**Crear una Api Rest que permita realizar un crud por medio de EndPoint a una base de datos no relacional de una prendería en mongoDB**

Jose Alejandro Velez Muñoz

Centro de Gestión y Desarrollo Sostenible Surcolombiano.

Análisis y Desarrollo de Software

ID Ficha: 2692929

Pitalito – Huila

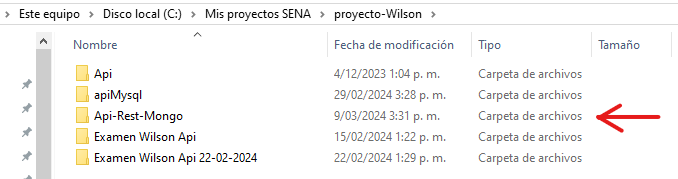
2024

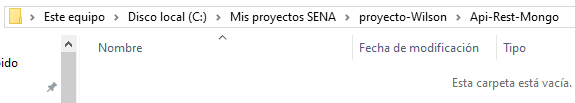
# Introducción

# Crear una Api Rest que permita realizar un crud por medio de EndPoint a una base de datos no relacional de una prendería en mongoDB

* + - 1. Primero creamos la carpeta de la API “Api-Rest-Mongo”

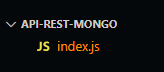
Para iniciar este proyecto de la api con mongoDB primero creamos la carpeta donde se va realizar el proyecto, que en mi caso la llame ‘Api-Rest-Mongo’.





* + - 1. Creamos el archivo principal index.js

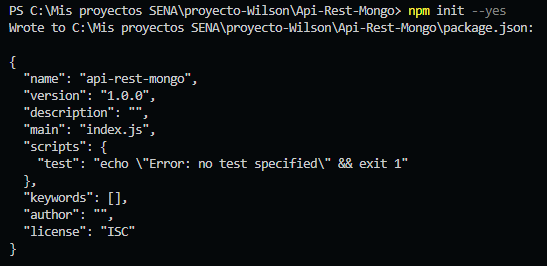
Dentro del proyecto creamos el archivo principal que de llama index.js el cual va las rutas para cada una de las consultas y requerimientos de cada colección, y también es donde generamos el puerto que se va utilizar



* + - 1. Generamos el archivo package.json.

Con el siguiente comando generamos el archivo package.json que nos permitirá guardar las librerías que se necesitan descargar.

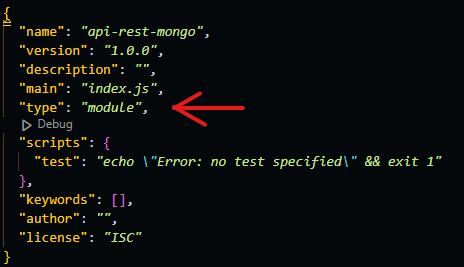




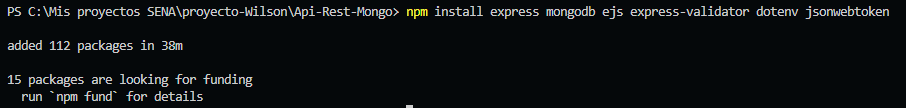
Y se creara un archivo llamado “package.json”.



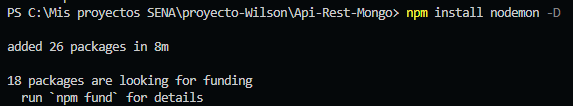
Y dentro del package.json le agregamos la siguiente línea ‘”type”: “module”,’ para utilizar la última versión de la API.



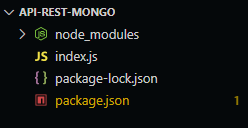
* + - 1. Luego le instalamos las librerías

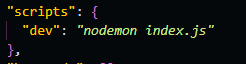






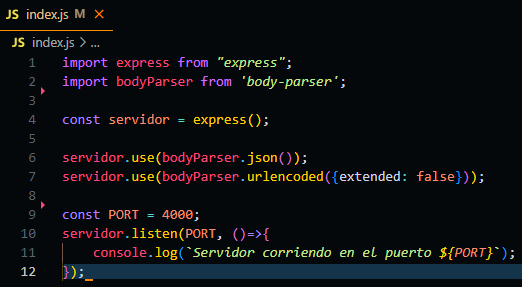




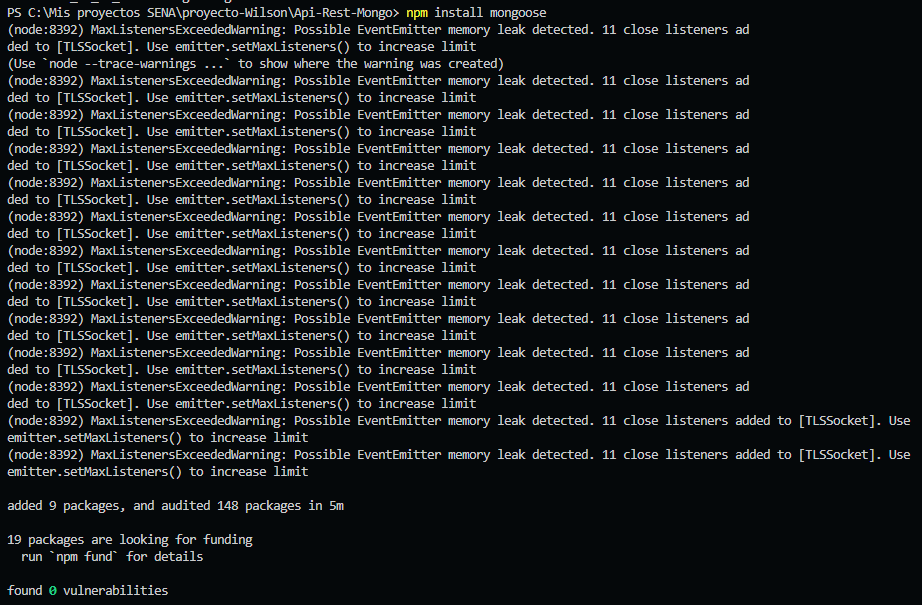


* + - 1. Luego configuramos el archivo principal index.js.

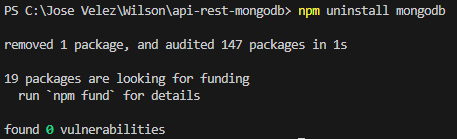
En el archivo ‘index,js’ importamos ‘express’ y ‘body-parser’, y crearemos una constante llamada servidor la cual nos permitirá agregar el puerto que necesitamos para nuestra API que vamos a desarrollar, que en este caso sería el puerto 4000.



* + - 1. Instalamos la librería mongoose.

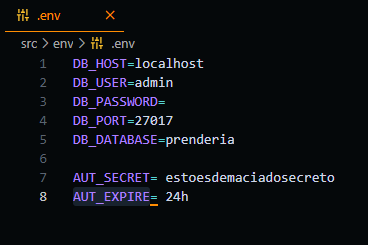


* + - 1. Desinstalamos la librería mongodb



* + - 1. Datos de MongoDB en el archivo .env

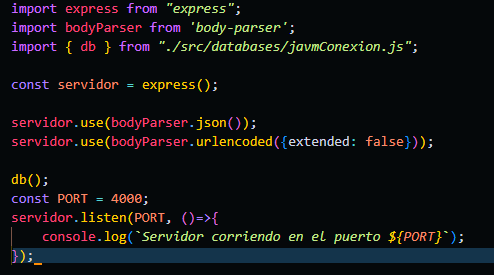
.env



* + - 1. Configuración a la base de datos.



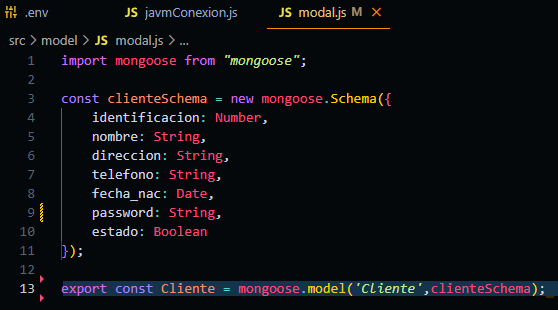
Luego de configurar la base de datos en el archivo javmConexion.js llamamos la constante db en el archivo index.js.



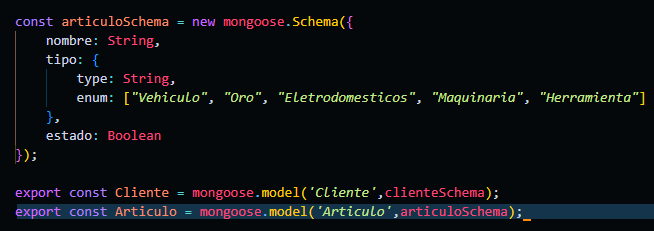
* + - 1. Creamos los modales.

Creamos los modales para las colecciones de la base de datos en Mongo DB junto con los campos necesarios que se va a necesitar en cada colección.

