

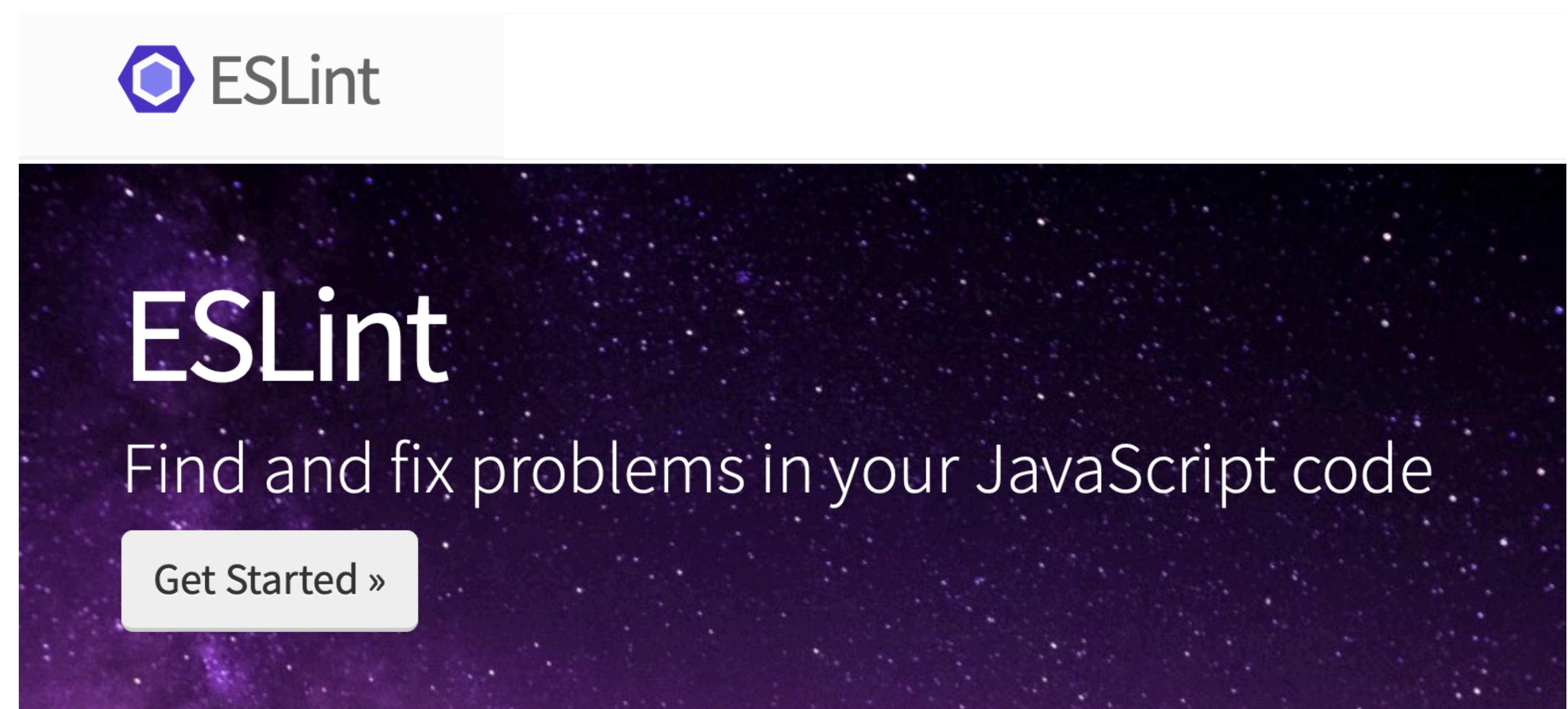
# Code Quality Tools



Full Stack Web Development

# Static Analysis

- Static program analysis is the analysis of computer software performed without executing any programs
- The term is usually applied to analysis performed by an automated tool.
- The Static Analysis may enforce a set of configurable Code Guidelines, which encapsulate a set of best practice rules



## Find Problems

ESLint statically analyzes your code to quickly find problems. ESLint is built into most text editors and you can run ESLint as part of your continuous integration pipeline.

