


Herramientas de desarrollo

 gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/configurando-el-ambiente-de-desarrollo.html

Antes de continuar vamos a mezclar el *feature* anterior y crear una rama para trabajar:

```
git checkout master
git merge feature/02-using-express
git checkout -b feature/03-add-development-tools
```

Nodemon

Es muy tedioso cada vez que realizamos un cambio en los archivos tener que detener e iniciar nuevamente la aplicación, para esto vamos a utilizar una librería llamada *nodemon*, esta vez como una dependencia solo para desarrollo, ya que no la necesitamos cuando la aplicación esté corriendo en producción, esta librería nos brinda la opción de que después de guardar cualquier archivo en el directorio de trabajo, nuestra aplicación se vuelva a reiniciar automáticamente.

```
npm install -D nodemon
```

Luego modificamos la sección de `scripts` del archivo `package.json` para añadir el *script* que ejecutará nuestra aplicación en modo de desarrollo y en modo de producción de la siguiente manera:

```
"scripts": {
  "dev": "nodemon",
  "start": "node index",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

No es necesario indicarle a *nodemon* que archivo va a ejecutar pues por convención buscará en archivo `index.js`, el cual es el punto de entrada de nuestra aplicación. Mas adelante se configurará el *script* de pruebas: `test`

Si el servidor se está ejecutando lo detenemos con `CTRL+C`, pero esta vez lo ejecutamos con `npm run dev` y abrimos el navegador en la siguiente dirección `http://localhost:3000`.

Antes de finalizar hacemos commit de nuestro trabajo:

```
git add .
git commit -m "Add Nodemon and npm scripts"
```

Más información:

[Nodemon](#)

ESLint

Es recomendado añadir una herramienta que nos compruebe la sintaxis de los archivos para evitar posibles errores y porque no normalizar la forma en que escribimos el código, para ello utilizaremos una librería llamada ESLint que junto con el editor de texto comprobará la sintaxis de los archivos en tiempo real:

```
npm install -D eslint
```

ESLint nos permite seleccionar una guía de estilo ya existente o también poder crear nuestra propia guía, todo depende del común acuerdo del equipo de desarrollo.

Inicializamos la configuración con el siguiente comando:

```
./node_modules/.bin/eslint --init
```

Si ESLint está instalado de manera global, el comando anterior se podría reemplazar con:

```
eslint --init
```

Contestamos las preguntas del asistente de configuración de la siguiente manera:

```
? How would you like to configure ESLint? Use a popular style guide
? Which style guide do you want to follow? Airbnb
? Do you use React? No
? What format do you want your config file to be in? JSON
```

Una vez finalizado el proceso creará un archivo en la raíz de nuestro proyecto llamado `.eslintrc.json`, el cual contiene toda la configuración. Adicionalmente vamos a añadir las siguientes excepciones:

1. Vamos a utilizar el objeto global `console` para imprimir muchos mensajes en la consola y no queremos que lo marque como error.
2. A modo de ejemplo mas adelante vamos añadir un parámetro que no utilizaremos inmediatamente pero nos servirá mucho para explicar un concepto el cual es `next` y no queremos que no los marque como error.

```
{
  "extends": "airbnb-base",
  "rules": {
    "no-console": "off",
    "no-unused-vars": ["error", {
      "argsIgnorePattern": "next"
    }]
  }
}
```

Si estas utilizando VSCode puedes añadir estos parámetros en la configuración para que siempre elimine los espacios en blancos restantes a la hora de guardar el archivo pero a su vez también deja una línea en blanco al final de cada archivo, estas son buenas prácticas y son requeridas por defecto en el ESLint:

```
{
  "editor.formatOnSave": true,
  "files.insertFinalNewline": true,
}
```

Antes de finalizar hacemos commit de nuestro trabajo

```
git add --all  
git commit -m "Add ESLint and config"
```

Más información:

[ESLint](#)