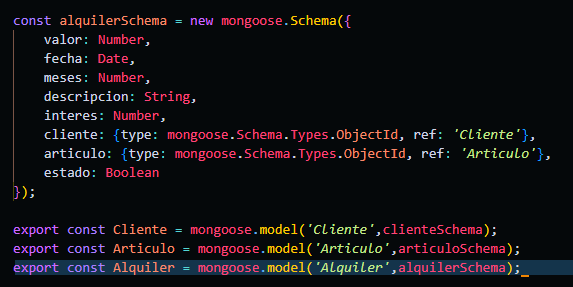
Y el primer modal que se creo fue el de clientes con los siguientes campos: identificación, nombre, dirección, teléfono, fecha\_nac (fecha nacimiento), password (contraseña), Y el estado de tipo booleano.



De segundo creamos el modal para la colección de los articulo con los siguientes campos: nombre de tipo estring, tipo de tipo string y con un enum[‘vehiculo’, ‘oro’, ‘electrodomesticos’, ‘Maquinaria’, ‘Herramienta’], y el estado de tipo booleano.



Luego creamos el modal para la colección alquiler con los siguientes campos: valor de tipo number, fecha de tipo date, mese de tipo nubmer, descripción de tipo string, interés de tipo number, cliente y articulo que van los \_id del cliente y del artículo, y por último el estado de tipo booleano.



Y por último creamos el modal para la colección del interés con los siguientes campos: mes de tipo nubmer, fecha de tipo date, valor de tipo number, alquiler que va el \_id del alquiler, y el estado de tipo booleano.



* + - 1. Elaboramos rutas y consultas con los clientes.

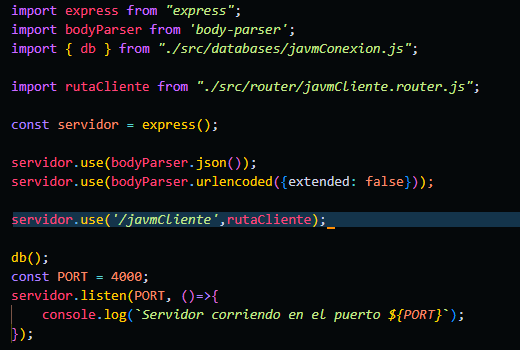
Primero creamos la consulta para listar todos los datos que estén registrados en la colección clientes, y esto lo realizamos en el archivo controlador (javmCliente.controller.js).



Luego en el archivo de rutas (javmCliente.router.js) generamos una constante llamada *‘rutaCliente’* la cual la exportamos al archivo index.js, luego creamos la ruta *‘/javmListar’* para listar el cliente con el método *‘get’*.



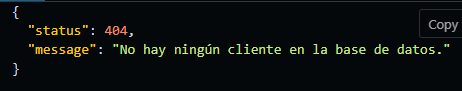
Y en el archivo principal *‘index.js’* importamos las rutas del cliente *‘rutaClientes’*, luego con el servidor creamos la ruta ‘/javmCliente’ y seguido el nombre de la rutaCliente que nos permitirá hacer las peticiones de las consultas realizadas en el controlador.



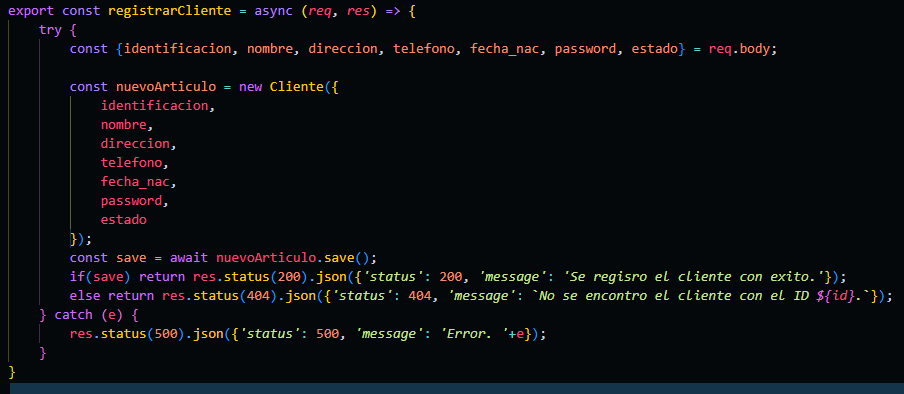
Y para listar los clientes se utilizará la siguiente ruta junto con el método get.



Y este es el mensaje que sale en caso de que no haiga datos guardados en colección de cliente, y en caso que haiga datos se mostrara una lista de clientes con sus respectivos datos.



Con este código nos permitirá registrar datos en la colección cliente utilizando los campos que se pide para este registro.



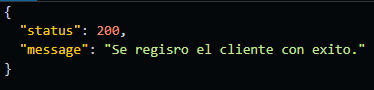
Al igual que con listar cliente aremos lo mismo, pero registrar cliente utilizando la ruta ‘/javmRegistrar’ y utilizando el método ‘post’.



Al igual que la ruta de listar cliente, en registrar cliente seria ‘localhost:4000/javmCliente/javmRegistrar’ y se utilizaría el método ‘post’, en la siguiente imagen se presentará un breve ejempro de lo que se registraría en la colección clientes.



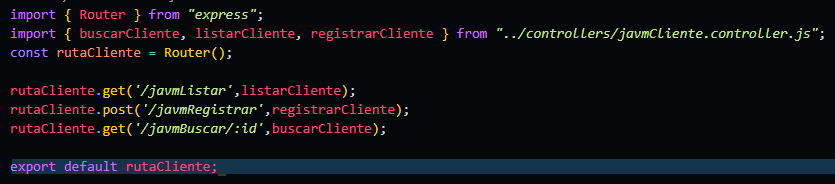
Y en la siguiente imagen se presentará el mensaje cuando el dato se guarda con éxito.



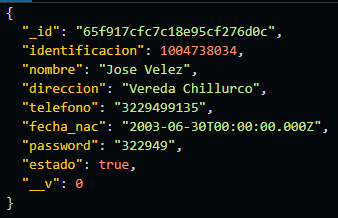
El siguiente código es una consulta que nos permite buscar clientes desde su ‘id’ que estén registrados.



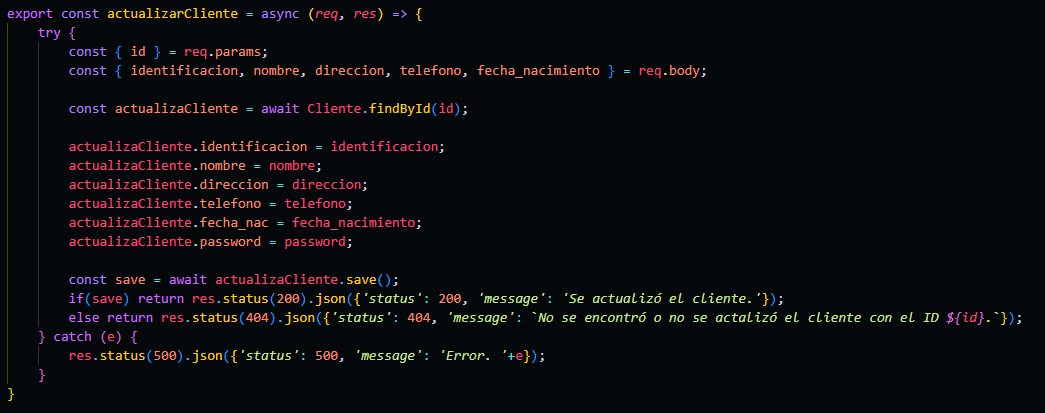
Al igual que a listar y registrar clientes, para buscar necesitamos importar la consulta y darle la ruta ‘/javmBuscar/:id’ que nos permitía remplazar el ‘:id’ por el id que obtiene el cliente, y esta ruta es de método ‘get’.



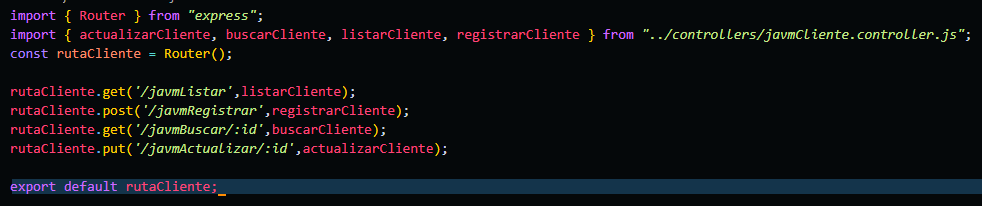
Al igual que registrar y listar, la ruta de buscar se escribe ‘localhost:4000/javmCliente/javmBuscar/65f9119e8d92fd6d6349d257’ lo cual el ‘:id’ se reemplazó por ‘65f9119e8d92fd6d6349d257’, y es de método ‘get’, y en la siguiente imagen mostraremos el resultado cuando es correcto.



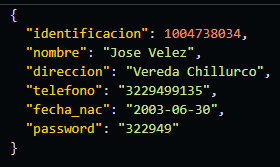
Este código nos permite actualizar los datos de los clientes buscando al cliente que se quiere actualizar desde su id.