## Fix Automatically

Many problems ESLint finds can be automatically fixed. ESLint fixes are syntax-aware so you won't experience errors introduced by traditional find-and-replace algorithms.

## Customize

Preprocess code, use custom parsers, and write your own rules that work alongside ESLint's built-in rules. You can customize ESLint to work exactly the way you need it for your project.

# AirBnB Code Style

- AirBnb defined a set of guidelines for use in their applications
- The rationale, documentation and ‘reasonable’ nature of the guidelines has made it a popular standard to aim for.
- Combine ESLint, Airbnb rules + IDE plugins



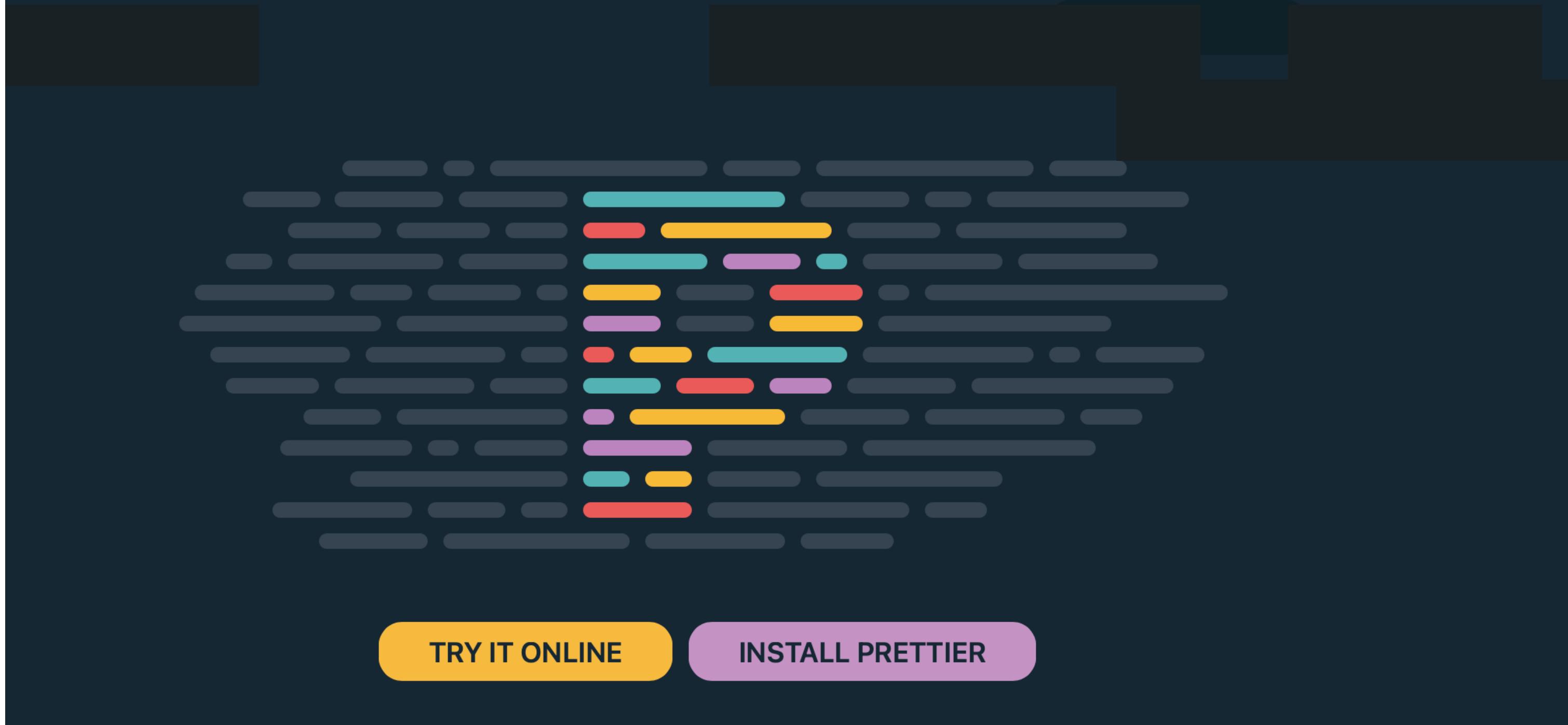
A screenshot of a code editor interface showing a file with the following code:

```
6  const __filename = fileURLToPath(import.meta.url);
7  const __dirname = path.dirname(__filename);
8
9  let test = 1;
10
11  async function() {
12    const port: 3000,
13      host: "localhost",
14    );
15
16  await server.start();
```

