ESLint

ESLint란?

ESLint는 대부분의 텍스트 편집기에 내장되어 있으며, 코드를 분석하여 문제를 찾습니다.

ESLint는 ECMAScript / JavaScript 코드에서 발견되는 패턴을 식별하고 보고하는 도구로, 코드를 보다 일관성있게 만들고 버그를 피할 수 있도록 합니다. 여러면에서 JSLint 및 JSHint와 유사하지만 몇 가지 예외가 있습니다.

- ESLint는 Espree를 사용하여 자바 스크립트 구문 분석.
- ESLint는 AST를 사용하여 코드의 패턴을 평가합니다.
- ESLint는 완전한 플러그인형 이며, 모든 규칙은 런타임에 더 추가 할 수 있습니다.

Installation

SSL 지원으로 빌드 된 Node.js (`^8.10.0`, `^10.13.0`또는 `>=11.10.1`). (공식 Node.js 배포를 사용하는 경우 SSL은 항상 내장되어 있습니다.)

yarn

```
yarn add eslint --dev
```

npm

```
npm install eslint --save-dev
```

그런 다음 구성 파일을 설정해야합니다.

```
$ npx eslint --init
```

그 후에는 다음과 같은 파일이나 디렉토리에서 ESLint를 실행할 수 있습니다.

```
$ npx eslint yourfile.js
```

💡 ESLint를 로컬이 아닌 전역으로 설치할 수도 있습니다. 그러나 이것은 권장되지 않으며 사용하는 플러그인 또는 공유 가능한 구성은 모두 로컬로 설치해야합니다.

구성

eslint --init를 실행 하면 디렉토리에 .eslintrc 파일이 생깁니다. 여기에 다음과 같이 구성된 규칙이 표시됩니다.

```
{
    "rules": {
        "semi": ["error", "always"],
        "quotes": ["error", "double"]
    }
}
```

`"semi"` 과 `"quotes"는` ESLint 의 규칙 이름입니다 . 첫 번째 값은 규칙의 오류 수준이며 다음 값 중 하나일 수 있습니다.

- `"off"` 또는 `0` -규칙을 해제
- `"warn"` 또는 `1` - 규칙을 경고로 설정
- `"error"` 또는 `2` - 규칙을 오류로 설정

rules의 종류는 공식문서 참조.

구성 파일 확장

구성 파일은 기본 구성에서 사용 가능한 규칙 세트를 확장 할 수 있습니다.

```
{
    "extends": [
        "eslint:recommended", // eslint의 Default
        "plugin:vue/essential", // vue.js plugin
        "airbnb", // airbnb 규칙 확장
        "plugin:prettier/recommended" // Prettier plugin
    ],
    "rules": {
        "semi": ["error", "always"],
        "quotes": ["error", "double"]
    }
}
```

ESLint 환경 설정

.eslintrc 파일에 env를 통하여 환경을 설정할 수 있습니다.

```
{
    "env": {
      "browser": true,
      "node": true,
      "es6": true,
```

```
    },
    "extends": [
        "plugin:vue/essential",
        "@vue/airbnb",
        "@vue/typescript",
        "plugin:prettier/recommended"
    ],
    "rules": {
        // "no-console": process.env.NODE_ENV === "production" ? "error" : "off",
        "no-console": "off",
        "no-debugger": process.env.NODE_ENV === "production" ? "error" : "off",
        "class-methods-use-this": 0,
        "max-len": "off",
        "import/no-extraneous-dependencies": ["error", { devDependencies: true }],
        "no-plusplus": ["error", { allowForLoopAfterthoughts: true }], // for (let i = 0; i < 10; i++)
        "no-underscore-dangle": ["error", { allow: ["_data", "_name"] }], // for this._data
    }
}
```

환경은 사전 정의 된 전역 변수를 정의합니다. 사용 가능한 환경은 다음과 같습니다.

- `browser` - 브라우저 전역 변수.
- `node` - Node.js 전역 변수 및 Node.js 범위.
- `commonjs` - CommonJS 전역 변수 및 CommonJS 범위 지정 (Browserify / WebPack을 사용하는 브라우저 전용 코드에 사용).
- `shared-node-browser` - Node.js와 브라우저에 공통 인 전역
- `es6ecmaVersion` - 모듈을 제외한 모든 ECMAScript 6 기능을 활성화합니다 (이는 자동으로 파서 옵션을 6 으로 설정합니다).
- `es2017ecmaVersion` - 모든 ECMAScript 2017 전역을 추가하고 파서 옵션을 8로 자동 설정합니다 .
- `es2020ecmaVersion` - 모든 ECMAScript 2020 전역을 추가하고 파서 옵션을 11로 자동 설정합니다 .
- `worker` - 웹 워커 글로벌 변수.
- `amdrequire()define()` - amd 사양에 따라 전역 변수를 정의하고 정의합니다 .
- `mocha` - 모카 테스트 전역 변수를 모두 추가합니다.
- `jasmine` - 버전 1.3 및 2.0에 대한 모든 Jasmine 테스트 전역 변수를 추가합니다.
- `jest` - Jest 전역 변수.
- `phantomjs` - PhantomJS 글로벌 변수.
- `protractor` - protractor 전역 변수.
- `qunit` - QUnit 글로벌 변수.
- `jquery` - jQuery 전역 변수.
- `prototypejs` - Prototype.js 전역 변수.
- `shelljs` - ShellJS 글로벌 변수.
- `meteor` - meteor 글로벌 변수.

- `mongo` - MongoDB 전역 변수.
- `applescript` - AppleScript 전역 변수.
- `nashorn` - Java 8 Nashorn 글로벌 변수.
- `serviceworker` - 서비스 워커 글로벌 변수.
- `atomtest` - atomtest 도우미 글로벌.
- `embertest` - Ember 테스트 도우미 글로벌.
- `webextensions` - WebExtensions 글로벌.
- `greasemonkey` - greasemonkey 글로벌.