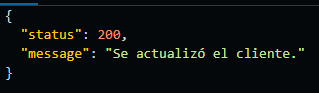


Al igual que las anteriores, debemos generar una ruta, en este caso ‘/javmActualizar/:id’, teniendo en cuenta que el ‘:id’ va a ser remplazado en el momento de escribir la ruta completa, y es de método ‘put’.

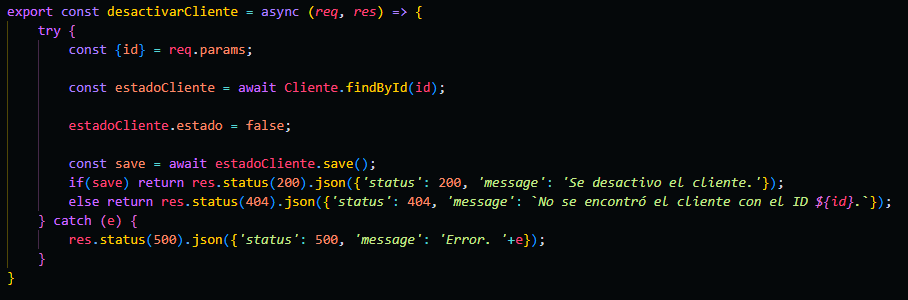


En este EndPoint la ruta seria ‘localhost:4000/javmCliente/javmActualizar/65f917cfc7c18e95cf276d0c’, lo cual el ‘:id’ fue remplazado por ‘65f917cfc7c18e95cf276d0c’, y el de método ‘put’, y los datos que se actualiza y la respuesta que se obtendría, se representa en las dos siguientes imágenes, y a diferencia a, se actualizar la mayoría de los datos menos el estado.

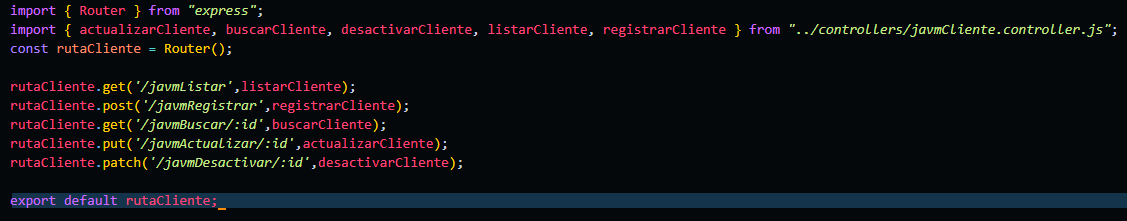




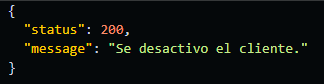
En este código nos permite desactivar a un cliente, que se busca desde el id, en este caso se actualiza solamente el estado de verdadero (true) a falso (false), para que se desactive el cliente.



Como las demás, para las rutas de la colección de clientes, para desactivar la ruta seria ‘/javmDesactivar/:id’ y seguido la importación de la consulta, sin olvidar que el método es ‘patch’.



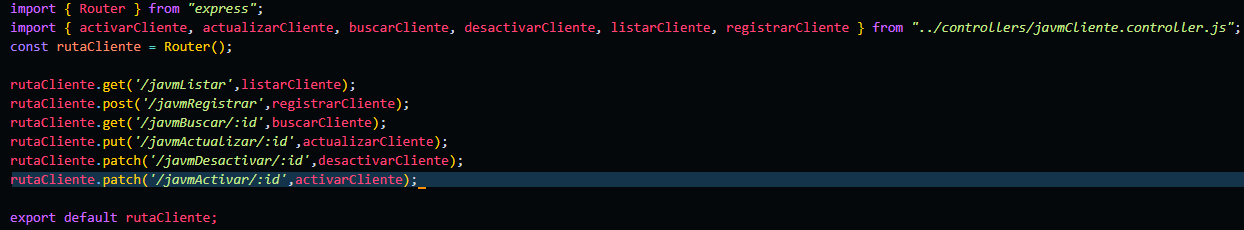
Para este EndPoint la ruta es ‘localhost:4000/javmCliente/javmDesactivar/65f917cfc7c18e95cf276d0c’ utilizando en método ‘patch’, y en la siguiente imagen se muestra un mensaje de que se desactivo el cliente, en caso de que el cliente no exista el mensaje es que el id no existe.



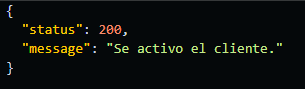
En este código nos permite activar a un cliente, que se busca por el id, en este caso se actualiza solamente el estado de falso (false) a verdadero (true), para que se active el cliente.



Y como se viene hablando, para las rutas de la colección de clientes, para activar la ruta seria ‘/javmActivar/:id’ y seguido la importación de la consulta, sin olvidar que el método es ‘patch’.



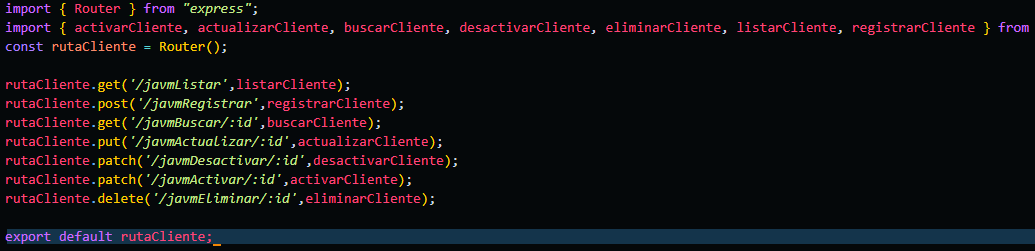
Para este EndPoint la ruta es ‘localhost:4000/javmCliente/javmActivar/65f917cfc7c18e95cf276d0c’ utilizando en método ‘patch’, y en la siguiente imagen se muestra un mensaje de que se activó el cliente, en caso de que el cliente no exista el mensaje es que el id no existe.



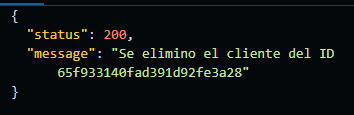
Este código nos permite eliminar a un cliente, utilizando el id del cliente y así cumplir poder eliminarlo.



Y en el archivo ‘javmCliente.router.js, con la constante rutaCliente y utilizando el método delete, y la ruta es ‘/javmEliminar/:id’ e importando la constante de eliminarCliente, podemos eliminar el cliente que se deseamos.



La peticion para eliminar cliente es ‘localhost:4000/javmCliente/javmEliminar/65f917cfc7c18e95cf276d0c’, y el método es ‘delete´, si esta petición se ejecuta correctamente se mostrar el mensaje de la siguiente imagen.



* + - 1. Elaboramos rutas y consultas con artículo.

Primero creamos la consulta para listar todos los datos que estén registrados en la colección artículo, y esto lo realizamos en el archivo controlador (javmArticulo.controller.js).



Luego en el archivo de rutas (javmArticulo.router.js) generamos una constante llamada ‘rutaArticulo’ la cual la exportamos al archivo index.js, luego creamos la ruta ‘/javmListar’ para listar el cliente con el método ‘get’.



Y en el archivo principal ‘index.js’ importamos las rutas del articulo ‘rutaArticulo’, luego con el servidor creamos la ruta ‘/javmArticulo’ y seguido el nombre de la rutaArticulo que nos permitirá hacer las peticiones de las consultas realizadas en el controlador.



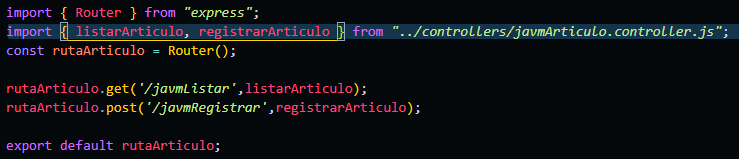


Y para listar los clientes se utilizará la siguiente ruta junto con el método get ‘localhost:4000/javmArticulo/javmListar’. En caso de que no allá datos guardados en colección de articulo sale un mensaje diciendo ‘no hay ningún artículo guardado.’, y en caso que haiga datos se mostrara una lista de artículos con sus respectivos datos.

Con este código nos permitirá registrar datos en la colección articulo utilizando los campos que se pide para este registro.



Así como se izó para listar articulo aremos lo mismo, pero ahora haremos para registrar articulo utilizando la ruta ‘/javmRegistrar’ y utilizando el método ‘post’.



Al igual que la ruta de listar artículo, en registrar artículo serio ‘localhost:4000/javmArticulo /javmRegistrar’ y se utilizaría el método ‘post’, y al registrar estos datos se registraría el nombre, tipo, y el estado (booleano) del artículo. Si se registra muestra un mensaje ‘Se registró el articulo’ si no se registra muestra un mensaje de ‘No se registró el articulo’ o de lo contrario es un error de sistema.

El siguiente código es una consulta que nos permite buscar artículos desde su ‘id’ que estén registrados en la colección de la base de datos de MongoDB.

