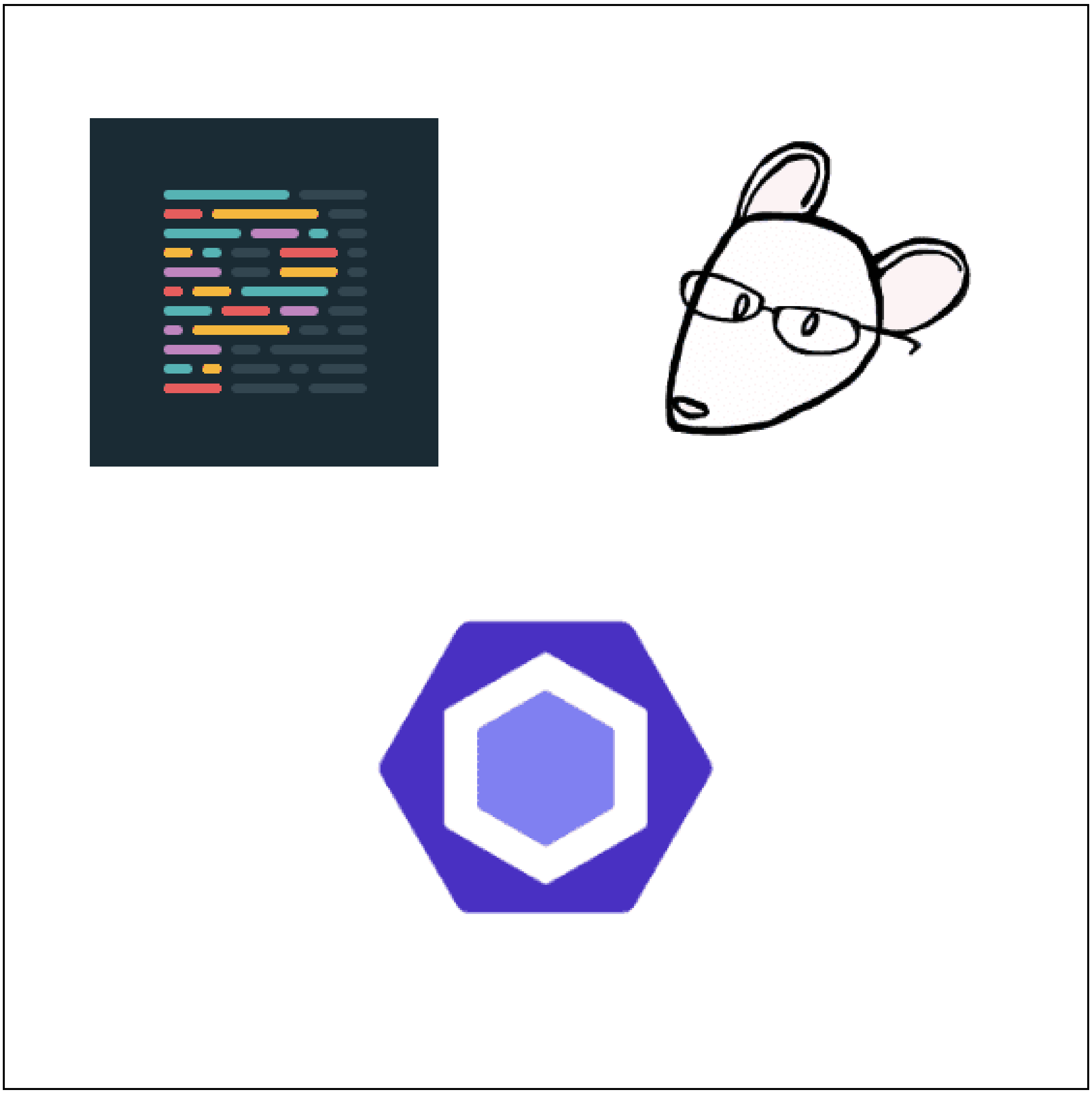
Set up ESlint, Prettier & EditorConfig without conflicts

