

FULL STACK DEVELOPER

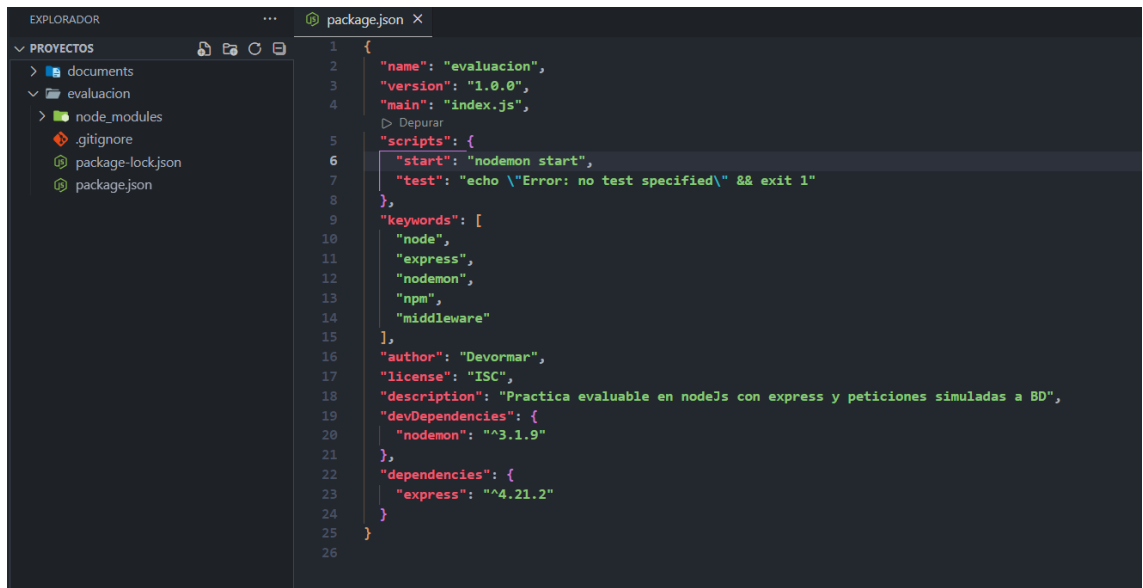
Módulo 5 – NodeJS

Jorge Borrego Marqués

Prueba Evaluable

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

- Folder del Proyecto: PROYECTOS/evaluacion



```
1 {
2   "name": "evaluacion",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "start": "nodemon start",
7     "test": "echo \\Error: no test specified\\' && exit 1"
8   },
9   "keywords": [
10    "node",
11    "express",
12    "nodemon",
13    "npm",
14    "middleware"
15  ],
16  "author": "Devormar",
17  "license": "ISC",
18  "description": "Practica evaluable en nodejs con express y peticiones simuladas a BD",
19  "devDependencies": {
20    "nodemon": "^3.1.9"
21  },
22  "dependencies": {
23    "express": "^4.21.2"
24  }
25 }
```