The code editor highlights the variable 'test' with a red underline, indicating it is never reassigned. A tooltip from ESLint suggests using 'const' instead of 'let'. The tooltip also includes options to 'Remove unused local variable' and 'More actions...'. A large blue arrow points from the bottom left towards the code editor.

# Code Formatting

- Prettier a prominent formatter in the Javascript Ecosystem
- “Opinionated” => formatter makes majority of key decisions.
- Some customisations - but far less than typical formatters



The background features a dark blue header with a decorative pattern of horizontal bars in various colors (teal, red, yellow, purple) on the right side. Below the header is a dark grey section containing two buttons: a yellow one labeled "TRY IT ONLINE" and a purple one labeled "INSTALL PRETTIER".

## # What is Prettier?

- \* An opinionated code formatter
- \* Supports many languages
- \* Integrates with most editors
- \* Has few options

## # Why?

- \* You press save and code is formatted
- \* No need to discuss style in code review
- \* Saves you time and energy
- \* And more

# Configure Code Quality Tools

## .prettierrc.json

```
{  
  "trailingComma": "es5",  
  "tabWidth": 2,  
  "semi": true,  
  "singleQuote": false,  
  "printWidth": 180  
}
```

## .eslintrc.json

```
{  
  "env": {  
    "node": true,  
    "es2021": true  
  },  
  "extends": [  
    "airbnb-base",  
    "prettier"  
  ],  
  "parserOptions": {  
    "ecmaVersion": "latest",  
    "sourceType": "module"  
  },  
  "rules": {  
    "quotes": [  
      "error",  
      "double"  
    ],  
    "import/extensions": "off",  
    "import/prefer-default-export": "off",  
    "object-shorthand": "off",  
    "no-unused-vars": "off",  
    "no-underscore-dangle": "off",  
    "no-param-reassign": "off",  
    "no-undef": "off",  
    "func-names": "off",  
    "no-console": "off"  
  }  
}
```