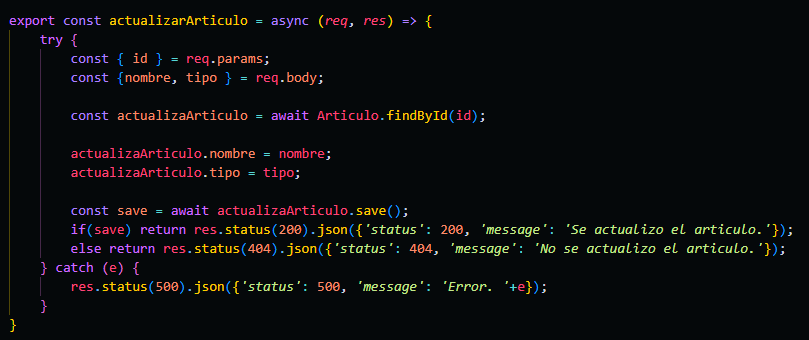


Al igual que a listar y registrar artículos, para buscar necesitamos importar la consulta y darle la ruta ‘/javmBuscar/:id’ que nos permitía remplazar el ‘:id’ por el id que obtiene el artículo, y esta ruta es de método ‘get’.

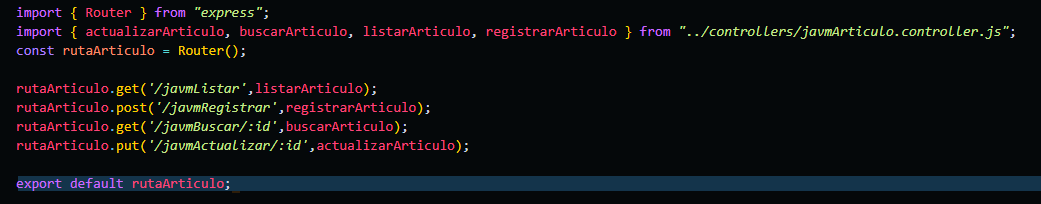


Al igual que registrar y listar, la ruta de buscar se escribe ‘localhost:4000/javmArticulo /javmBuscar/65f957ffb84c8a20931610e5 lo cual el ‘:id’ se reemplazó por ‘65f957ffb84c8a20931610e5’, y es de método ‘get’, y en la siguiente imagen mostraremos el resultado cuando es correcto.

Este código nos permite actualizar los datos de los artículos, buscando al artículo que se quiere actualizar desde su id.



Al igual que al modal cliente, debemos generar una ruta, en este caso ‘/javmActualizar/:id’, teniendo en cuenta que el ‘:id’ va a ser remplazado en el momento de escribir la ruta completa, y es de método ‘put’.

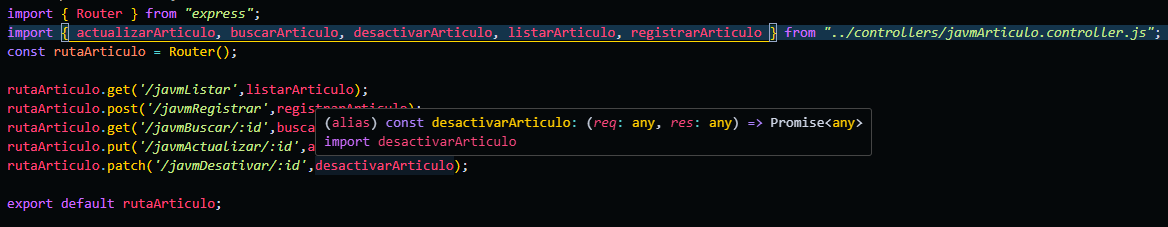


En este EndPoint la ruta seria ‘localhost:4000/javmCliente/javmActualizar/65f957ffb84c8a20931610e5’, lo cual el ‘:id’ fue remplazado por ‘65f957ffb84c8a20931610e5’, y el de método ‘put’, y los datos que se actualiza y la respuesta que se obtendría, se representa en las dos siguientes imágenes, y a diferencia a, se actualizar la mayoría de los datos menos el estado.

Al igual que al modal cliente, en este código nos permite desactivar a un artículo, que se busca desde el id, en este caso se actualiza solamente el estado de verdadero (true) a falso (false), para que se desactive el artículo.



Al igual que al modal cliente, en la colección de articulo, para desactivar la ruta seria ‘/javmDesactivar/:id’ y seguido la importación de la consulta, sin olvidar que el método que es ‘patch’.

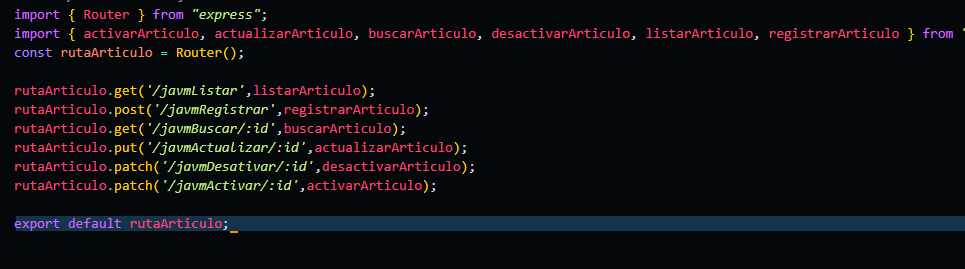


Para este EndPoint la ruta es ‘localhost:4000/javmArticulo/javmDesactivar/65f957ffb84c8a20931610e5’ utilizando en método ‘patch’, y en la siguiente imagen se muestra un mensaje de que se desactivo el cliente, en caso de que el cliente no exista el mensaje es que el id no existe.

En este código al igual que el de cliente, nos permite activar a un artículo, que se está buscando por el id, en este caso se actualiza solamente el estado de falso (false) a verdadero (true), para que se active el cliente.



Y como se viene hablando, para las rutas de la colección de clientes, para activar la ruta seria ‘/javmActivar/:id’ y seguido la importación de la consulta, sin olvidar que el método es ‘patch’.

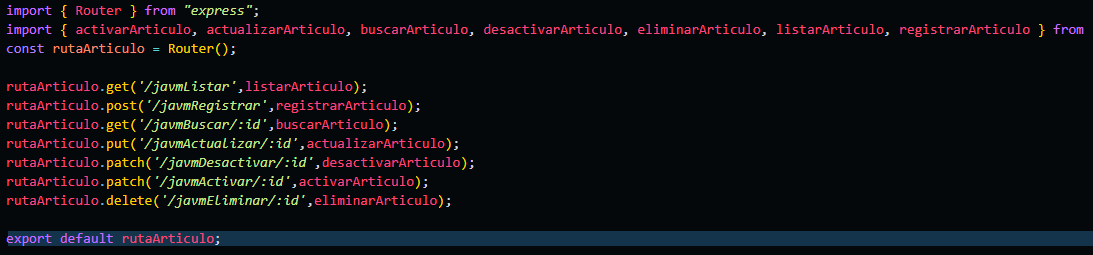


Para este EndPoint la ruta es ‘localhost:4000/javmArticulo/javmActivar/65f957ffb84c8a20931610e5’ utilizando en método ‘patch’. En caso de que se actualicé el estado se muestra un mensaje que dice ‘Se activó el articulo’, si no se actualiza se muestra un mensaje que dice ‘No se encontró en artículo con el id 65f957ffb84c8a20931610e5, de lo contrario es que tiene un error del sistema.

Este código nos permite eliminar a un artículo, utilizando el id del articulo y así poder eliminarlo.



Y en el archivo ‘javmArticulo.router.js, con la constante rutaArtliclo y utilizando el método delete, y la ruta es ‘/javmEliminar/:id’ e importando la constante de eliminarArticulo, podemos eliminar el artículo que se deseamos.



La peticion para eliminar artículo es ‘localhost:4000/javmArticulo/javmEliminar/65f957ffb84c8a20931610e5, y el método es ‘delete´, si esta petición se ejecuta correctamente se mostrar el mensaje de la siguiente imagen.

* + - 1. Elaboramos rutas y consultas con alquiler.

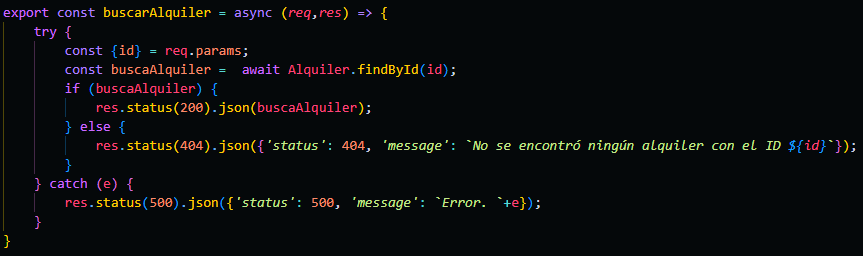
Como en las dos colecciones anteriores, la de cliente y articulo. En esta colección de alquiler realizamos una función flecha para listar todos los alquileres registrados en la base de datos, en el cual si es un status 200 deberá mostrar los alquileres que estén registrados en la base de datos, si es un status 404 debe mostrar un mensaje que especificara de que no hay ningún alquiler registrado y si es un status 500 va mostrar un mensaje de error en el sistema.