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

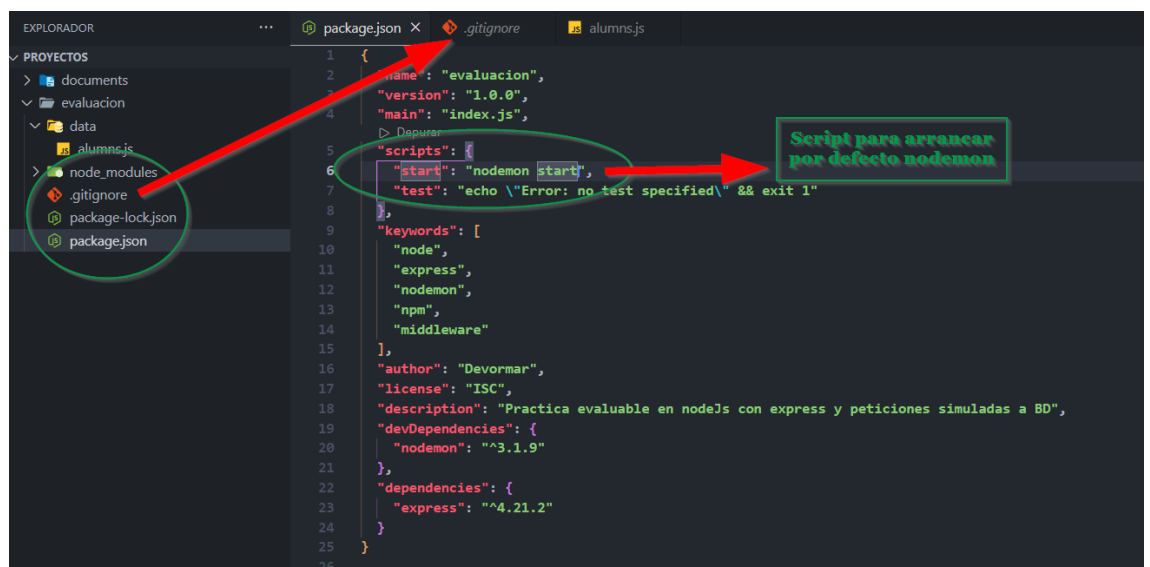
- Inicializo un Repositorio nuevo de git en la carpeta ./Proyectos:
git init
- Añado una carpeta nueva en la carpeta \Proyectos\evaluacion, donde inicializamos un proyecto nuevo *npm init -y*.

- Instalación de la devDependencia: `npm install nodemon --save-dev`
- Instalación de la Dependencia: `npm install express --save`

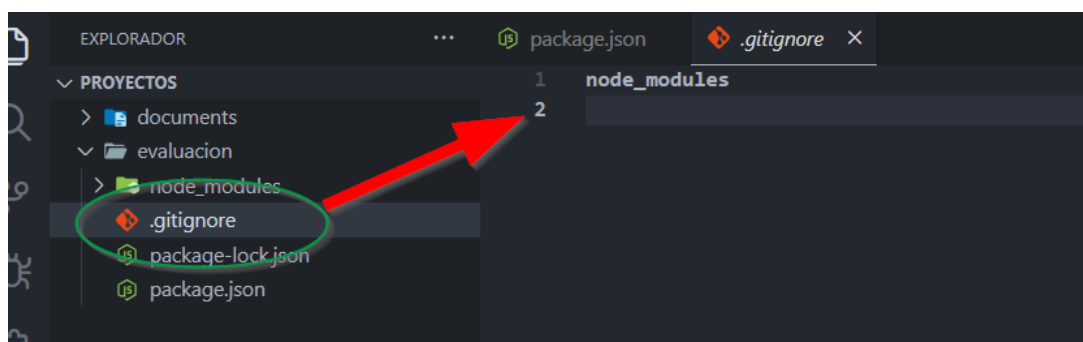
```
"devDependencies": {
  "nodemon": "^3.1.9"
},
"dependencies": {
  "express": "^4.21.2"
}
```

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

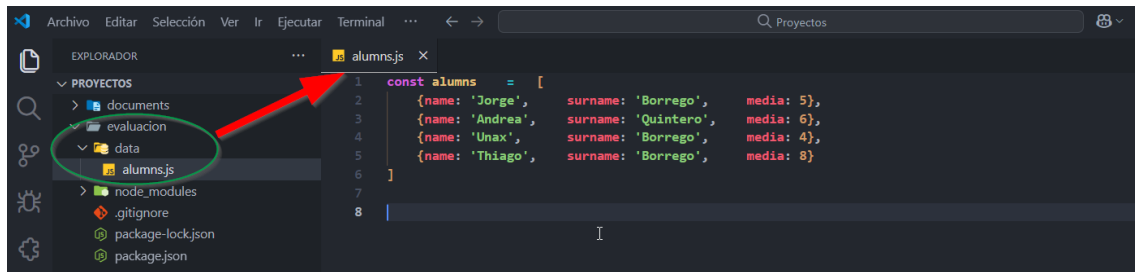
- Script para arrancar por defecto nodemon:



- Añadimos archivo .gitignore y en el cuerpo node_modules:



4. 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.

