ESLint e Prettier - Trilha Node.js

Faaala Dev!

Nesse guia você verá o passo a passo de como realizar a configuração do ESLint, Prettier e EditorConfig da mesma forma como é usado nas aulas.

Mesmo que em alguma aula você identifique que possui alguma configuração que a instrutora Daniele Leão não possui, fica tranquilo(a). Isso é porque ela foi adicionando ao longo das aulas mas eu já to te passando tudinho aqui com antecedência, beleza?

Mesmo assim, sempre que ela adicionar uma nova configuração, dá uma conferida só pra ter certeza de que você já possui ela 😉.

Introdução

Uma ferramenta que nos auxilia no momento de padronizarmos o nosso projeto, e talvez seja a mais importante, é o **Eslint**. Com ele conseguimos automatizar os **padrões de códigos** do nosso projeto, e podemos utiliza-lo para projetos em **NodeJS, ReactJS e React Native**.

Por exemplo, no **Javascript** o uso do **ponto e vírgula** ao final de uma linha é **facultativo**, ou seja, diferente de algumas linguagens, a falta dele não interfere para que o código seja compilado. Outra utilização que também é opcional é o uso de **aspas duplas** ou **aspas simples**.

Já quando estamos criando um objeto, o uso da **vírgula** no último item do objeto também é opcional, como podemos ver no exemplo abaixo.

```
const aluno = { nome: "Mariana", idade: 20, }; const aluno = { nome:
"Daniel", idade: 21 };
```

No primeiro objeto utilizamos **vírgula** após o valor dentro do atributo **idade**, já no segundo não utilizamos, o que **não** interfere na execução do código.

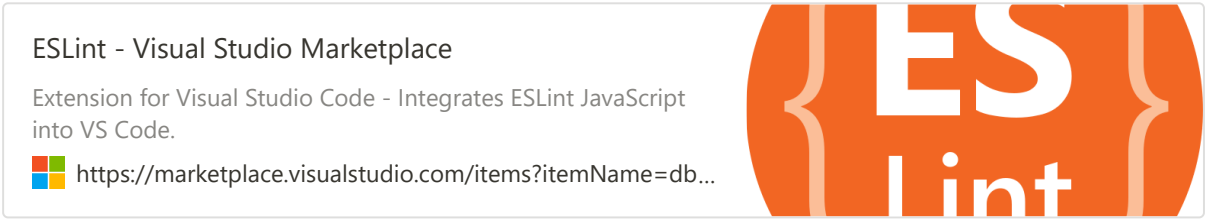
O **Eslint** integra, não somente para o **VSCode,** mas também com qualquer outro tipo de editor, o que mais uma vez ajuda na padronização do código, caso um outro desenvolvedor esteja desenvolvendo no mesmo projeto, mas não queira usar o **VSCode**.

O **Prettier** é mais uma ferramenta que vamos utilizar para ajudar na padronização de código, ele consiste em várias configurações que são feitas para que o código seja formatado para seguir um padrão.

Alguns exemplos de formatações que ele faz é a quebra de linha quando ela tem mais de 80 caracteres, adicionar `;` no final das linhas dentre outras funcionalidades muito úteis para um projeto.

Instalação

Antes de iniciar de fato a configuração do **Eslint** em nosso projeto, precisamos instalar a **extensão** do **Eslint** no **VSCode.** É ela quem irá nos auxiliar para que nossas configurações sejam entendidas dentro do nosso código.



Uma outra configuração que é geral e precisamos fazer para o **VSCode** formatar o código sempre que salvarmos algum arquivo é adicionar uma opção chamada `codeActionsOnSave` nas configurações, assim como mostrado abaixo:

```
"editor.codeActionsOnSave": { "source.fixAll.eslint": true }
```

Eslint

Pra começar, vamos instalar o **Eslint** como uma dependência de desenvolvimento dentro do nosso projeto **NodeJS**.

```
yarn add eslint -D
```

Após a instalação, precisamos inicializar o **eslint** pra conseguirmos inserir as configurações dentro do projeto.

