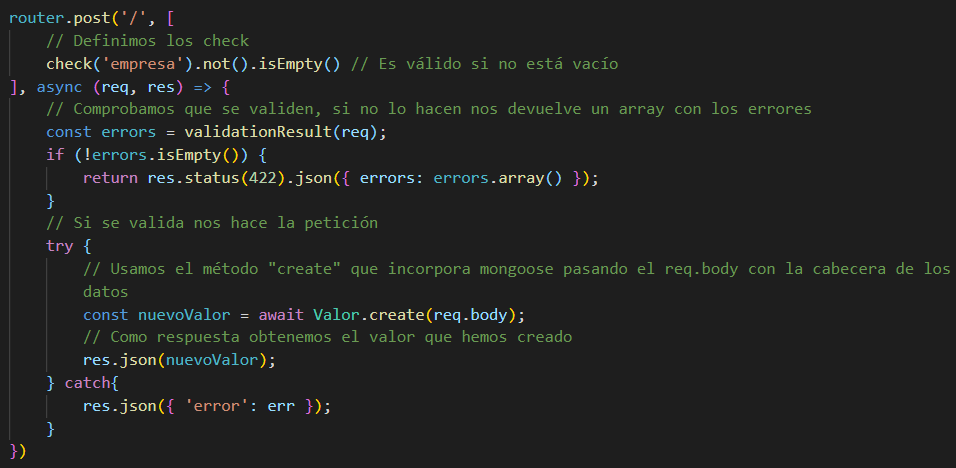
Instalamos la librería



Requerimos el uso de validaciones que vamos a usar. Con **check** hacemos las diferentes validaciones y con **ValidationResult** nos devolverá un **booleano** para confimar si se validan o no los checks

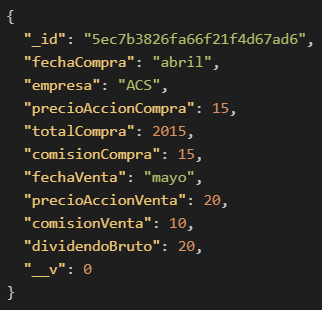


Lo definimos dentro de la petición **POST**

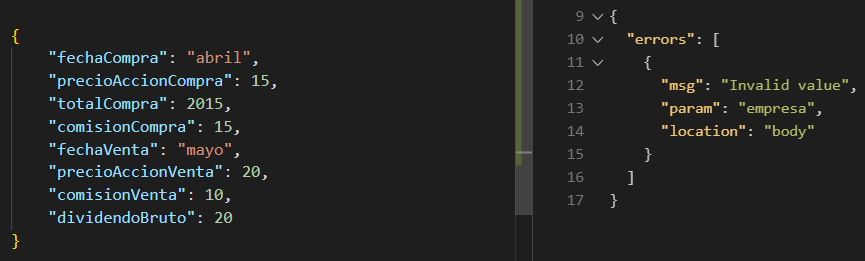


Probamos que funcione

Si el campo **empresa** no está vacío se realiza la petición

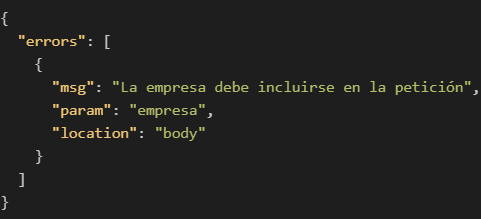


Si el campo está vacío nos devuelve el array con los errores



Podemos definir un mensaje propio para comprender el error





Una vez comprobado que funciona podemos definir el resto de checks para nuestros campos