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

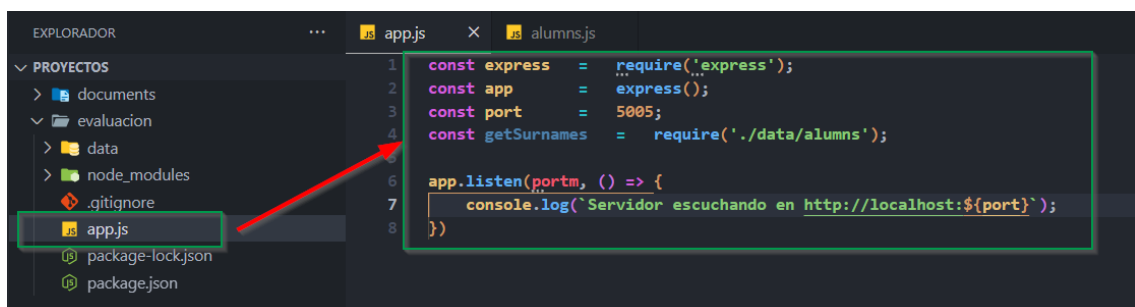
En archivo `./data/alumns.js` creamos una función que recibe como parámetro un string, y que retorna un subArray con todas las coincidencias de alumnos cuyo apellido coincide con el parámetro pasado en la función.



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.

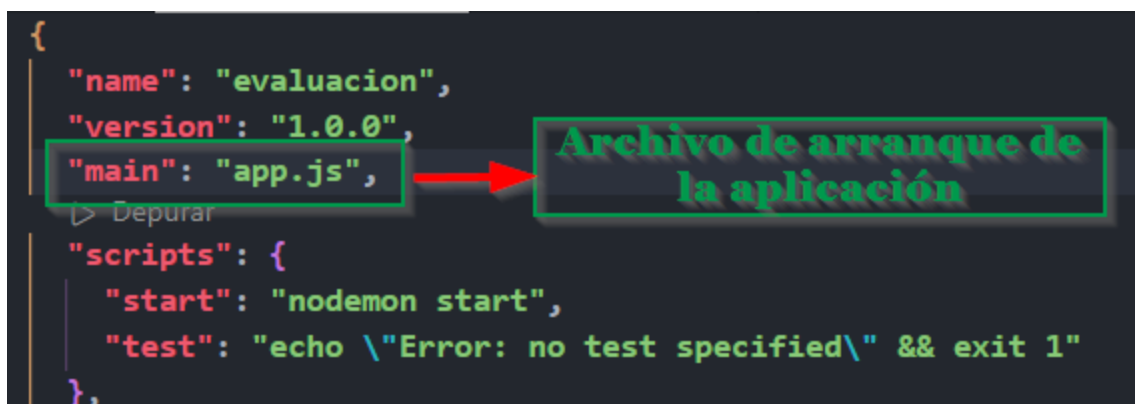
Exportamos la función por medio de *module.exports* para poder trabajar desde el archivo *app.js* con los valores , y que recogeremos por medio de un *require()*.

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



```
1 const express = require('express');
2 const app = express();
3 const port = 5005;
4 const getSurnames = require('./data/alumnos');
5
6 app.listen(port, () => {
7   console.log(`Servidor escuchando en http://localhost:${port}`);
8 })
```

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:



```
{
  "name": "evaluacion",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "nodemon start",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  }
},
```

Archivo de arranque de la aplicación

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

```
JS app.js  X  JS alumns.js
1  const express = require('express');
2  const app     = express();
3  const port    = 5005;
4  const getSurnames = require('./data/alumns');
5
6  app.listen(port, () => {
7    console.log(`Servidor escuchando en http://localhost:${port}`);
8  })
```

- 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*.