Faremos isso inserindo o seguinte código no terminal:

```
yarn eslint --init
```

Ao inserir a linha acima, serão feitas algumas perguntas para configuração do projeto, conforme iremos ver à seguir:

1 - How would you like do use Eslint? (Qual a forma que queremos utilizar o **Eslint**)

- **To check syntax only** ⇒ Checar somente a sintaxe
- **To check syntax and find problems** ⇒ Checar a sintaxe e encontrar problemas
- **To check syntax, find problems and enforce code style** ⇒ Checar a sintaxe, encontrar problemas e forçar um padrão de código

Nós iremos escolher a última opção `To check syntax, find problems and enforce code style`.

2 - What type of modules does your project use? (Qual tipo de módulo seu projeto usa?)

- **JavaScript modules (import/export)**
- **CommonsJS (require/exports)**

Como em nosso projeto estamos utilizando o **Typescript,** vamos selecionar a **primeira** opção `Javascript modules (import/export)`

3 - Which framework does your project use? (Qual framework seu projeto está utilizando?)

- **React**
- **Vue.JS**
- **None of these**

Como estamos configurando o nosso **backend** vamos escolher a opção `None of these`

4 - Does your project use TypeScript? (Seu projeto está utilizando Typescript?)

- **No**
- **Yes**

Vamos selecionar a opção `Yes`.

5 - Where does your code run? (Onde seu código está rodando?)

- **Browser**
- **Node**

Vamos selecionar a opção **Node**, para isso, utilizamos a tecla `Espaço` para desmarcar o **Browser** e selecionarmos a opção `Node`

6 - How would you like to define a style for your project? (Qual guia de estilo queremos utilizar?)

- **Use a popular style guide ⇒** Padrões de projetos já criados anteriormente por outra empresa
- **Answer questions about your style ⇒** Criar seu próprio padrão de projeto

Vamos selecionar a primeira opção `Use a popular style guide`

7 - Which style guide do you want to follow? (Qual guia de estilo você deseja seguir?)

- **Airbnb:** https://github.com/airbnb/javascript

- **Standard:** https://github.com/standard/standard
- **Google:** https://github.com/google/eslint-config-google

Nós iremos utilizar a primeira opção `Airbnb`. Com ela, nós vamos definir que nosso projeto utilizará **ponto e vírgula** ao final de cada linha, utilizará **aspas simples** e algumas outras configurações. Para saber todas as possíveis configurações, acessar a documentação da guia desejada.
Lembrando que, não há um padrão correto, nós iremos utilizar o **Airbnb**, porém você pode utilizar qualquer guia, desde que seu time todo também esteja utilizando.

8 - What format do you want your config file to be in? (Qual formato de configuração do Eslint que você deseja salvar?)

- **Javascript**
- **YAML**
- **JSON**

Vamos selecionar a opção `JSON`.

Depois que respondemos as perguntas, o **ESLint** irá informar quais as dependências necessárias de acordo com a sua configuração e pedir para instalá-las automaticamente.

9 - Would you like to install them now with npm? (Você deseja instalar as dependências agora utilizando npm?)

Caso estivéssemos utilizando o **NPM** a resposta seria `Yes`, mas como estamos utilizando o **Yarn** vamos responder `No` e adicionar manualmente as dependências.

```
Checking peerDependencies of eslint-config-airbnb@latest The config that
you've selected requires the following dependencies: @typescript-
eslint/eslint-plugin@latest eslint-config-airbnb-base@latest
eslint@^5.16.0 || ^6.8.0 || ^7.2.0 eslint-plugin-import@^2.22.1
@typescript-eslint/parser@latest ? Would you like to install them now
with npm? No
```

Para adicionar manualmente as dependências, basta seguir os passos abaixo:

- Iniciar o comando com `yarn add` para instalar as dependências e a tag `-D` para adicioná-las como desenvolvimento;

- Copiar os pacotes listados acima removendo o `eslint@^5.16.0 || ^6.8.0 || ^7.2.0` pois já temos o **ESLint** instalado.

O comando final deve ter essa estrutura :

> ⚠️ **Não copie o comando abaixo. Utilize isso apenas como exemplo, pois as versões podem mudar**

```
yarn add -D @typescript-eslint/eslint-plugin@latest eslint-config-airbnb-base@latest eslint-plugin-import@^2.22.1 @typescript-eslint/parser@latest
```

Precisamos também instalar um plugin que irá nos auxiliar a organizar a ordem dos imports dentro dos arquivos e outro para permitir importações de arquivos TypeScript sem que precisemos passar a extensão do arquivo:

```
yarn add -D eslint-plugin-import-helpers eslint-import-resolver-typescript
```

Com as dependências instaladas vamos criar na raiz do projeto um arquivo `.eslintignore` com o conteúdo abaixo para ignorar o Linting em alguns arquivos:

```
/*.js node_modules dist
```

Agora vamos começar a configuração do arquivo que foi gerado na inicialização do **ESLint**, o `.eslintrc.json` , a primeira coisa a ser feita é adicionar dentro de `"env"` a linha:

```
"jest": true
```

Ainda dentro de `"env"`, verifique se a primeira linha está como `"es2020": true`, caso contrário faça a alteração deixando assim.

O próximo passo é adicionar dentro de `"extends"` a linha:

```
"plugin:@typescript-eslint/recommended"
```

Agora, precisamos configurar o plugin que instalamos para que seja usado pelo ESLint. Para isso, adicione o seguinte dentro de `"plugins"`:

```
"eslint-plugin-import-helpers"
```

Em seguida, adicionamos dentro de `"rules"` as seguintes configurações:

```
"camelcase": "off", "import/no-unresolved": "error", "@typescript-
eslint/naming-convention": [ "error", { "selector": "interface",
"format": ["PascalCase"], "custom": { "regex": "^I[A-Z]", "match": true }
} ], "class-methods-use-this": "off", "import/prefer-default-export":
"off", "no-shadow": "off", "no-console": "off", "no-useless-constructor":
"off", "no-empty-function": "off", "lines-between-class-members": "off",
"import/extensions": [ "error", "ignorePackages", { "ts": "never" } ],
"import-helpers/order-imports": [ "warn", { "newlinesBetween": "always",
"groups": ["module", "/^@shared/", ["parent", "sibling", "index"]],
"alphabetize": { "order": "asc", "ignoreCase": true } } ], "import/no-
extraneous-dependencies": [ "error", { "devDependencies":
["**/*.spec.js"] } ]
```

Por fim, para que o **Node.js** consiga entender arquivos **Typescript** é necessário acrescentar uma configuração adicional nas importações pois por padrão vai ser apresentado um erro dizendo que as importações de arquivos **Typescript** não foram resolvidas. Para resolver isso basta adicionar logo **abaixo** das `"rules"` no `.eslintrc.json` o seguinte:

```
"settings": { "import/resolver": { "typescript": {} } }
```

Para finalizar e aplicar todas as mudanças vamos fechar o VS Code e reabrir na **pasta raiz** do projeto, pois senão o **ESLint** não vai reconhecer as dependências instaladas e aplicar as regras de Linting.

Feito isso, para verificar se está realmente funcionando basta reabrir qualquer arquivo do projeto e tentar errar algo no código para que ele mostre o erro e formate automaticamente quando o arquivo for salvo.