# Code Quality Tools

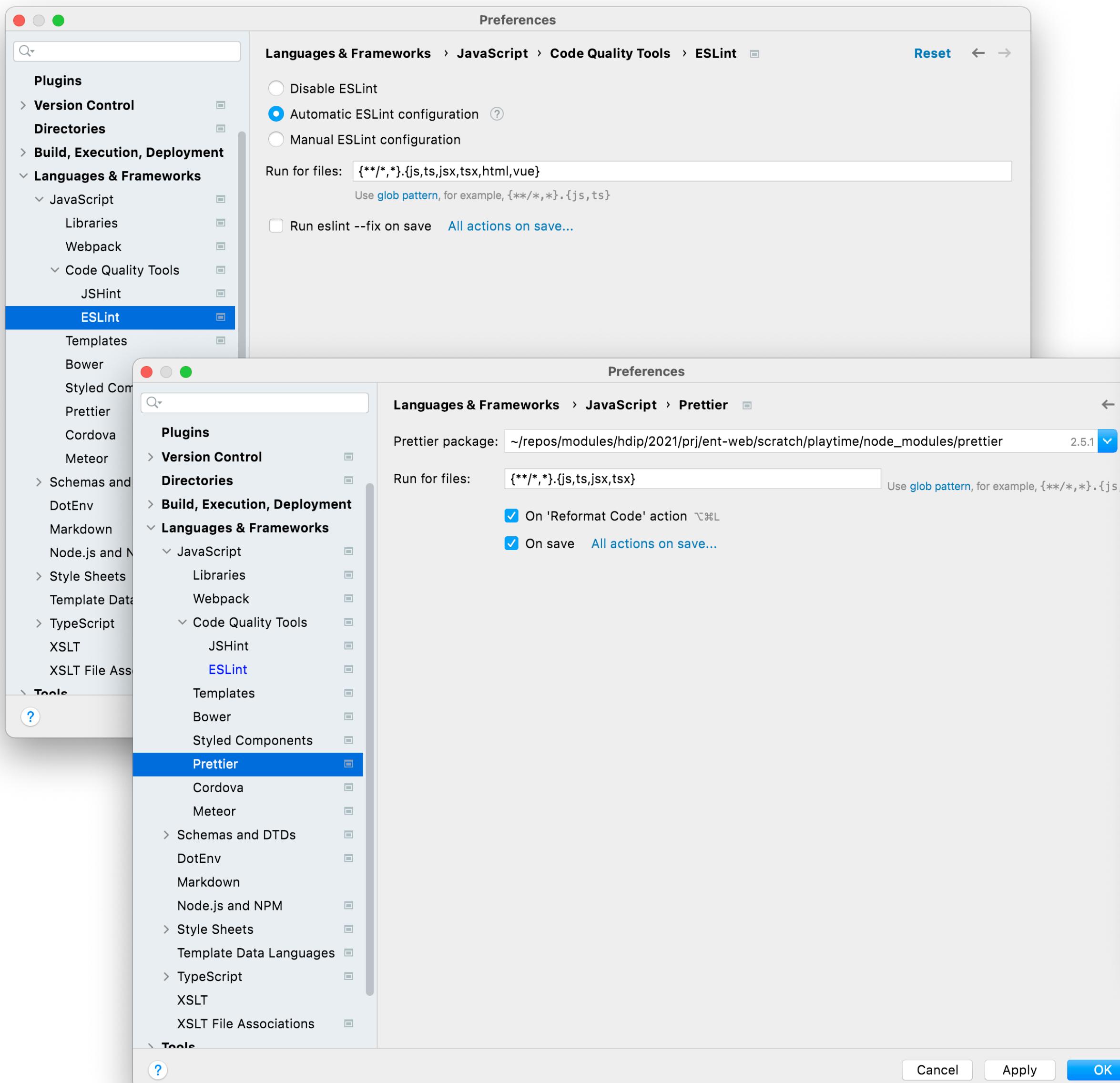
```
npm install -D eslint
npm install -D eslint-config-airbnb-base
npm install -D eslint-config-prettier
npm install -D eslint-plugin-import
npm install -D prettier
```