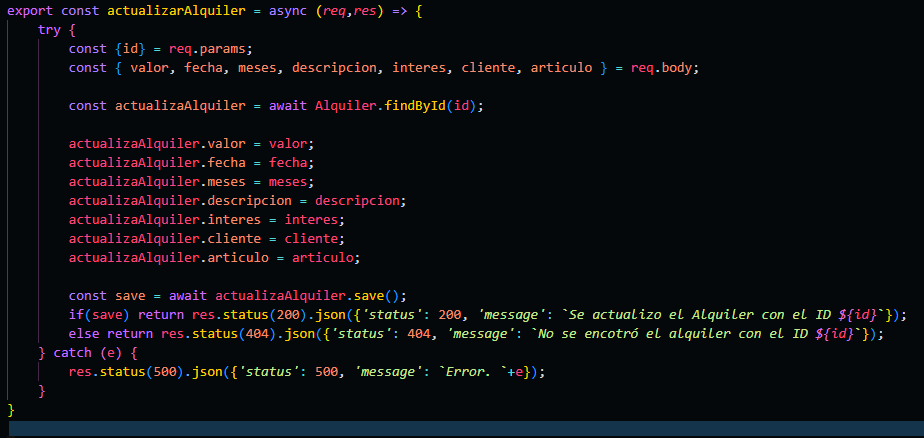
Así como se ha venido haciendo con todos los requerimientos realizados hasta ahora, se realizó el de registraAlquiler que nos permite registrar un alquiler en la base de datos de mongo, si el alquiler se registró, es un status 200 deberá mostrar un mensaje de se registró el alquiler, si es un status 404 debe mostrar un mensaje que especificando de que no se registró el alquiler y si es un status 500 va mostrar un mensaje de error en el sistema.



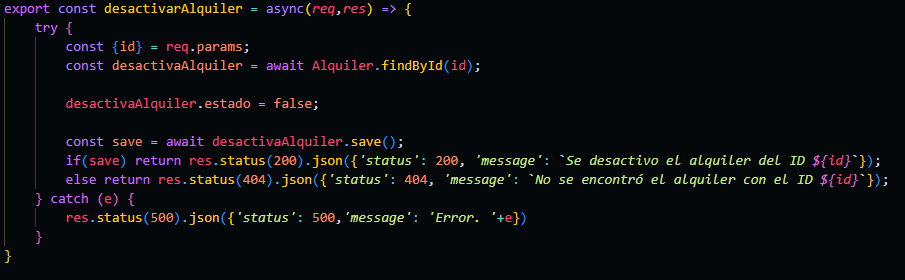
Luego creamos la constante de la función para buscar un alquiler registrado en la base de datos desde su id, si en ese id existe mostrara los datos del alquiler en un status 200, si no mostrara un mensaje en status 404 y un mensaje explicando que no hay ningún alquiler registrado por ese id y de lo contrario mostrara un status 500 y un mensaje de error en el sistema.



La siguiente función nos permite actualizar los alquileres que estén registrados en la base de datos, buscándolos desde el id, si los dato se actualizan con éxito se mostrara en un status 200 y un mensaje de que se actualizo el alquiler de ese id, si no se mostrara un status 404 y un mensaje de que no se encontró el alquiler de ese id, y de lo contrario se mostrara un status 500 y un mensaje de error en el sistema.



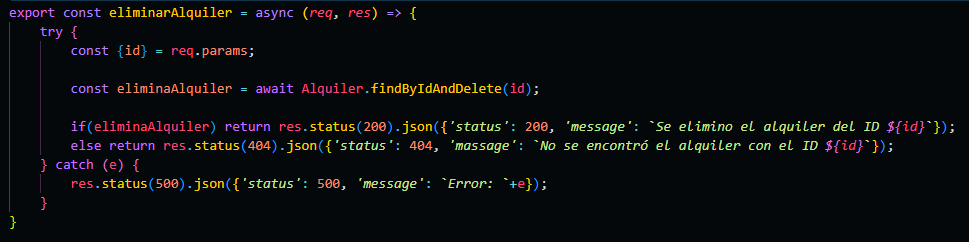
En esta función puede desactivar los alquileres, buscándolos desde el id siempre y cuando allá uno registrado, si el alquiler se desactiva se mostrará un status 200 y un mensaje de que se desactivo el alquiler de este id, si no, se mostrara en un status 404 y un mensaje de que no se encontró el alquiler de dicho id, o si no se mostrara en un status 500 y un mensaje de error en el sistema.



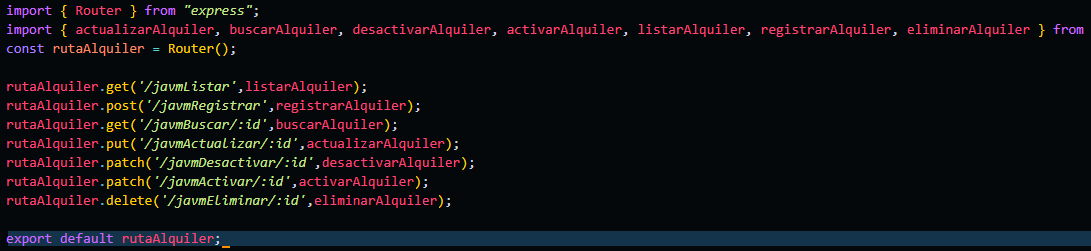
En esta función nos permite activar los alquileres, buscándolos desde el id de un alquiler registrado. si el alquiler se activa se mostrará un status 200 y un mensaje de que se activó el alquiler de ese id, si hay error, se mostrara en un status 404 y un mensaje de que no se encontró el alquiler de dicho id, o si no, se mostrará en un status 500 y un mensaje de error en el sistema.



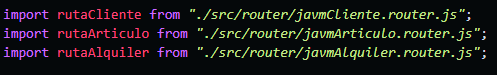
Esta función nos permite eliminar un alquiler que este registrado en la bases de datos, utilizando el id para cumplir con esta función, si el alquiler se desactiva se mostrar un status 200 y un mensaje de que se desactivo el alquiler de este id, si no se mostrara en un status 404 y un mensaje de que no se encontró el alquiler de dicho id, o si no se mostrara en un status 500 y un mensaje de error en el sistema.

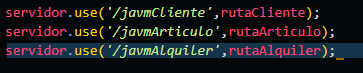


En el archivo de las rutas ‘javmAlquiler.router.js’ importamos Router desde la librería de express, y creamos una constante que se puso el nombre de rutaAlquiler, también se importó cada requerimiento que se ralizo en el archivo del controlador, teniendo en cuenta que para listar los alquileres y buscar un alquiler por su id se requiere el método GET, para registrar un alquiler se utiliza el método POST, para actualizar un alquiler se utiliza el método PUT, para activar y desactivar se requiere el método PATCH, y para eliminar se requiere el método delete, sin olvidar que para actualizar, desactivar, activar y eliminar se realiza por medio del id de dicho alquiler que este registrado.



Para el index.js





* + - 1. Elaboramos rutas y consultas con interés.

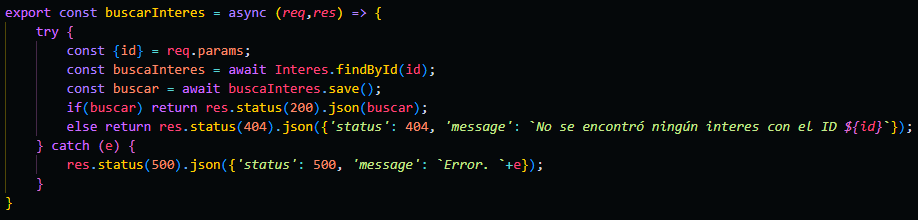
Así Como en las colecciones anteriores. En esta colección de interés realizamos una función para listar todos los intereses registrados en la base de datos.



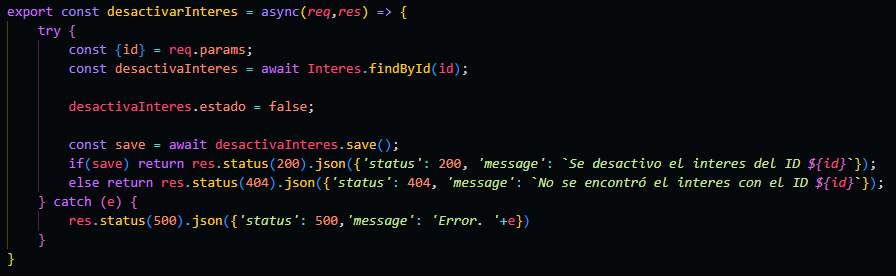
Así como se ha venido haciendo con todos los requerimientos, se realizó el de registraInteres que nos permite registrar un interés en la base de datos.

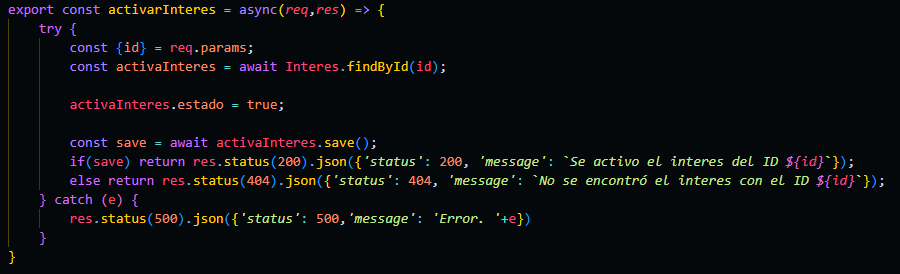


Luego creamos la constante de la función para buscar un interés desde su id.