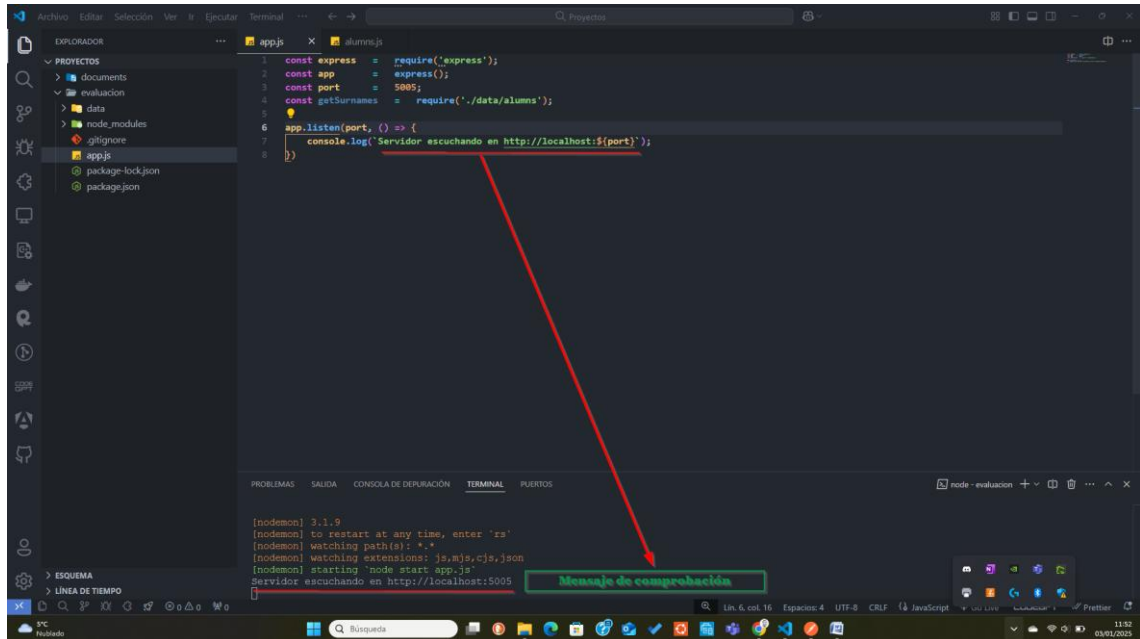
The screenshot shows a code editor with two files: `app.js` and `alumns.js`. The `app.js` file contains the following code:

```
1  const express = require('express');
2  const app     = express();
3  const port    = 5005;
4
5
6  app.listen(port, () => {
7    console.log(`Servidor escuchando en http://localhost:${port}`);
8  })
```

Annotations in the image:

- A green box highlights the code from line 1 to line 4, with the text: "Bloque de código, donde disponemos del servidor".
- A red arrow points from the `app.listen` method call to a green box with the text: "Usamos método listen del Objeto del servidor express para obtener mensaje por consola al arrancar el servidor".
- The terminal at the bottom shows the command `npm start` being executed, with a green box highlighting it and the text: "Comando de arranque del servidor".