- ESLint
- Prettier
- Airbnb Code Style

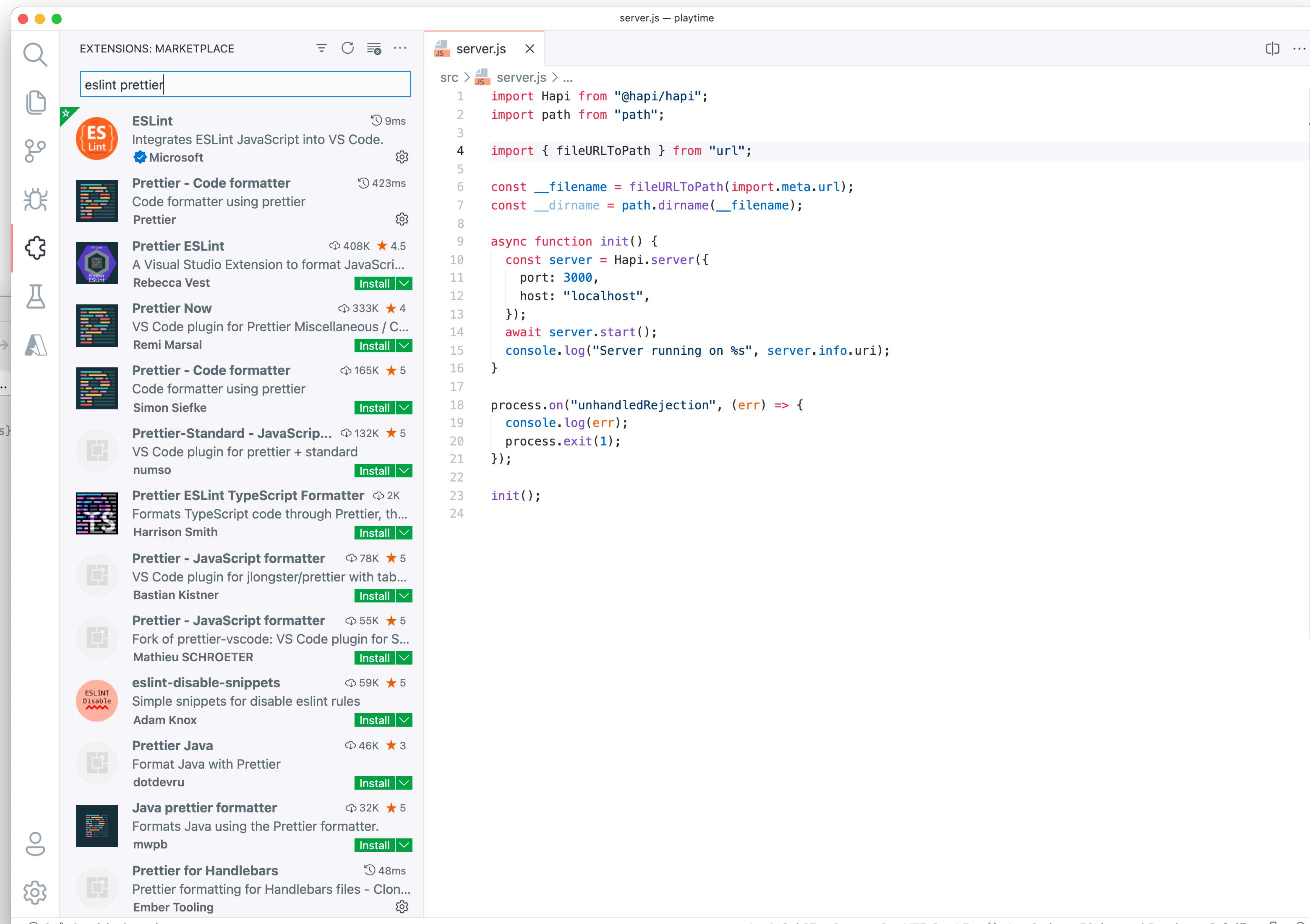
## package.json

```
{
  "name": "playtime",
  "version": "0.1.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "eslint": "^8.4.1",
    "eslint-config-airbnb-base": "^15.0.0",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-import": "^2.25.3",
    "prettier": "^2.5.1"
  }
}
```

# Code Quality IDE Configurations

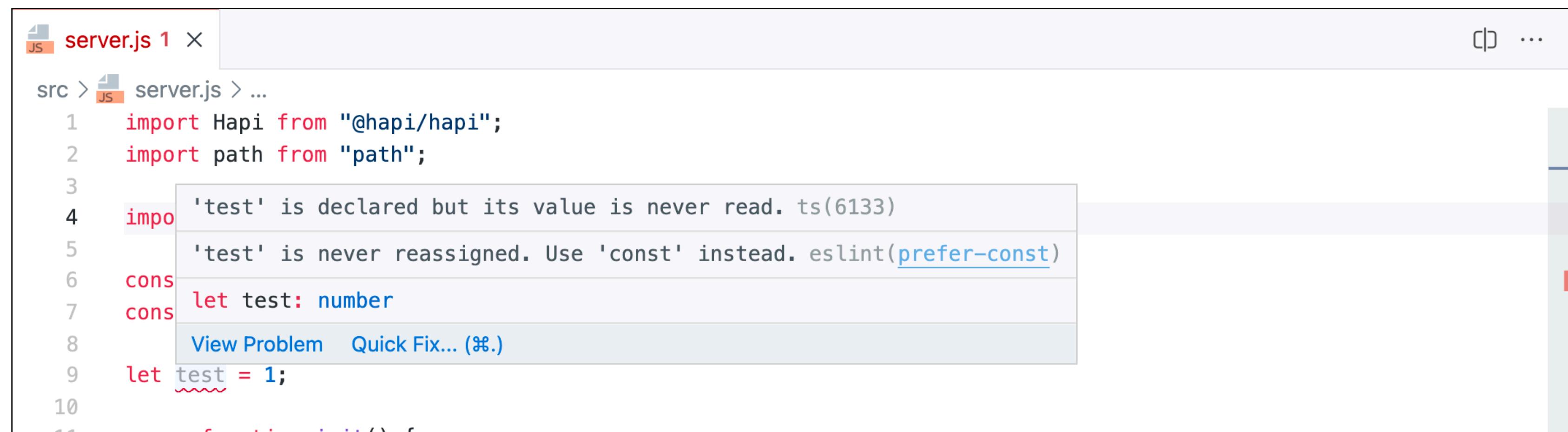
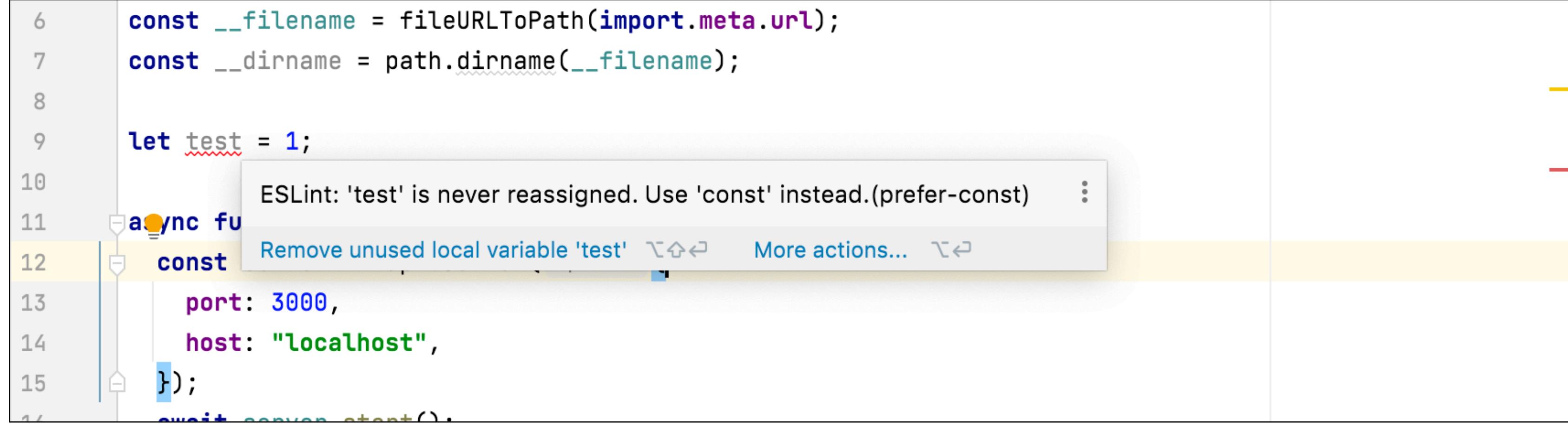


# WebStorm



# VSCODE

```
"rules": {  
  "import/extensions": "off",  
  "import/prefer-default-export": "off",  
  "object-shorthand": "off",  
  "no-unused-vars": "off",  
  "no-underscore-dangle": "off",  
  "no-param-reassign": "off",  
  "no-undef": "off",  
  "func-names": "off",  
  "no-console": "off"  
}
```



```
npm run lint
```

```
> playtime@0.1.0 lint
> ./node_modules/.bin/eslint . --ext .js

/Users/edeleastar/repos/modules/hdip/2021/prj/ent-web/scratch/playtime/src/server.js
  9:5  error  'test' is never reassigned. Use 'const' instead  prefer-const

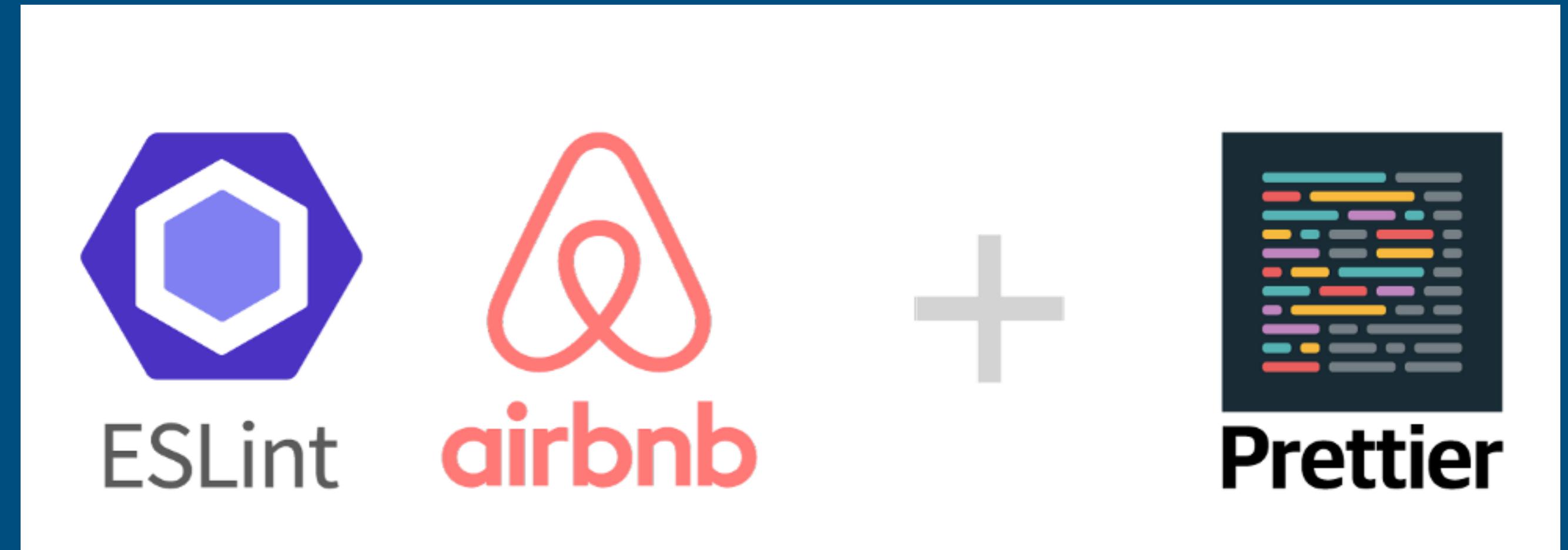
✖ 1 problem (1 error, 0 warnings)
  1 error and 0 warnings potentially fixable with the `--fix` option.
```

The screenshot shows a code editor window with a file named `server.js`. The file content is as follows:

```
src > JS server.js > ...
1 import Hapi from "@hapi/hapi";
2 import path from "path";
3
4 import 'test' is declared but its value is never read. ts(6133)
5 'test' is never reassigned. Use 'const' instead. eslint(prefer-const)
6 const test: number
7 const
8 let test = 1;
9
10
11
```

An ESLint error message is displayed in a tooltip over the line `5 'test' is never reassigned. Use 'const' instead. eslint(prefer-const)`. The message states: `'test' is never reassigned. Use 'const' instead. eslint(prefer-const)`. The editor interface includes tabs for `server.js` and `...`, and a status bar at the bottom.

# Code Quality Tools



Full Stack Web Development