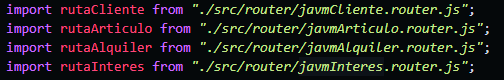


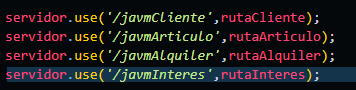






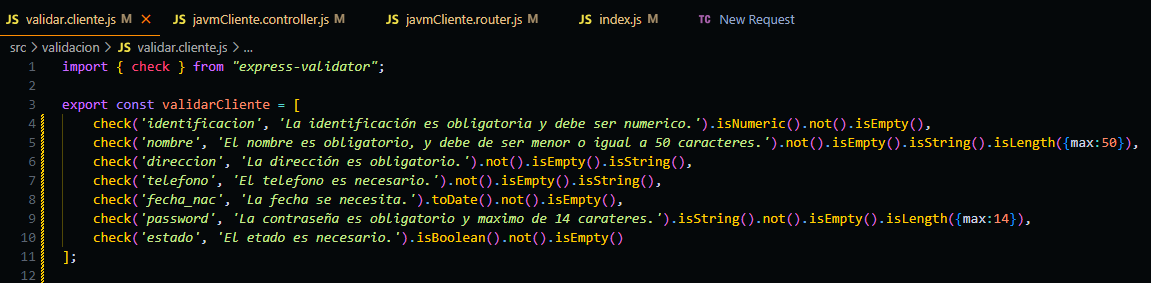
Para el index.js



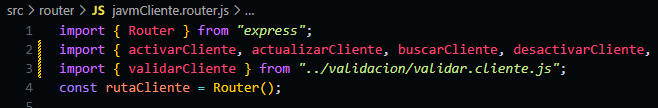


* + - 1. Hacer las validaciones del cliente.

Hacemos uso de la librería de express-validator importándola desde check en el archivo validar.cliente.js, y a cada campo para el registro lo vamos a validar para que no envié errores o campos vacíos a la base de datos.



En el archivo de rutas (javmCliente.router.js) importamos la validación para el registro.



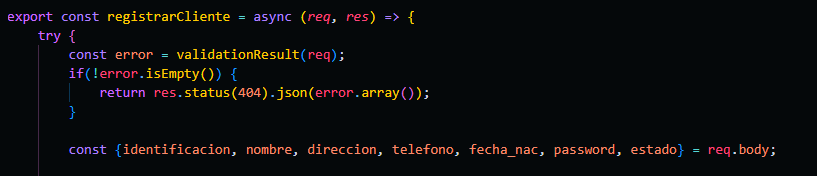
Y se lo añadimos a la ruta del registro.



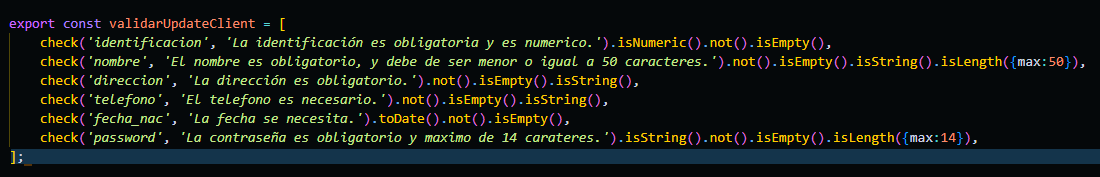
Y en el controlador importamos ‘exprex-validator’ y asemos uso de validationResult.



Y de igual manera, al registro agregamos las líneas de error que se visualizan en el código para que no acepten los errores que le llegan al archivo router.



Luego asemos las validaciones para los campos de actualizar.



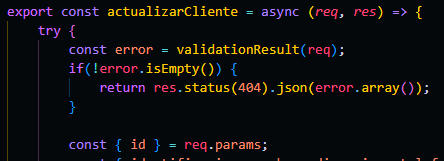
En el archivo de las rutas de clientes importamos la validación para la ruta de actualizar.



Y se lo añadimos a la ruta de actualizar



Al igual que al registro, al actualizar clientes agregamos las líneas de error que se visualizan en el código para que no acepten los errores que le llegan al archivo router, utilizando validationResult de express-validator.

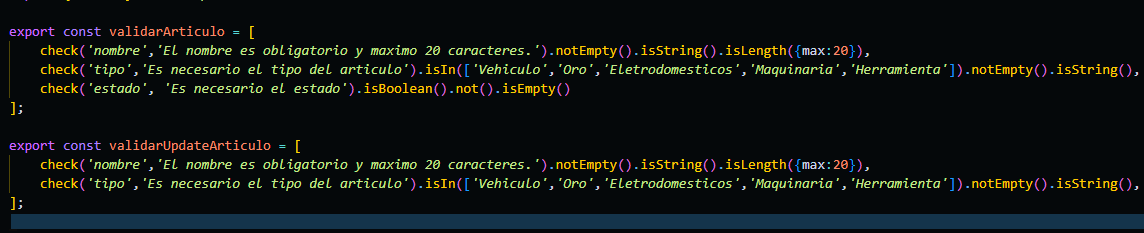


* + - 1. Hacer las validaciones del artículo.

Primero importamos ‘check’ de la librería ‘express-validator’, y la importamos en el archivo ‘validar.articulo.js’.



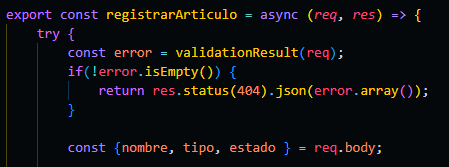
Luego validamos cada uno de los campos de registro y de actualización de los artículos.



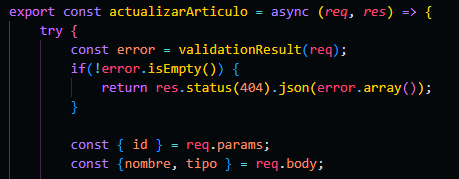
Después importamos ‘validationResult’ desde ‘express-validator’ en el archivo ‘javnArticulo.controller.js’ para validar las constantes de registrar y actualizar interés.



Y para la constate del registro para los artículos le agregamos los códigos de error que se visualizan dentro del ‘try’, y este código es para que no acepte los errores que llegan al router.



Al igual que a la constante de registrar hacemos lo mismo en la constante actualizar artículos, y agregamos los códigos de error y que no acepte los errores que llegan al router.



En el archivo de las rutas de los artículos ‘javmArticulo.router.js’ importamos las validaciones que se utilizara en registrar y actualizar un artículo.



Y para la ruta de registrar y actualizar los artículos le agregamos las validaciones correspondientes a cada una.

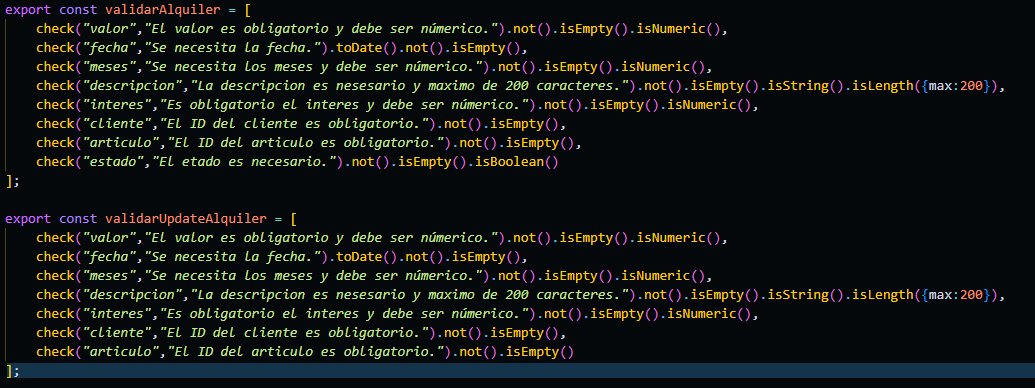


* + - 1. Hacer las validaciones del alquiler.

Al igual que las dos colecciones anteriores, primero importamos ‘check’ de la librería ‘express-validator’, y la importamos en el archivo ‘validar.alquiler.js’.



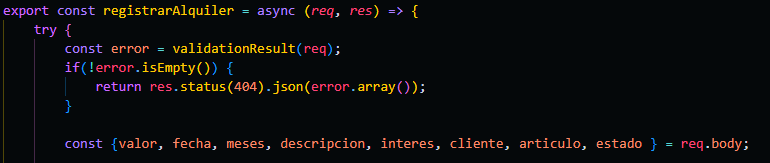
Luego validamos cada uno de los campos de registro y de actualización que se requiere en alquiler.



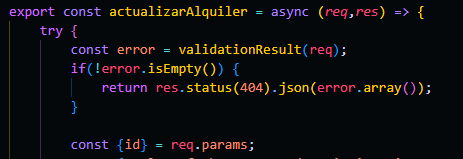
Después importamos ‘validationResult’ desde ‘express-validator’ en el archivo ‘javnAlquiler.controller.js’ para validar las constantes de registrar y actualizar interés, asi como en las dos colecciones anteriores.



Y para la constate del registro para los alquileres le agregamos los códigos de error que se visualizan dentro del ‘try’, y este código es para que no acepte los errores que llegan al router del alquiler.



Al igual que a la constante de registrar hacemos lo mismo en la constante actualizar alquiler, agregamos los códigos de error y para que no acepte los errores que llegan al router, así como en las colecciones anteriores.



En el archivo de las rutas de los artículos ‘javmArticulo.router.js’ importamos las validaciones que se utilizara en registrar y actualizar un artículo.



Y para la ruta de registrar y actualizar los artículos le agregamos las validaciones correspondientes a cada una.

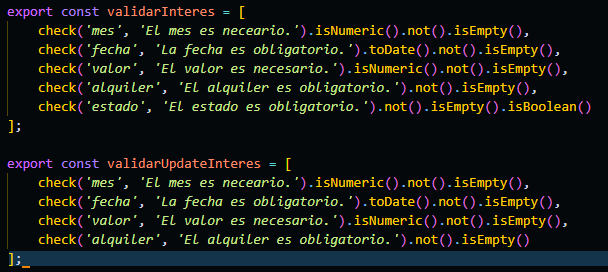


* + - 1. Hacer las validaciones del interés.

Al igual que las colecciones anteriores, importamos ‘check’ desde ‘express-validator’, y la importamos en el archivo ‘validar.interes.js’.



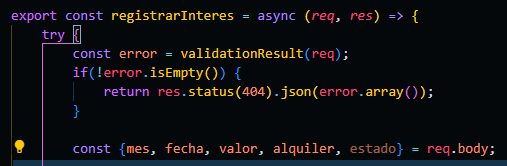
Luego validamos cada uno de los campos de registrar y de actualizar el interés.



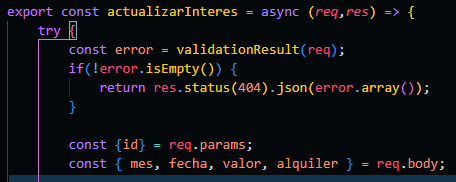
Después importamos ‘validationResult’ desde ‘express-validator’ en el archivo ‘javnInteres.controller.js’ para validar las constantes de registrar y actualizar interés.



Al igual que las colecciones anteriores, para la constate del registro de los intereses le agregamos los códigos de error que se visualizan dentro del ‘try’, y este código es el que permite que no acepte los errores que llegan al router del interes.



Al igual que a la constante de registrar, hacemos lo mismo en la constante actualizar interés, agregamos los códigos de error y para que no acepte los errores que llegan al router, así como en las colecciones anteriores.



En el archivo de las rutas de los artículos ‘javmInteres.router.js’ importamos las validaciones que se utilizara en registrar y actualizar un interes.

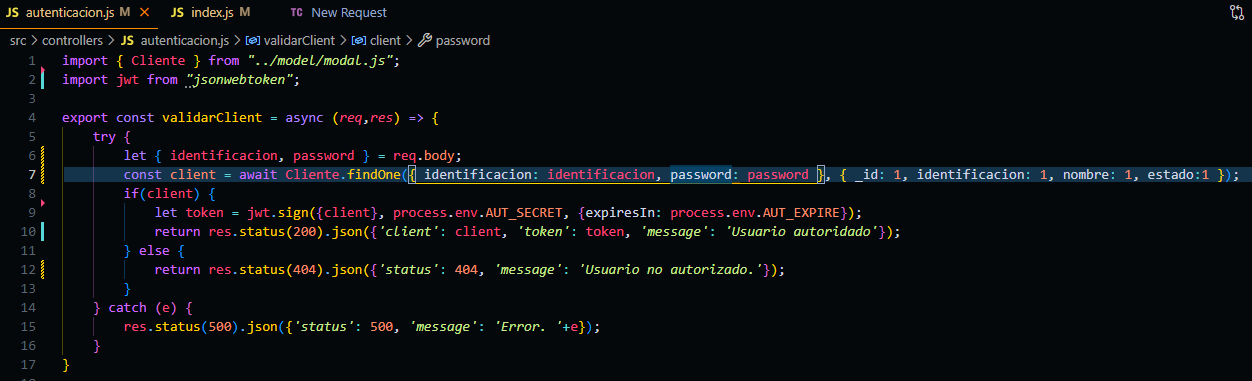


Y para la ruta de registrar y actualizar los artículos le agregamos las validaciones correspondientes a cada una.

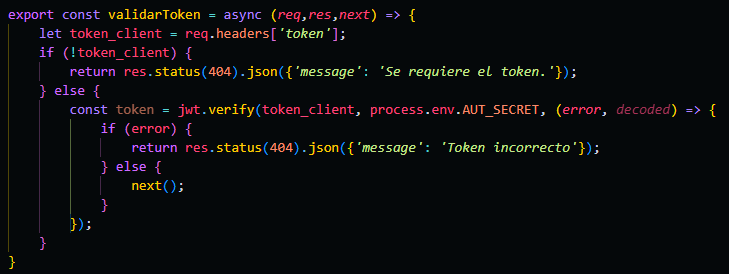


* + - 1. Generamos el token.

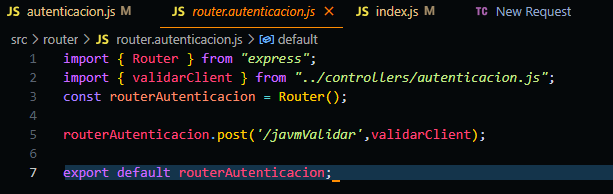
Para este caso de validar token necesitamos el modal de clientes y la librería de jsonwebtoken, para genera el token vamos necesita la identificación y un password de un cliente.



Nos permite validar el token una vez que lo hayamos guardado Headers.



En el archivo router.autentcacion.js importamos ‘validarClien’ y creamos la ruta para generar el token.

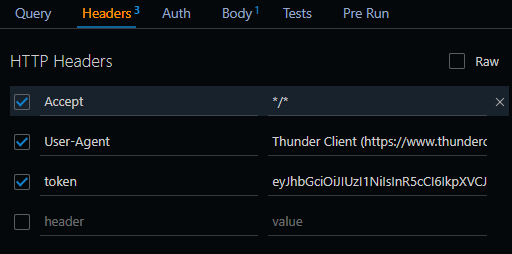


Luego en el archivo principal index.js importamos la ruta de la autenticación del token (routerAutenticacion).

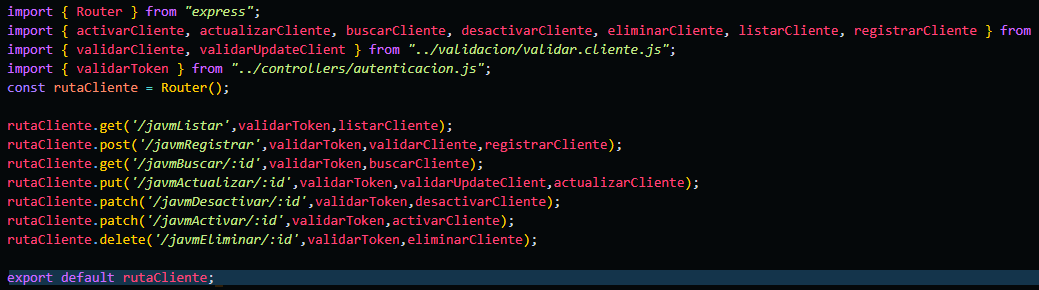




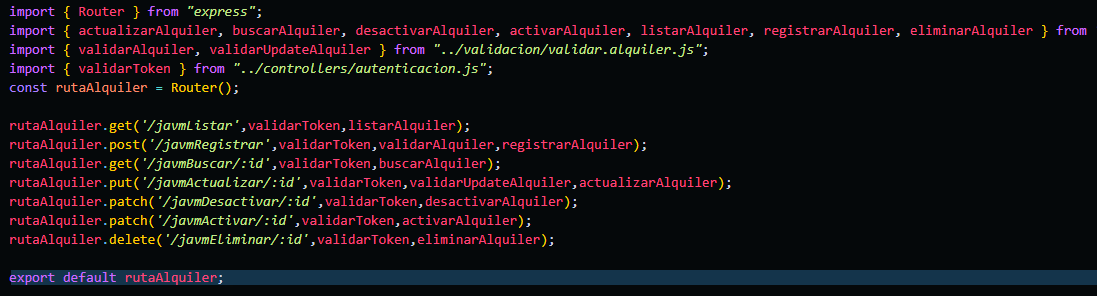
Ya guardando el token en Headers podemos utilizar la constante validarToken.

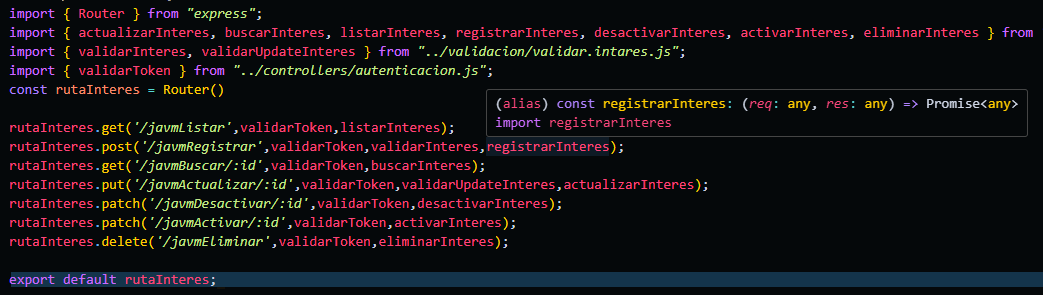


* + - 1. Validar los EndPoint con el Token.