Una vez arrancado el servidor comprobamos la respuesta por consola:



7. 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):



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

```
const express = require('express');
const app = express();
const port = 5005;
const getSurnames = require('./data/alumns');

app.listen(port, () => {
  console.log(`Servidor escuchando en http://localhost:${port}`);
})

app.get('/respuesta-json/:id', (req, res) => {
  res.status(200).json(getSurnames(req.params.id));
})
```

Método get del objeto del servidor por el cual recibimos en formato json los valores coincidentes del Array *alumns*

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*":

9. 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*:

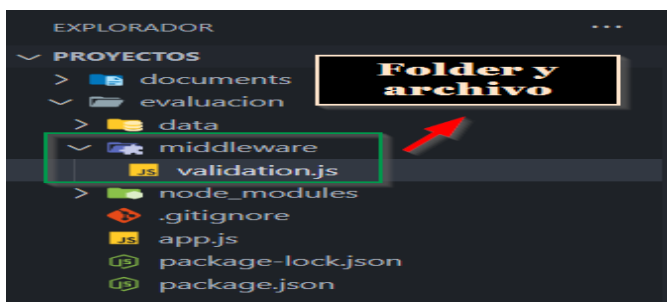


- Respuesta:

```
Body Cookies Headers (7) Test Results | ↺
Pretty Raw Preview Visualize JSON ↕
1  [
2    {
3      "name": "Jorge",
4      "surname": "Borrego",
5      "media": 5
6    },
7    {
8      "name": "Unax",
9      "surname": "Borrego",
10     "media": 4
11   },
12   {
13     "name": "Thiago",
14     "surname": "Borrego",
15     "media": 8
16   }
17 ]
```


10. 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.

```
1
2  const logMessage = (req, res, next) => {
3    const regex = /\d/gi;
4    const surname = req.params.id;
5
6    if (regex.test(surname) === true)
7      res.status(400).send('Error: El surname no puede contener dígitos.')
8    next();
9  }
10
11  module.exports = logMessage;
```

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 carácter 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*.

11. 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*



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

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "name": "Jorge",
4     "surname": "Borrego",
5     "media": 5
6   },
7   {
8     "name": "Unax",
9     "surname": "Borrego",
10    "media": 4
11  },
12  {
13    "name": "Thiago",
14    "surname": "Borrego",
15    "media": 8
16  }
17 ]
  
```

Respuestas a las coincidencias encontradas en el Array alumnos en formato json()

- <http://localhost:5005/Borrego2>

GET http://localhost:5005/Bk

GET http://localhost:5005/Borrego2

URL con parámetro surname contenido un dígito: "Borrego2"

Send

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

Body Cookies Headers (7) Test Results

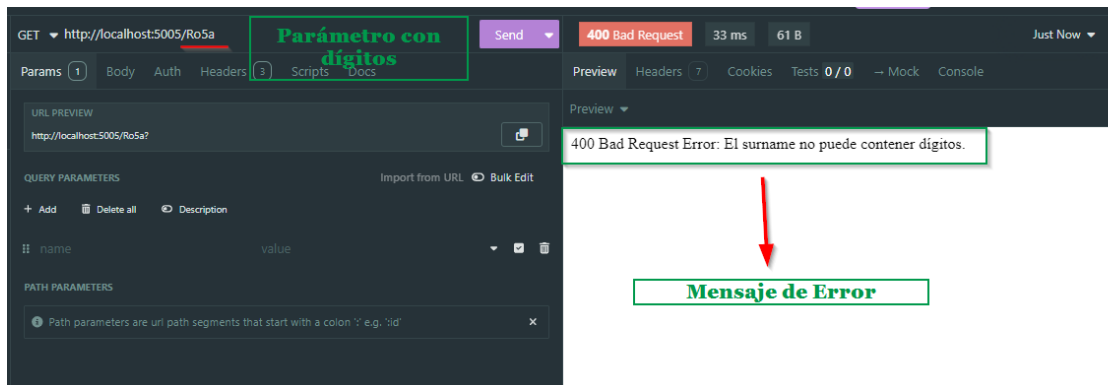
Pretty Raw Preview Visualize HTML

1 400 Bad Request Error: El surname no puede contener digitos.

Respuesta programada en caso de encontrar dígitos en el parámetro de cabecera

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

I. Ejemplo: Parámetro con dígitos.



II. Ejemplo: Parámetro sin dígitos:

