**FULL STACK DEVELOPER**

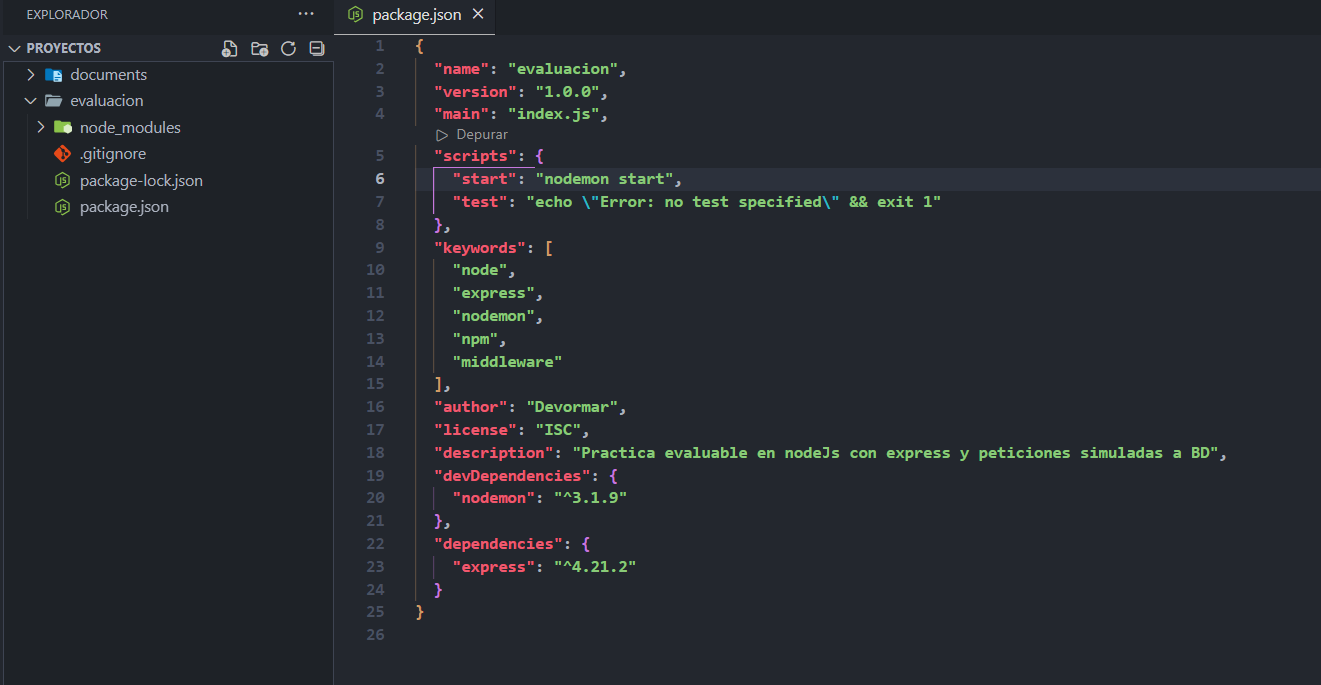
**Módulo 5 – NodeJS**

**Jorge Borrego Marqués**

**Prueba Evaluable**

1. **Crea un directorio vacío denominado evaluación y ábrelo con Visual Studio Code.**

* Folder del Proyecto:PROYECTOS/evaluacion

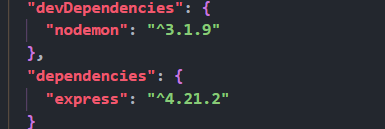


1. **Inicializa un proyecto NodeJS e instala las librerías Nodemon y Express.**

* Inicializo un Repositorio nuevo de git en la carpeta ./Proyectos:

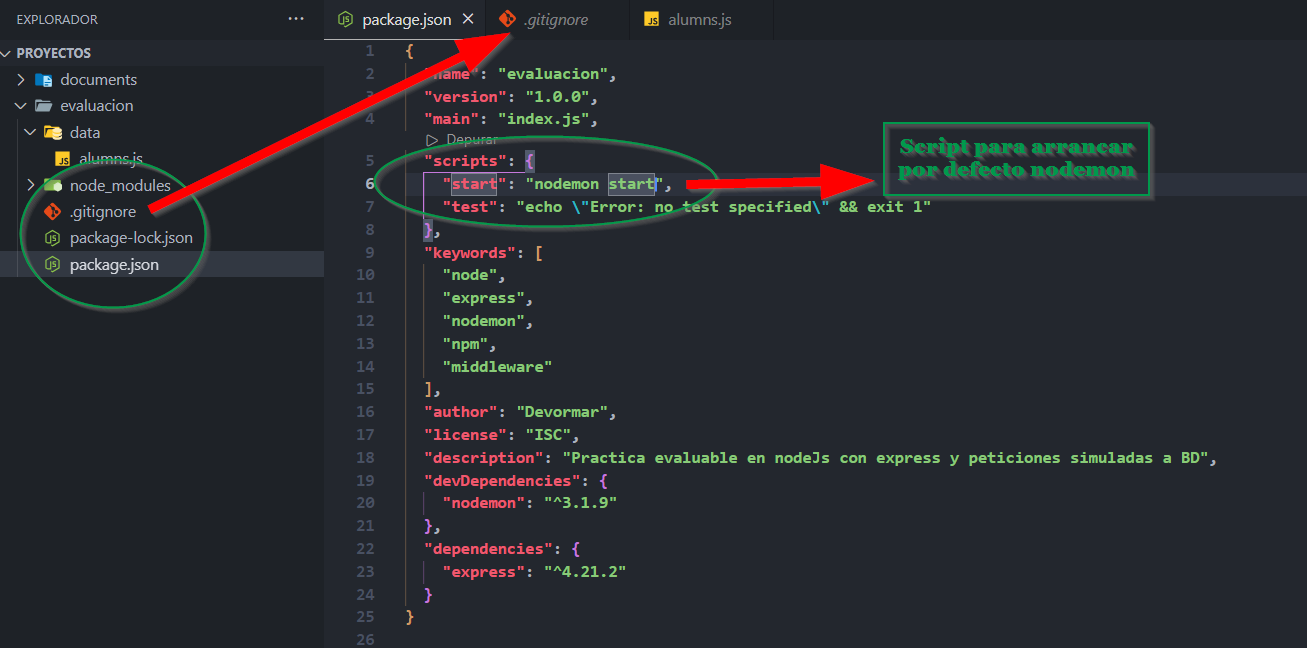
*git init*

* Añado una carpeta nueva en la carpeta \Proyectos\evaluacion, donde inicializamos un proyecto nuevo *npm init –y.*
* Instalación de la devDependencia: *npm install nodemon –save-dev*
* Instalación de la Dependencia: *npm install express –save*

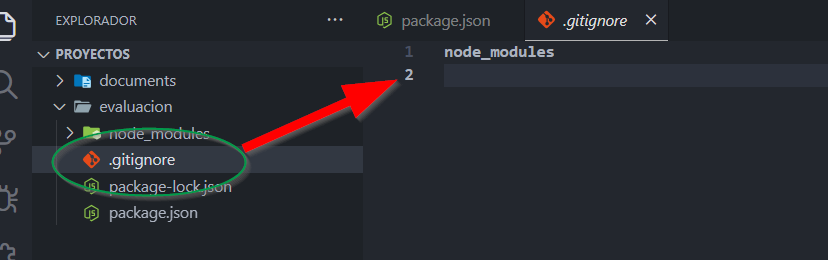


1. **Genera un *script* para utilizar nodemon con el archivo principal app.js y un archivo .gitignore.**

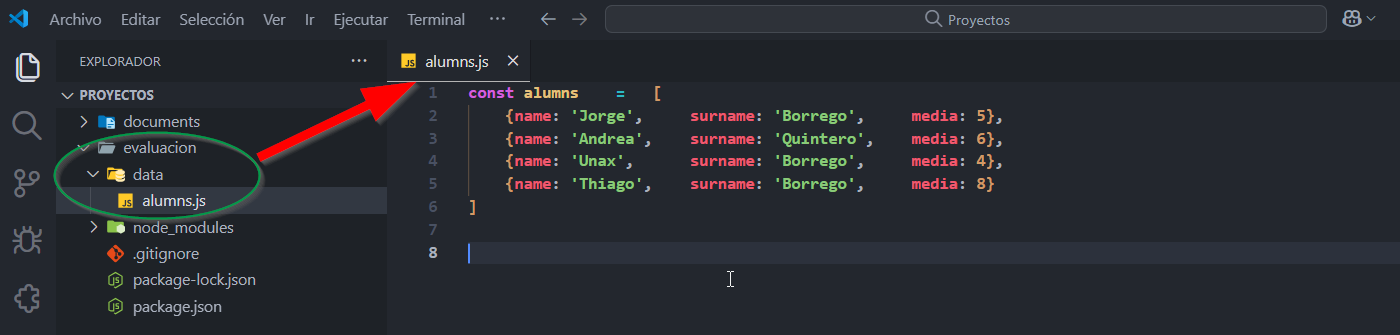
* Script para arrancar por defecto nodemon:



* Añadimos archivo .gitignore y en el cuerpo node\_modules:

****

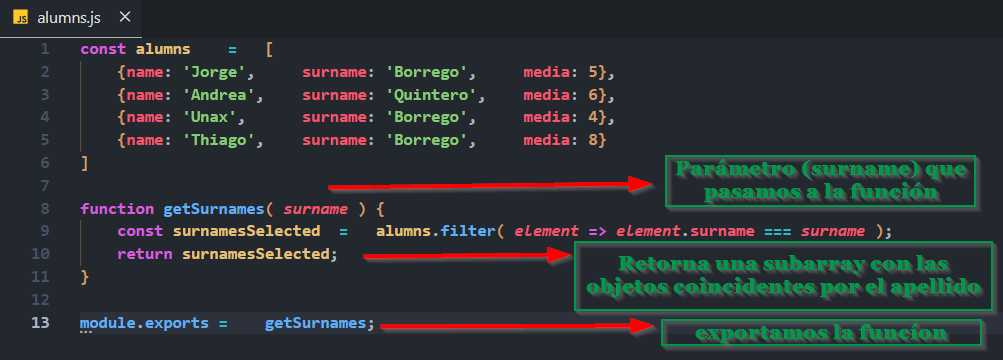
1. **Crea un directorio denominado data en la raíz del proyecto con un archivo alumns.js que contenga un *array* de datos de alumnos**.



Creamos en la raíz del proyecto un directorio nuevo nombrado como “data”. Dentro del directorio creamos un archivo js llamado alumns.js que contiene una Array con una serie de alumnos en forma de objetos con los atributos de: name: sting, surname: string y media: number.

1. **En el archivo anterior, crea y exporta una función que, a partir de un valor de apellido, devuelva las coincidencias.**

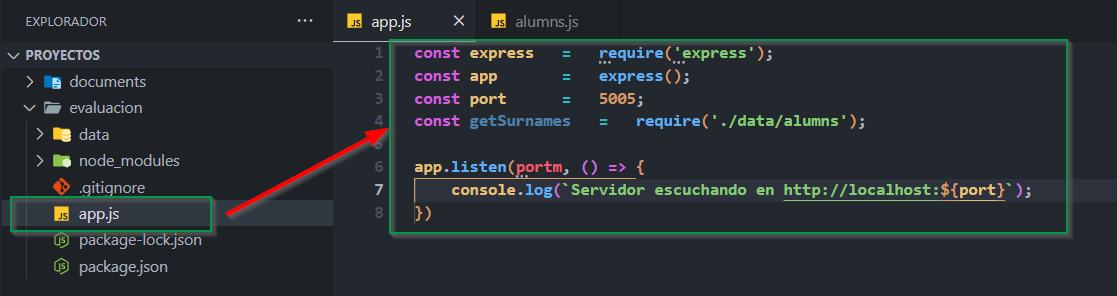
En archivo .*/data/alumns.js* creamos una función que recibe como parámetro un string, y que retorna un subArray con todas las coincidencias de alumnos cuyo apellido coincide con el parámetro pasado en la función.



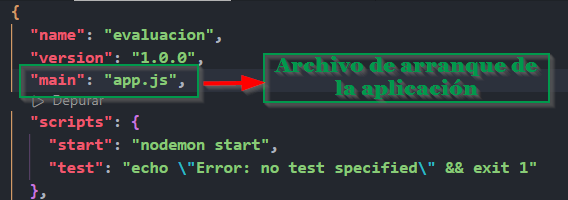
En la función usamos el método *filter()* de los arrays, por el cual generamos un nuevo *subArray* con las coincidencias de los objetos cuyo *surname* coincide con el parámetro de llamada a la función.

Exportamos la función por medio de *module.exports* para poder trabajar desde el archivo *app.js* con los valores , y que recogeremos por medio de un *require().*

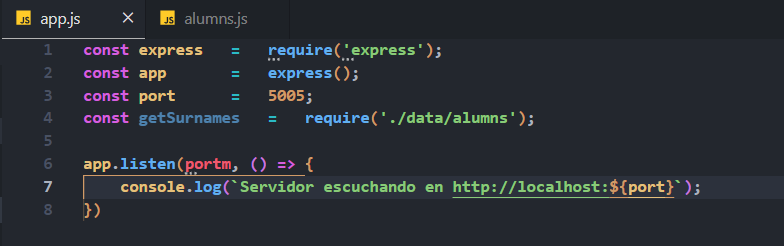
1. **En la raíz del proyecto, crea el archivo principal app.js con la sintaxis de servidor de Express.**



En la raíz del proyecto creamos el archivo *app.js*, donde anteriormente hemos indicado en el archivo package.json que iba a ser el archivo de arranque del proyecto:

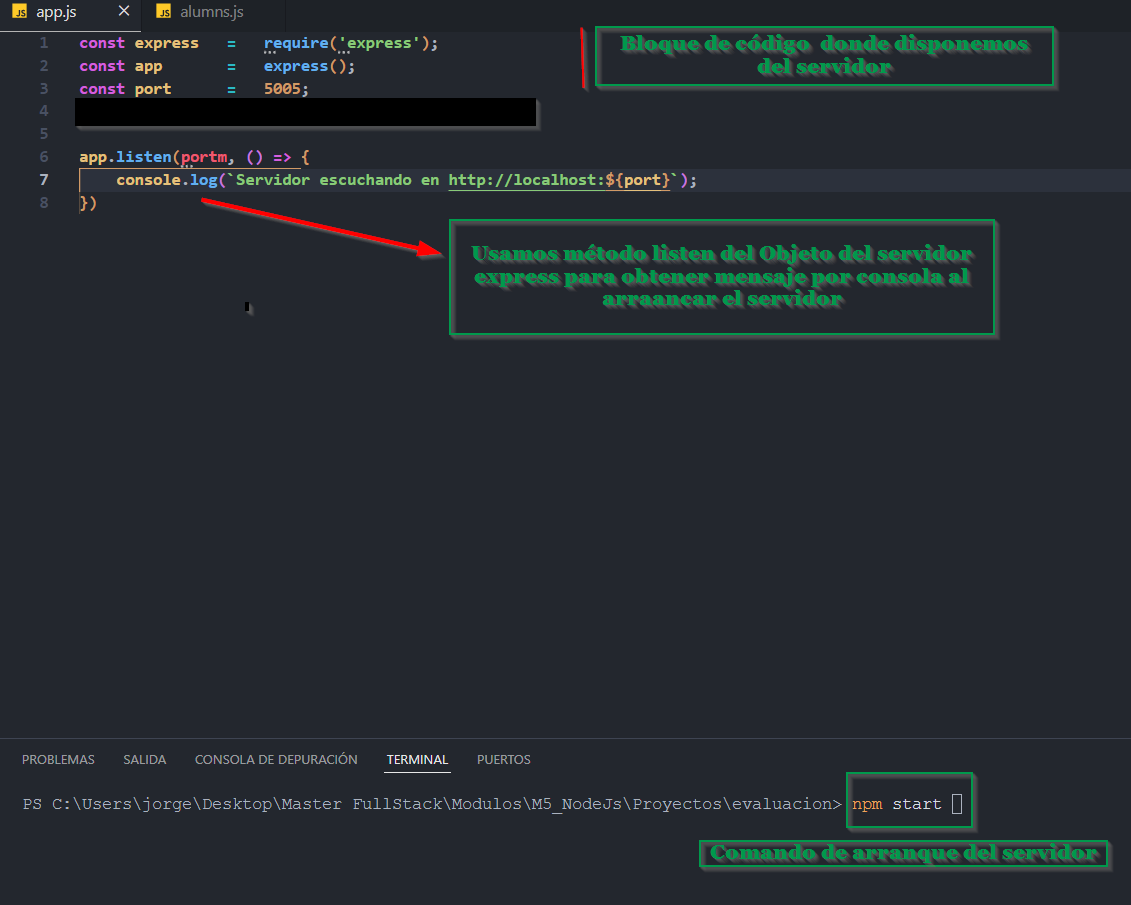


En este archivo codificamos la lógica necesaria para trabajar con el servidor express.

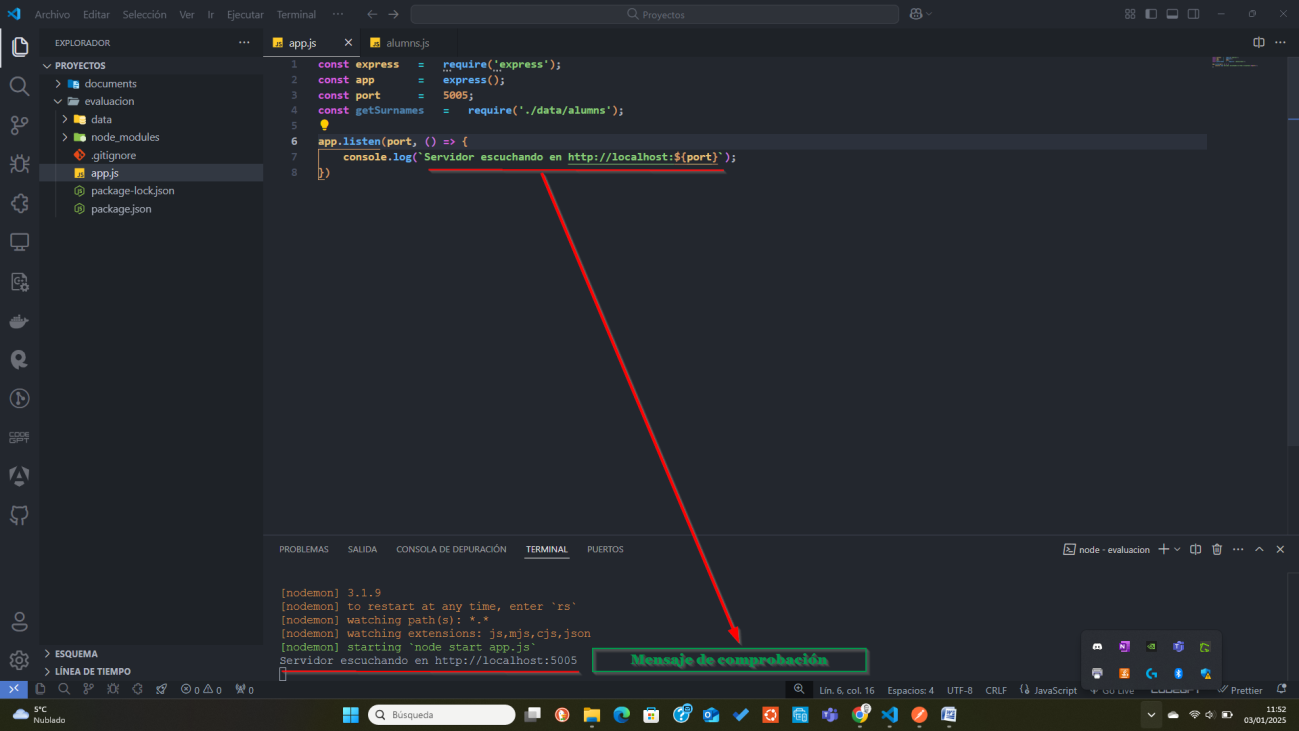


* Requerimos el módulo
* Instanciamos un objeto del método en una constante
* Definimos el puerto
* Usamos un método de escucha del servidor para comprobar que al arrancar el Servidor todo funciona Ok.

Desde una nueva terminal arrancamos el proyecto con el comando *npm start*, que inicia *nodemon start* y por consola recibimos el mensaje configurado en el console.log del método listen del Objeto de *express.*

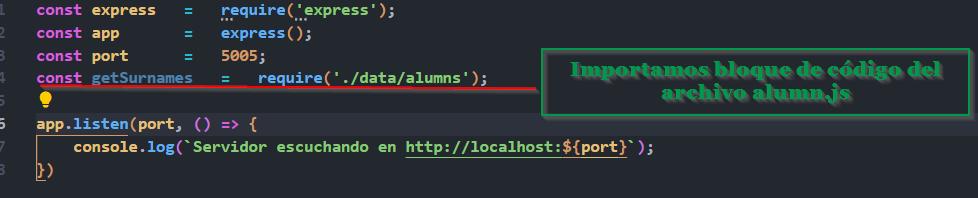


Una vez arrancado el servidor comprobamos la respuesta por consola:

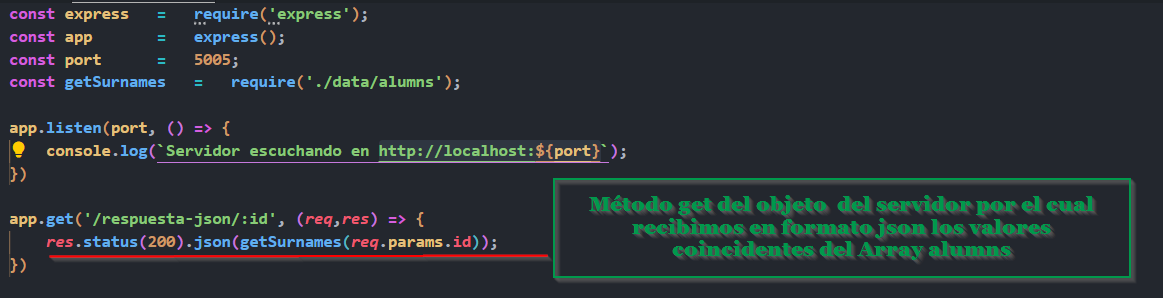


1. **Importa en app.js la función desarrollada en el archivo alumns.js.**

Importamos la función anteriormente definida en el archivo alumns.js del directorio data, por medio de un require(path del archivo):

****

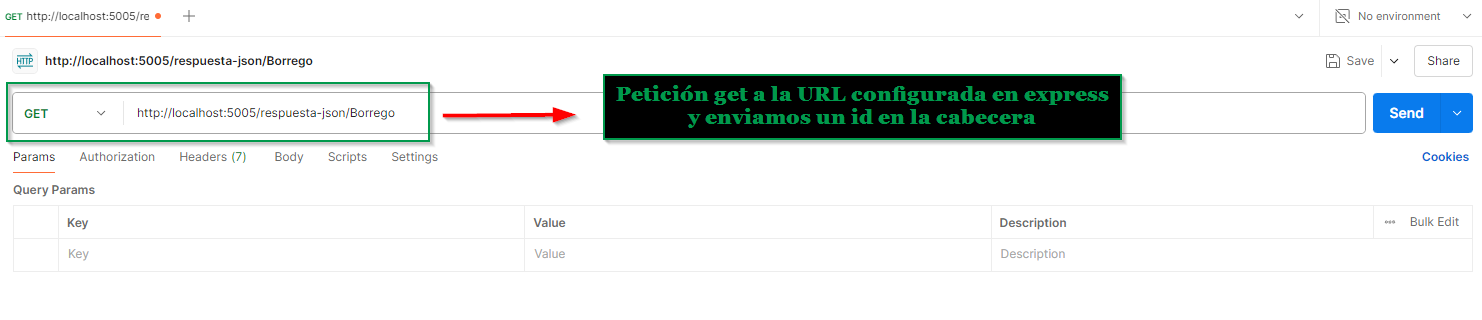
1. **. Crea un método "get" en app.js que reciba un parámetro de apellidos y ejecute la función importada para devolver los resultados.**



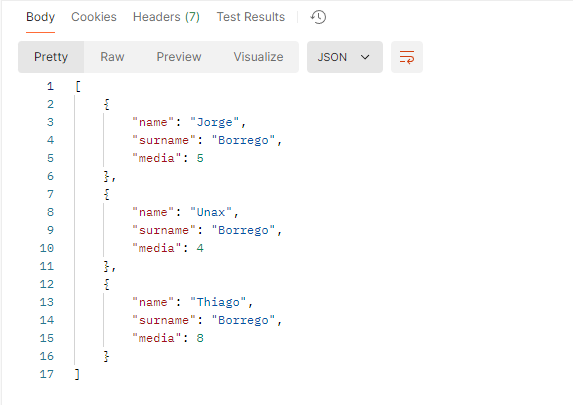
Por medio del método *get* del Objeto instanciado, recibimos una respuesta en formato *json* (método json()), de la función importada del fichero *alumns.js*, y como parámetro por cabecera el valor que le indiquemos. En este caso el parámetro es “*Borrego*”:

1. **Comprueba una petición "get" con Postman.**

Ahora para comprobar la respuesta de la función importada pasamos por la cabecera y en la ruta especificada, enviamos una petición *get* por medio de *Postman*:

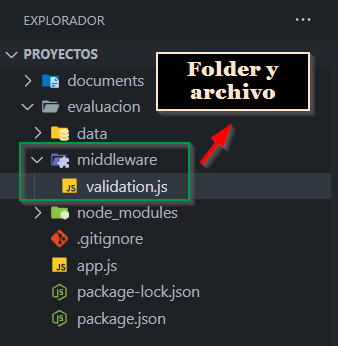


* **Respuesta:**

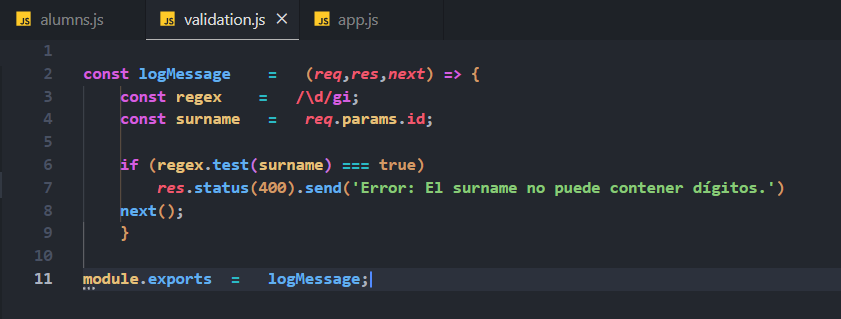
****

1. **Crea un directorio denominado middleware en la raíz del proyecto y un archivo validation.js en el que se utilice una función de middleware con Express que compruebe si el parámetro de apellidos contiene números y devuelva un mensaje de error si es así.**

* Directorio /middleware archivo validation.js:



Una vez creado el archivo validation.js mostramos el bloque necesario para cumplir con la petición del ejercicio.



En primer lugar definimos una función en una constante que llamaremos *logMessage* cuyos parámetros necesarios para identificar como middleware son *(req, res, next).*

Para comprobar que el parámetro *string* (el *surname*) que pasamos en la cabecera no contenga dígitos, declaramos una *regex* en una constante que usaremos como test de coincidencias. Esta *regex* identifica si el *string* contiene dígitos : */\d/gi.*

La constante *surname* es el identificador que pasamos por el método *get* en la cabecera de la *URL*., recogido por el método *req.params* del *middleware*.

Si el test encuentra que algún caracter del identificador es un dígito, devuelve una respuesta por medio del método *res*:

* *res.status(400).send(‘mensaje de error’),* con el mensaje que indica el error.

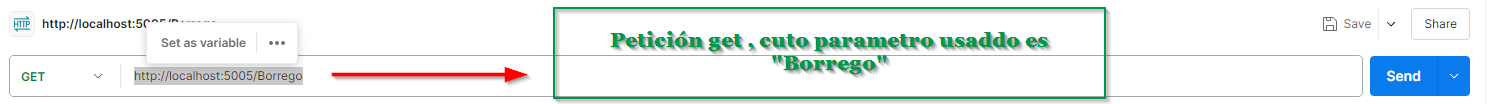
Si el test no identifica ningún dígito proseguimos con la secuencia de código por medio de el método *next().*

Por último importamos la función con *module.exports*, que será recogida en el archivo raíz *app.js (por medio de un require)*, que mostrará las coincidencias en formato *json()* de los *alumnos* de la tabla definida en */data/alumns.js.*

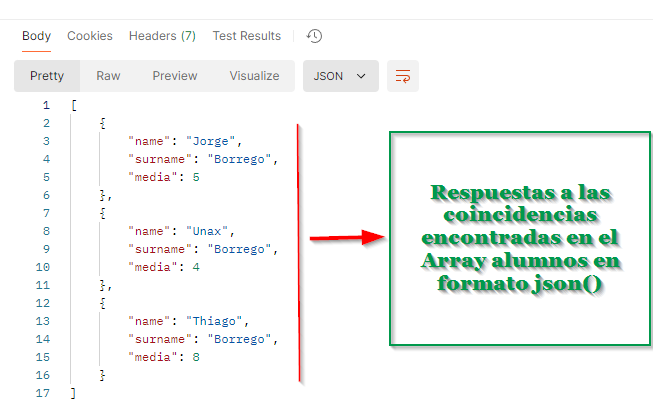
1. **Comprueba una petición "get" con Postman que haga ejecutar la función middleware de validación.**

Para comprobar que los resultados sean los esperados según el bloque de programación anterior, usamos Postman para realizar las peticiones get:

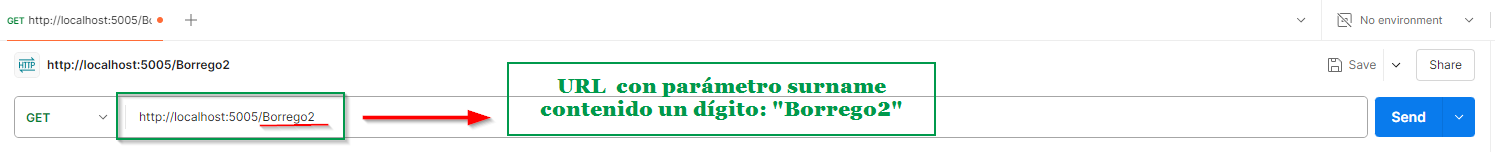
* [*http://localhost:5005/Borrego*](http://localhost:5005/Borrego)



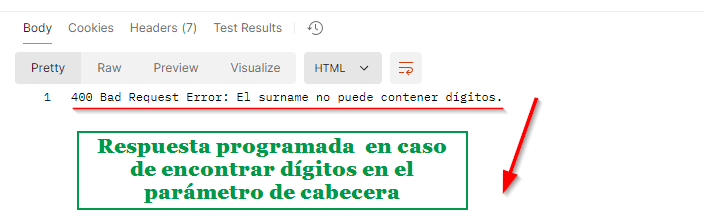
* Body de la respuesta (formato *json()*):



* [*http://localhost:5005/Borrego*](http://localhost:5005/Borrego)*2*

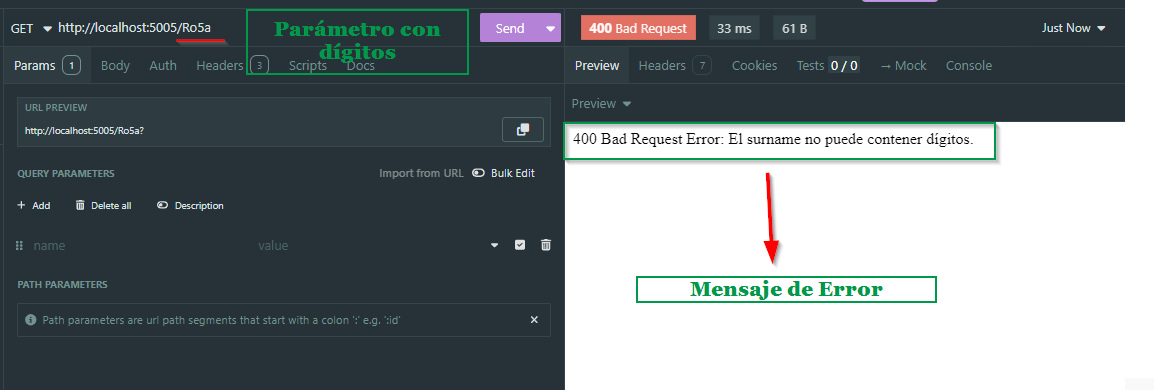


* Body de la respuesta (formato *json()*):



Mostramos dos ejemplos más con otros parámetros en una vista global de la respuesta (en este caso el programa para realizar las peticiones:“*Insomnia*”):

1. Ejemplo: Parámetro con dígitos.



1. Ejemplo: Parámetro sin dígitos:

