

Dev Tools: ESlint

Aleksandar Zajic • 12 / 21 / 2017

ESLint

Created

June 2013

Nicholas C. Zakas

- As pluggable linting utility for JavaScript

About ESLint

- a type of static analysis that is frequently used to find problematic patterns or code that doesn't adhere to certain style guidelines.
- JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error.
- Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it.
- The primary reason ESLint was created was to allow developers to create their own linting rules.
- ESLint is written using Node.js to provide a fast runtime environment and easy installation via npm.
- [npm] Eslint-config-airbnb-base
- [npm] Eslint-config-airbnb

Philosophy

- Everything is pluggable (rule and formatter api used by bundled and custom rules)
- Additional rules and formatters can be specified at runtime by `// enable` `//disable`
- Standalone rules, can be turned on/off or set to warning or error level individually

Error levels:

- 0 - off (“off”)
- 1 - warning (“warn”)
- 2 - error (“error”) // breaks the build

GOOD... GOOD...

**LET THE ESLINT FLOW THROUGH
YOU**

memegenerator.net

ESlint installation



Installation - Part 1

Requirements:

NODE.js, Project folder

Checking the latest versions

```
$ npm info "eslint-config-airbnb@latest" peerDependencies
```

Dependencies

- eslint
- eslint-plugin-import
- eslint-plugin-jsx-a11y
- eslint-plugin-react
- eslint-config-airbnb-base
- eslint-loader // for webpack

Installation - Part 2

In project folder:

```
$ yarn add eslint eslint-plugin-import eslint-plugin-jsx-a11y  
eslint-plugin-react eslint-config-airbnb-base eslint-loader --dev
```

Or

```
$ npm install --save-dev eslint eslint-plugin-import  
eslint-plugin-jsx-a11y eslint-plugin-react eslint-config-airbnb-base  
eslint-loader
```

Global install

```
$ npm install -g eslint
```


Installation - Part 3 - Webpack

In webpack.config.js:

```
module: {  
  rules: [  
    // eslint  
    {  
      enforce: 'pre',  
      test: /\.js$/,  
      loader: 'eslint-loader',  
      options: {  
        configFile: path.join(__dirname, '.eslintrc'),  
      },  
    },  
  ],  
}
```

ESlint usage



Simple `.eslintrc` configuration file

```
{
  "ecmaFeatures": {
    "jsx": true,
    "modules": true
  },
  "env": {
    "browser": true,
    "node": true
  },
  "parser": "babel-eslint",
  "rules": {
    "quotes": [2, "single"],
    "strict": [2, "never"],
    "react/jsx-uses-vars": 2,
  },
  "plugins": [
    "react"
  ],
  "extends": [
    "plugin:react/recommended",
    "airbnb-base"
  ]
}
```

`ecmaFeatures` - an object indicating which additional language features you'd like to use.

`env` - An environment defines global variables that are predefined.

`parser` - A wrapper around the Babel parser that makes it compatible with ESLint.

`Rules` - ESLint comes with a large number of rules. You can modify which rules your project uses either using configuration comments or configuration files. To change a rule setting, you must set the rule ID equal to one of these values: 0, 1, 2

`Plugins` - plugin is an npm package that usually exports rules

`Extends` - extends a set of rules from base configurations.

Changing Rules

change an inherited rule's severity without changing its options:

Base config: "eqeqeq": ["error", "allow-null"]

Derived config: "eqeqeq": "warn"

Resulting actual config: "eqeqeq": ["warn", "allow-null"]

override options for rules from base configurations:

Base config: "quotes": ["error", "single", "avoid-escape"]

Derived config: "quotes": ["error", "single"]

Resulting actual config: "quotes": ["error", "single"]

Disabling Rules in code

Inline Comments

```
/* eslint-disable */
```

```
alert('foo');
```

```
/* eslint-enable */
```

```
/* eslint-disable no-alert,  
no-console */
```

```
alert('foo');
```

```
console.log('bar');
```

```
/* eslint-enable no-alert,  
no-console */
```

Disabling Rules in code

Inline Comments

```
alert('foo'); // eslint-disable-line no-alert, quotes, semi
```

```
// eslint-disable-next-line
```

```
alert('foo');
```

Rules Explained





TRAILING SPACES
NOT ALLOWED

"prefer-destructuring": 1,

```
// bad
Const name = this.state.name
Const company = this.state.company
// good
Const {name, company} = this.state;
```

```
Let arr = [1,2,3,4]
// bad
const first = arr[0];
const second = arr[1];
// good
const [first, second] = arr;
var a, b, rest;
[a, b] = [10, 20];
console.log(a); // 10
console.log(b); // 20
```

```
[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(a); // 10
console.log(b); // 20
console.log(rest); // [30, 40, 50]
```

```
// bad
function getFullName(user) {
  const firstName = user.firstName;
  const lastName = user.lastName;
  return `${firstName} ${lastName}`;
}
```

```
// good
function getFullName(user) {
  const { firstName, lastName } = user;
  return `${firstName} ${lastName}`;
}
```

```
// best
function getFullName({ firstName, lastName }) {
  return `${firstName} ${lastName}`;
}
```

"arrow-parens": [2, "always"],

Bad

```
a => {};  
a => a;  
  
a => {'\n'};  
  
a.then(foo => {});  
  
a.then(foo => a);  
  
a(foo => { if (true) {} } );
```

Good

```
() => {};  
(a) => {};  
  
(a) => a;  
  
(a) => {'\n'}  
  
a.then((foo) => {});  
  
a.then((foo) => { if (true) {} } );
```

"arrow-parens": [2, "as-needed", { "requireForBlockBody": true }],

Bad

```
(a) => a;  
a => {};  
a => {'\n'};  
a.map((x) => x * x);  
a.map(x => {  
  return x * x;  
});  
a.then(foo => {});
```

Good

```
(a) => {};  
(a) => {'\n'};  
a => ({});  
() => {};  
a => a;  
a.then((foo) => {});  
a.then((foo) => { if (true) {} });  
a((foo) => { if (true) {} });  
(a, b, c) => a;
```

"arrow-body-style": [2, "always"]

Bad

```
(a) => a;  
a => {};  
a => {'\n'};  
a.map((x) => x * x);  
a.map(x => {  
  return x * x;  
});  
a.then(foo => {});
```

Good

```
(a) => {};  
(a) => {'\n'};  
a => ({});  
() => {};  
a => a;  
a.then((foo) => {});  
a.then((foo) => { if (true) {} });  
a((foo) => { if (true) {} });  
(a, b, c) => a;
```

AirBnB Configuration (Best Practice)



“Object-shorthand”: 2

```
// bad
const atom = {
  value: 1,
  addValue: function (value) {
    return atom.value + value;
  },
};
```

```
// good
const atom = {
  value: 1,
  addValue(value) {
    return atom.value + value;
  },
};
```

```
const lukeSkywalker = 'Luke Skywalker';
```

```
// bad
const obj = {
  lukeSkywalker: lukeSkywalker,
};
```

```
// good
const obj = {
  lukeSkywalker,
};
```

Do not call Object.prototype methods directly

// very bad

```
const original = { a: 1, b: 2 };
```

```
const copy = Object.assign(original, { c: 3 }); // this mutates `original`
```

```
delete copy.a; // so does this
```

// bad

```
const original = { a: 1, b: 2 };
```

```
const copy = Object.assign({}, original, { c: 3 }); // copy => { a: 1, b: 2, c: 3 }
```

// good

```
const original = { a: 1, b: 2 };
```

```
const copy = { ...original, c: 3 }; // copy => { a: 1, b: 2, c: 3 }
```

```
const { a, ...noA } = copy; // noA => { b: 2, c: 3 }
```

Use array spreads ... to copy arrays.

// bad

```
const len = items.length;  
const itemsCopy = [];  
let i;
```

```
for (i = 0; i < len; i += 1) {  
  itemsCopy[i] = items[i];  
}
```

// good

```
const itemsCopy = [...items];
```


Use array spreads ... to copy arrays.

// bad

```
const len = items.length;  
const itemsCopy = [];  
let i;
```

```
for (i = 0; i < len; i += 1) {  
  itemsCopy[i] = items[i];  
}
```

// good

```
const itemsCopy = [...items];
```

Use line breaks after open and before close array brackets if an array has multiple lines

```
// bad
const arr = [
  [0, 1], [2, 3], [4, 5],
];
const objectInArray = [{
  id: 1,
}, {
  id: 2,
}];
const numberInArray = [
  1, 2,
];
```

```
// good
const arr = [[0, 1], [2, 3], [4, 5]];

const objectInArray = [
  {
    id: 1,
  },
  {
    id: 2,
  },
];

const numberInArray = [
  1,
  2,
];
```

Function Signatures

```
// bad
function foo(bar,
  baz,
  quux) {
// ...
}
// good
function foo(
  bar,
  baz,
  quux,
) {
// ...
}
```

```
// bad
console.log(foo,
  bar,
  baz);

// good
console.log(
  foo,
  bar,
  baz,
);
```

FOLLOW THE RULES

FOOL!

memegenerator.net

Our .eslintrc config



Rules Explained

```
"key-spacing": [1, {  
  "singleLine": {  
    "beforeColon": false,  
    "afterColon": true  
  },  
  "multiLine": {  
    "beforeColon": false,  
    "afterColon": true  
  },  
  "align": {  
    "beforeColon": false,  
    "afterColon": true,  
    "on": "value"  
  }  
}],
```

```
static propTypes = {  
  
  user:      PropTypes.object,  
  
  className: PropTypes.string,  
  
  isActive:  PropTypes.bool,  
  
  toggleNav: PropTypes.func,  
  
  signOut:   PropTypes.func,  
  
};
```

```
"quote-props": [2, "consistent"],
```

// Always use same quotes

```
"class-methods-use-this": 0,
```

// If a class method does not use THIS , it can
sometimes be made into a static function

```
class A {  
  foo() {  
    this.bar = "Hello World"; // OK, this is used  
  }  
}
```

// exception constructor and static func.

```
"comma-dangle": [1, "always-multiline"],
```

// requires trailing commas when the last element
or property is in a different line than the closing]
or }

// bad

```
var foo = {  
  bar: "baz",  
  qux: "quux"  
};
```

```
var foo = { bar: "baz", qux: "quux", };
```

// good

```
var foo = {  
  bar: "baz",  
  qux: "quux",  
};
```

```
var foo = { bar: "baz", qux: "quux" };
```

```
"eol-last": 0,  
// Doesn't require empty line at the end of file
```

```
"global-require": 0,  
// require() can be used anywhere in the code
```

```
"import/prefer-default-export": 0,  
// When there is only a single export from a module,  
prefer using default export over named export.
```

```
"no-unused-vars": [1, {"vars": "local", "args":  
"after-used", "ignoreRestSiblings": false }],  
// When there is only a single export from a module,  
prefer using default export over named export.
```

```
"import/no-mutable-exports": 0,  
// Forbids the use of mutable exports with 'var' or  
'let'.
```

```
// good  
export const count = 1  
export function getCount() {}  
export class Counter {}
```

```
// bad  
export let count = 2  
export var count = 3  
  
let count = 4  
export { count } // reported here
```



```
"object-curly-newline": [1, {  
  "multiline": true  
}],
```

// Objects are defined in multiple lines

```
"object-curly-spacing": [1, "never"],
```

// no empty space after { and before }

```
"react/prop-types": [2, { "ignore": ["children",  
"location"]}],
```

// ES6 + Public Class Fields (draft:
<https://tc39.github.io/proposal-class-public-fields>
/)

```
class HelloEs6WithPublicClassField extends  
React.Component {  
  static propTypes = {  
    name: PropTypes.string.isRequired,  
  }  
  render() {  
    return <div>Hello {this.props.name}</div>;  
  }  
}
```

```
"react/jsx-key": 1,
```

// Warn if an element that likely requires a key prop--namely, one present in an array literal or an arrow function expression.

```
[  
  <Hello key="first" />,  
  <Hello key="second" />,  
  <Hello key="third" />,  
];  
  
data.map((x, i) => <Hello key={i}>{x}</Hello>);
```

```
"react/prop-types": [2, { "ignore": ["children",  
"location"]}],
```

```
// ES6 + Public Class Fields (draft:  
https://tc39.github.io/proposal-class-public-fields  
/)  
class HelloEs6WithPublicClassField extends  
React.Component {  
  static propTypes = {  
    name: PropTypes.string.isRequired,  
  }  
  render() {  
    return <div>Hello {this.props.name}</div>;  
  }  
}
```

```

"indent": [1, 4, {
    "VariableDeclarator": {"var": 1,
"let": 1, "const": 1 },
    "outerIIFEBody": 0,
    "MemberExpression": 1,
    "FunctionDeclaration":
{"parameters": 1},
    "FunctionExpression":
{"parameters": 1},
    "CallExpression": {"arguments":
1},
    "ArrayExpression": 1,
    "ObjectExpression": 1,
    "ImportDeclaration": 1,
    "flatTernaryExpressions": false,
    "ignoredNodes": [
        "ConditionalExpression",
        "CallExpression >
FunctionExpression.callee >
BlockStatement.body"
    ]
}],

```

```

// use 4 spaces for indents = 1 tab
// indent of comma separated variable
declarations,
// initial indent of function
// chained functions
// align multi-line function parameters
// multi-line call arguments align
// align 1x4 spaces = 1 tab
// indent for ternary expressions
// ignore

```



Next steps

VS Code Settings

```
"eslint.nodePath": "/usr/local/bin/node",
```

```
// Run the linter on save (onSave) or on type (onType)
```

```
"eslint.run": "onType",
```

```
// Turns auto fix on save on or off.
```

```
"eslint.autoFixOnSave": true,
```

Requirements

Eslint package for VS Code



Link to .eslintrc

[`https://goo.gl/es8J5n`](https://goo.gl/es8J5n)

References

<https://eslint.org/>

<https://github.com/yannickcr/eslint-plugin-react/>

<https://github.com/airbnb/javascript>

<https://github.com/Khan/style-guides>

<https://github.com/benmosher/eslint-plugin-import>

Questions?