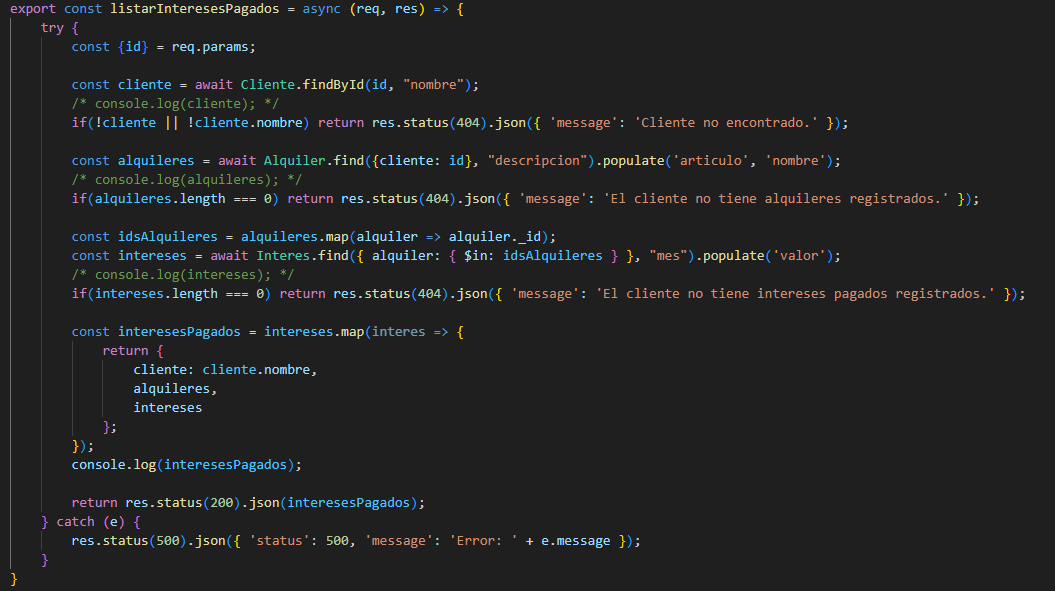


* + - 1. Las consultas.

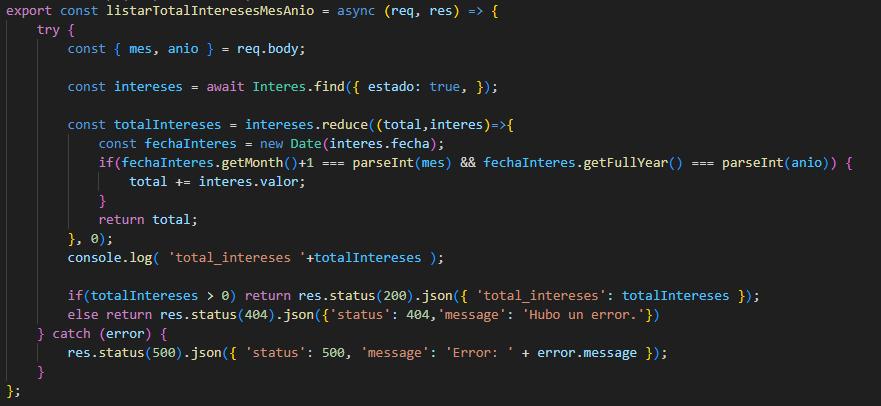
Al crear las consultas importamos los modales que necesitamos para dichas consultas.



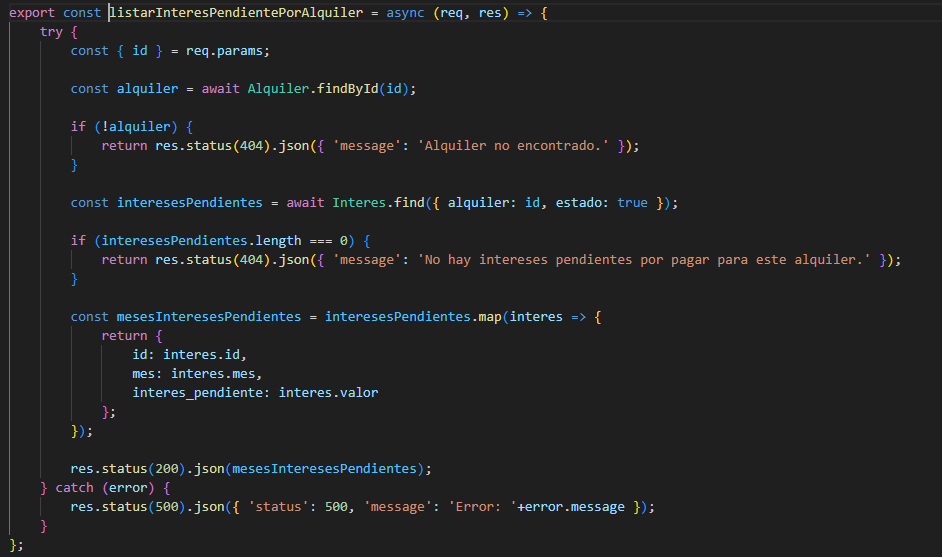
En esta consulta nos permite listar los intereses pagados por el ‘id’ de un cliente que este registrado en la base datos.



Esta consulta nos permite listar el total de intereses por el mes y el año de cada interés registrado.



En esta consulta nos permite listar los intereses pendientes por el ‘id’ de un alquiler.



* El código del articulo y cuanto interés ha pagado

Esta consulta nos permite listar los intereses pagados por el id de un artículo.



Y aquí se muestran las rutas con el método para la petición de las consultas.

Y así en como la utilizamos en el index,js



* + - 1. La documentación.

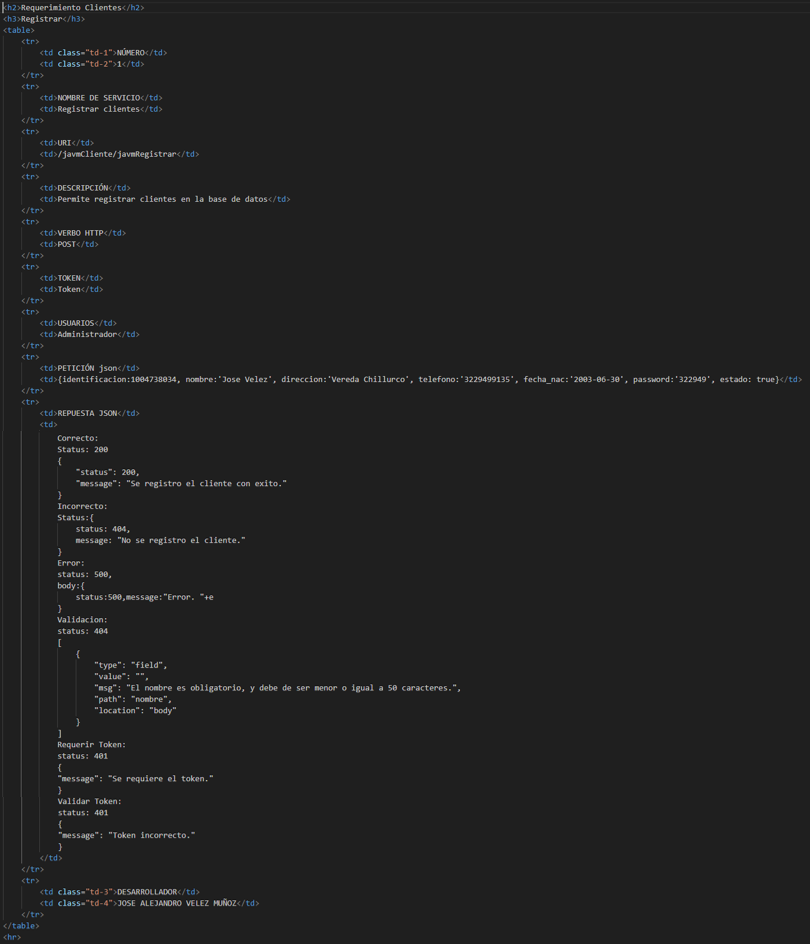
Dentro del <head></head> agregamos el archivo para darle un poco de estilos, y le agregamos un titulo a la pagina de la documentación.



En la primera línea dentro del <body></body> agregamos un título para la documentación del api con mongo.



En la siguiente imagen presentamos como se crea la primera tabla de registrar los clientes en la documentación y así como esta tabla también se crea las tablas para listar, buscar, actualizar, desactivar, activar y eliminar, y así mismo como se crea la documentación de la colección de clientes, también se crea la documentación de las colecciones de articulo, alquiler e intereses.



# Conclusión