O arquivo `.eslintrc.json` finalizado com todas as mudanças tem que ficar assim:

```
{ "env": { "es2020": true, "node": true, "jest": true }, "extends": [
"airbnb-base", "plugin:@typescript-eslint/recommended" ], "parser":
"@typescript-eslint/parser", "parserOptions": { "ecmaVersion": 12,
"sourceType": "module" }, "plugins": [ "@typescript-eslint", "eslint-
plugin-import-helpers" ], "rules": { "camelcase": "off", "import/no-
unresolved": "error", "@typescript-eslint/naming-convention": [ "error",
{ "selector": "interface", "format": ["PascalCase"], "custom": { "regex":
"^I[A-Z]", "match": true } } ], "class-methods-use-this": "off",
"import/prefer-default-export": "off", "no-shadow": "off", "no-console":
"off", "no-useless-constructor": "off", "no-empty-function": "off",
"lines-between-class-members": "off", "import/extensions": [ "error",
"ignorePackages", { "ts": "never" } ], "import-helpers/order-imports": [
"warn", { "newlinesBetween": "always", "groups": ["module", "/^@shared/",
["parent", "sibling", "index"]], "alphabetize": { "order": "asc",
"ignoreCase": true } } ], "import/no-extraneous-dependencies": [ "error",
{ "devDependencies": ["**/*.spec.js"] } ] }, "settings": {
"import/resolver": { "typescript": {} } } }
```

Prettier

> 💡 ⚠️ Antes de começar a configuração é importante que você se certifique de remover a extensão **Prettier - Code Formatter** do seu VS Code, ela pode gerar incompatibilidades com as configurações que vamos fazer.

A primeira coisa que vamos fazer para a configuração do **Prettier** é a instalação dos pacotes no projeto, e faremos isso executando:

```
yarn add prettier eslint-config-prettier eslint-plugin-prettier -D
```

Esse comando vai adicionar 3 dependências que serão as responsáveis por fazer a formatação do código e também integrar o **Prettier** com o **ESLint**.

Com a instalação feita vamos modificar o arquivo `.eslintrc.json` adicionando no `"extends"` as seguintes regras:

```
"prettier", "plugin:prettier/recommended"
```

Nos `"plugins"` vamos adicionar apenas uma linha com:

```
"prettier"
```

E nas `"rules"` vamos adicionar uma linha indicado para o **ESLint** mostrar todos os erros onde as regras do **Prettier** não estiverem sendo seguidas, como abaixo:

```
"prettier/prettier": "error"
```

O arquivo final vai ficar assim:

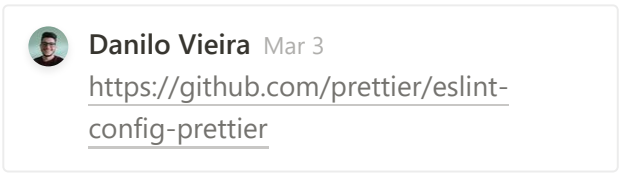
```
{ ... "extends": [ ... "prettier", "plugin:prettier/recommended" ], ...
"plugins": [ ... "prettier" ], "rules": { ... "prettier/prettier":
"error" }, ... }
```

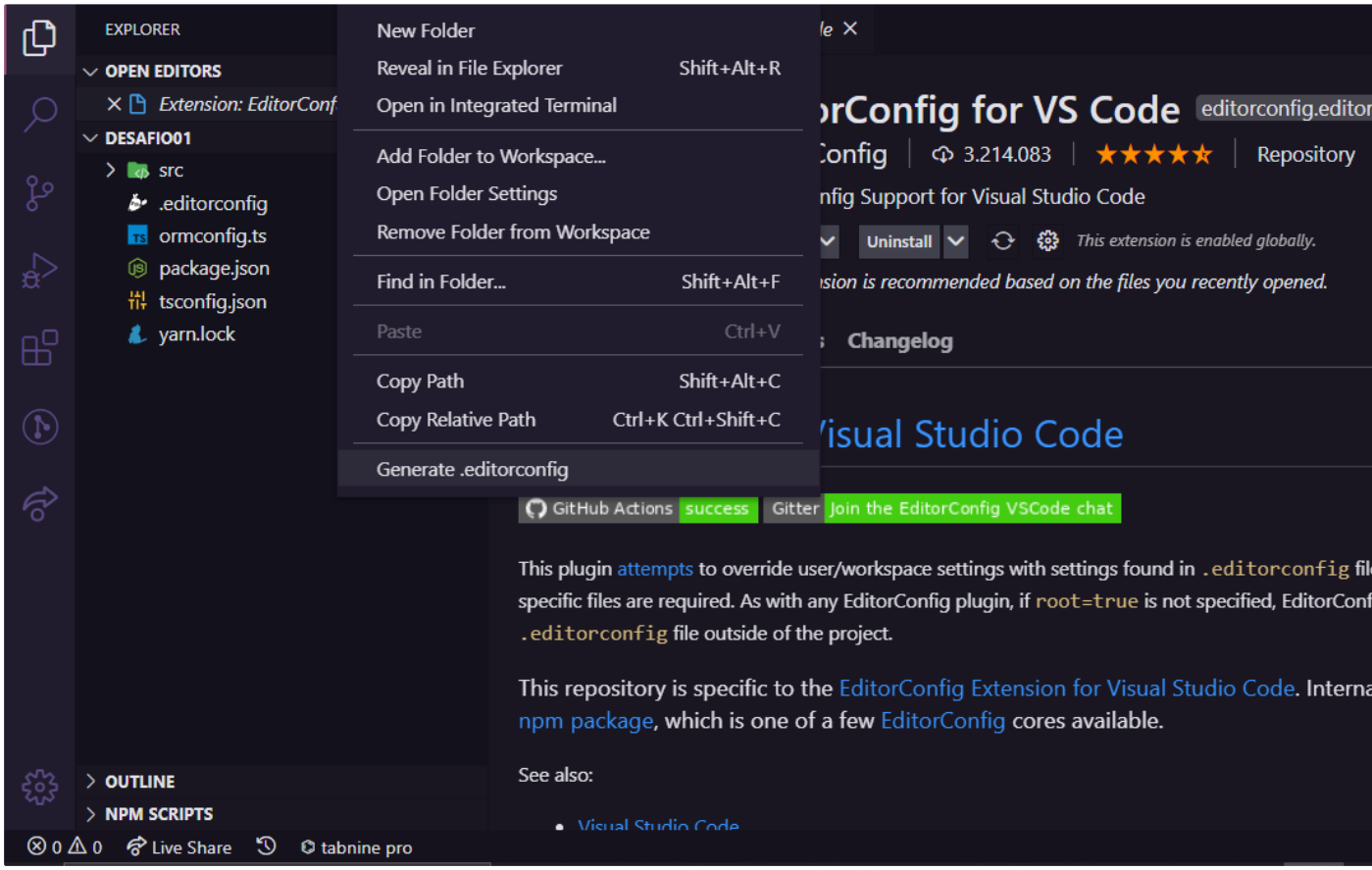
E a configuração está finalizada. Para garantir que o código seja formatado corretamente, você pode abrir os arquivos do projeto e salvar eles novamente.

Problemas no Windows

É provável que você enfrente alguns problemas de conflito entre o ESLint e o tipo de quebra de linha no Windows. Isso acontece porque quando usamos a quebra de linha no Windows, ela é interpretada como `\r\n` enquanto em sistemas Unix é `\n`. Como o ESLint tenta sempre corrigir para `\n`, esse conflito acaba acontecendo.

Para padronizar o tipo de quebra de linha usada pelo VS Code no Windows, iremos instalar uma extensão chamada EditorConfig for VS Code. Com ela instalada, na pasta raiz dos nossos projetos podemos clicar com o botão direito do mouse e escolher a opção `Generate .editorconfig`:





Com o arquivo criado você já está pronto para continuar. Todas as quebras de linha estarão no formato esperado pelo ESLint.