**Using EditorConfig and ESLint in Node.js Project**

There are a few tools available to help you to write quality code. In this post, we will show you how to set up Node.js project with [EditorConfig](https://editorconfig.org/) and [ESLint](https://eslint.org/docs/user-guide/getting-started).

EditorConfig is currently one of the most popular tools to maintain consistent coding styles between different editors or IDEs (for example, between ATOMS and Visual Studio Code). Some of the editors already have it pre-installed. For others, you need to install it. For example, [here](https://marketplace.visualstudio.com/items?itemName=chrisdias.vscodeEditorConfig) is the instruction for Visual Studio Code.

Once your editor has the plugin for EditorConfig installed, all you need to do is to create a .editorconfig file in the root directory of your project. You can check all the options available in the official website [here](https://editorconfig.org/).

**EditorConfig**

First of all, you need to set root = true so that the project uses this config file for all the target files. The target files are defined in []. In this example, the first set of rules applies to all js, yml and json files and the second set to .md file.

The line ending file format is line feed (lf), which is used for Unix/Linux. Alternatively, you can use carriage return and line feed (crlf) for Windows or carriage return (cr) for Mac in some case.

It also adds a newline at the end of the code if there is none, removes trailing whitespace and make all indentation to tab. The character set is utf-8

The change will happen automatically once you save your code file.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

root = true

[\*.{js,yml,json}]

end\_of\_line = lf

insert\_final\_newline = true

charset = utf-8

indent\_style = tab

trim\_trailing\_whitespace = true

[\*.md]

insert\_final\_newline = **false**

trim\_trailing\_whitespace = **false**

[/cc]

**ESLint**

[Linters](https://en.wikipedia.org/wiki/Lint_(software)) are tools to analyse source code to flag syntactical error. There are a few Javascript linters available. [ESLint](https://eslint.org/docs/user-guide/getting-started) is probably the most widely used one at the moment.

First of all, we need to install eslint as dev dependencies for your project.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

npm i eslint --save-dev

yarn add eslint --dev

[/cc]

Then, add scripts in package.json so that we can run it with npm command.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

"scripts" : {

    "eslint": "eslint"

  },

[/cc]

ESLint needs a config file (.eslintrc.js) because no rules are enabled by default. It is fully customisable and you can implement your own code standard with the config file. You can also use the pre-existing ones by extension. The best place to start with is to use [recommended rule](https://eslint.org/docs/rules/) by extending the rule.

The config file is a Javascript file. Here is the example of how you can use it. The extension can be added by the extends property. You can specify ECMA version in the parser option. The example uses 2018. For ES6, you can use es6.

<pre class=" toolbar:2 lang:js decode:true" >

module.exports = {

    "parserOptions" : {

        "ecmaVersion": "2018",

        "sourceType": "module"

    },

    "env": {

        "node": true,

        "es6": true

    },

    "extends":"eslint:recommended",

    "rules": {

        "no-empty": "error",

        "no-multiple-empty-lines": "warn",

        "no-var": "error",

        "perfer-const": "error"

    }

}

</pre>

You can also ignore a certain file or entire folder with .eslintingore file. This works exactly the same as .gitignore file. The example below ignores a file called reference.js and the entire node\_modules folder.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

node\_modules/\*

reference.js

[/cc]

To check run the linter you can simply use npm command. You need to add the path when you run it. The dot means all the files except the ones in .eslintignore.

In most cases, using the code editor’s extension plug in to do lint check rather than running it every time you edit your code. But, in some situations, knowing how to set up and run eslint this way can be helpful.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

npm run eslint .

[/cc]

Fun!