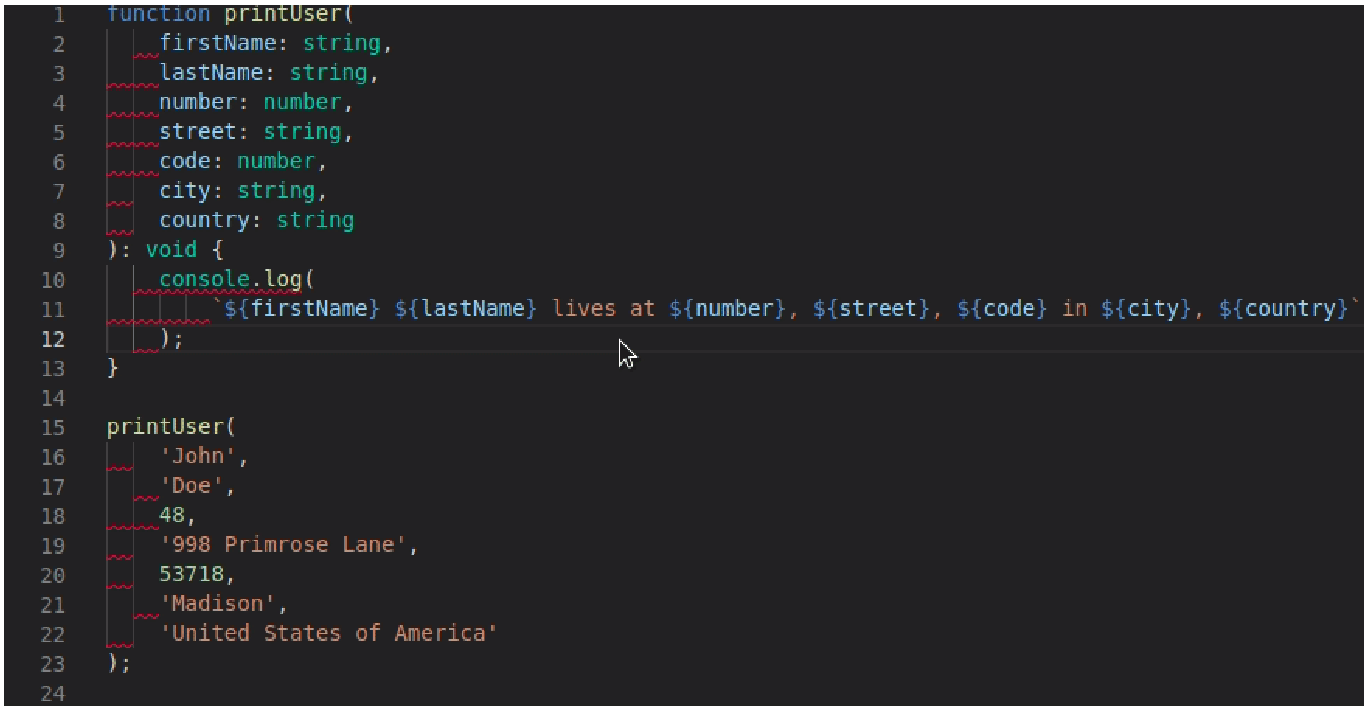
August 21, 2019 François Hendriks 12 min read



This is the second post of a series of articles on how to **empower your dev environment using ESLint, Prettier and EditorConfig**. If you have doubts regarding the use of all three tools, you may read my first article configuration.

Context

If you have already used Prettier alongside ESLint, you probably encountered **problems with your code formatting**.


```
function printUser(
    firstName: string,
    lastName: string,
    number: number,
    street: string,
    code: number,
    city: string,
    country: string
): void {
    console.log(
        `${firstName} ${lastName} lives at ${number}, ${street}, ${code} in ${city}, ${country}`
    );
}

printUser(
    'John',
    'Doe',
    48,
    '998 Primrose Lane',
    53718,
    'Madison',
    'United States of America'
);
```

I encountered these issues at work when we decided to migrate from TSLint to ESLint on one of our TypeScript projects. The idea was to upgrade our linting capabilities by using both ESLint and Prettier. We patched the issue above by adding an ESLint rule forcing two space indentations. However, other conflicts appeared and it became clear we could not patch them all by adding an ESLint rule for each conflict.

While a lot of online resources on this topic exist out there, most just give out a configuration that works for very specific projects and fail to give **an in depth explanation of why ESLint, Prettier or EditorConfig can cause formatting problems**. The objective here is to understand the

The pattern

We want to use **the right tool for the right issue** as discussed in the previous article on **Why You Should Use ESLint, Prettier & EditorConfig.** In this particular case, **ESLint** will be our `code quality` linter, **Prettier** our `code formatting` tool while **EditorConfig** will provide everyone with the right editor configuration.

- All configuration related to the editor (end of line, indent style, indent size…) should be handled by EditorConfig
- Everything related to `code formatting` should be handled by Prettier
- The rest (`code quality`) should be handled by ESLint

ESLint and Prettier

For the rest of this post, we will use the code already written in the first article. To sum it up, we had a `main.js` file alongside an ESLint configuration with Prettier installed.

main.js

```
function printUser(firstName, lastName, number, street,
    console.log(`${firstName} ${lastName} lives at ${num
}
printUser('John', 'Doe', 48, '998 Primrose Lane', 53718
```

eslintrc.json

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

Let Prettier do its job

In order to be able to use **ESLint alongside Prettier**, the idea is to **deactivate all ESLint rules that might conflict with Prettier** (`code formatting` rules). Fortunately for us, the `eslint-config-prettier` package already does that.

```
npm install eslint-config-prettier --save-dev
```

We will rewrite our `.eslintrc.json` file by adding `prettier` to the `extends` array and removing any `code formatting` rules we had:

```
{
  "extends": ["eslint:recommended", "prettier"],
  "env": {
    "es6": true,
    "node": true
  }
}
```

The `prettier` configuration will override any prior configuration in the extends array disabling all ESLint `code formatting` rules. With this configuration, Prettier and ESLint can be run separately without any issues.

The process of having to run two commands to lint and format our file is not very convenient. To fix this, **we will integrate Prettier with ESLint** by adding the `eslint-plugin-prettier` package.

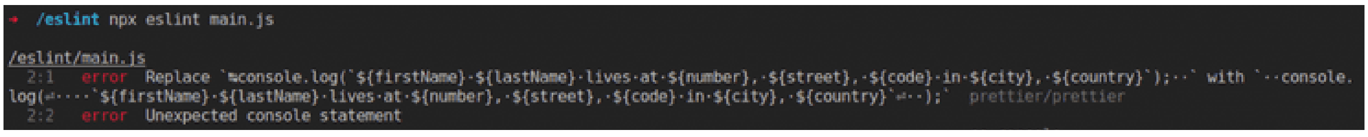
```
npm install eslint-plugin-prettier --save-dev
```

We will now rewrite our `.eslintrc.json` file by adding the prettier plugin in the `plugins` array and setting the newly established `prettier` rule to `error` so that any prettier formatting error is considered as an ESLint error.

```
{
  "extends": ["eslint:recommended", "prettier"],
  "env": {
    "es6": true,
    "node": true
  },
  "rules": {
    "prettier/prettier": "error"
  },
  "plugins": [
    "prettier"
  ]
}
```

We take our unlinted `main.js` file and apply ESLint on it:

```
npx eslint main.js
```



```
→ /eslint npx eslint main.js

/eslint/main.js
  2:1  error  Replace `⏎console.log(`${firstName}·${lastName}·lives·at·${number},·${street},·${code}·in·${city},·${country}`);··` with `··console.
log(⏎····`${firstName}·${lastName}·lives·at·${number},·${street},·${code}·in·${city},·${country}`⏎··);`  prettier/prettier
  2:2  error  Unexpected console statement
```

We can see that the lines that have too many characters or bad indentation, are marked by `prettier/prettier` as errors within our ESLint error output.

Now let's use the fix option of ESLint:

```
npx eslint --fix main.js
```

You will see that our file gets formatted the same way Prettier did.

You can replace all the prettier relevant configuration we made in our `.eslintrc.json` file by adding the `plugin:prettier/recommended` configuration in the extends array.

Here is what it would look like:

`.eslintrc.json`

```
{
  "extends": ["eslint:recommended", "plugin:prettier/re
  "env": {
    "es6": true,
    "node": true
  }
}
```

This configuration is the same as before but shorter and less explicit. For the sake of clarity, we will keep the first configuration for the rest of this post.

Adding an ESLint plugin

This setup is **all nice and good with this kind of small project** but eventually, when one starts to use Vue, React or other frameworks, **it becomes very easy to mess with both the ESLint and Prettier configurations**. A common problem that I encounter a lot happens when developers add an ESLint plugin and expect it to work without adding anything else to make Prettier work.

Example with TypeScript

At some point in the project above, we decided to use TypeScript. **Since TSLint will soon be deprecated**, we decided to replace it with ESLint. We are using TypeScript as an example. **What we do below is applicable to React, Vue or any other frameworks that have an ESLint plugin available**.

Here is our `main.ts` file inspired from our last `main.js` file:

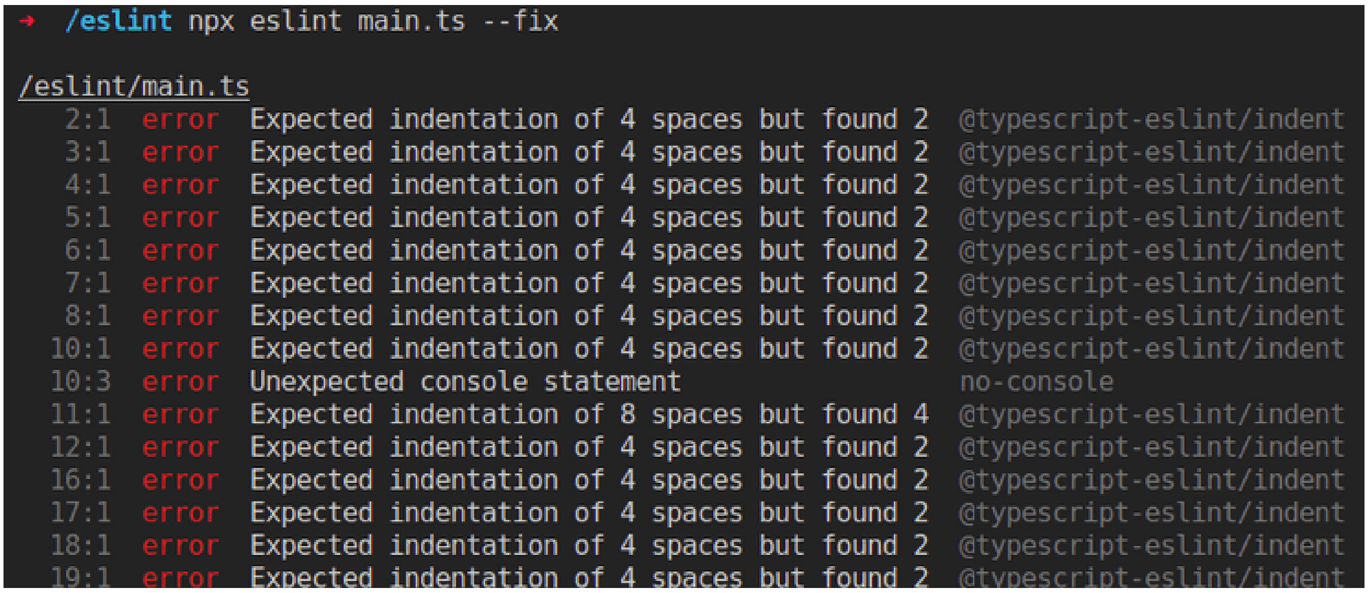
```
function printUser(firstName: string, lastName: string,
  console.log(`${firstName} ${lastName} lives at ${number
}
printUser('John', 'Doe', 48, '998 Primrose Lane', 53718
```

To make ESLint compatible with TypeScript or any other framework with specific syntax, we need to add a parser so that ESLint can read the new code and a new set of rules relevant to TypeScript (or any another framework that requires a parser) within a plugin.

We then modify our `.eslintrc.json` file to incorporate TypeScript by adding the TypeScript parser in the `parser` option and by adding the plugin to our `extends` array:

```
{
  "parser": "@typescript-eslint/parser",
  "extends": ["plugin:@typescript-eslint/recommended",
  "env": {
    "es6": true,
    "node": true
  },
  "rules": {
    "prettier/prettier": "error"
  },
  "plugins": [
    "prettier"
  ]
}
```

We then try ESLint on our file with the fix option:

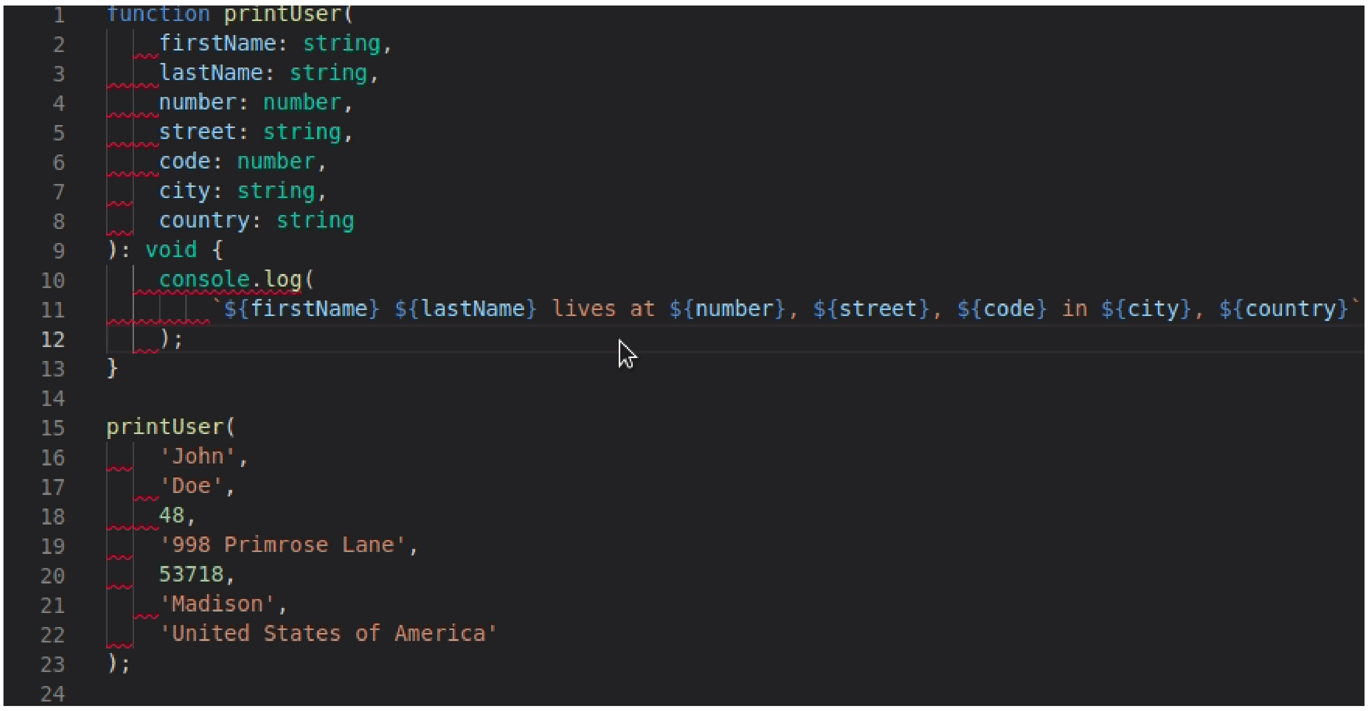


```
➜  /eslint npx eslint main.ts --fix

/eslint/main.ts
   2:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
   3:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
   4:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
   5:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
   6:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
   7:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
   8:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  10:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  10:3  error  Unexpected console statement                  no-console
  11:1  error  Expected indentation of 8 spaces but found 4  @typescript-eslint/indent
  12:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  16:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  17:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  18:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  19:1  error  Expected indentation of 4 spaces but found 2  @typescript-eslint/indent
  17 errors and 0 warnings potentially fixable with the `--fix` option.
```

And if we run the command multiple times, we get the same error even though the console says the errors can be fixed. The file does change and becomes:

```
function printUser(
  firstName: string,
  lastName: string,
  number: number,
  street: string,
  code: number,
  city: string,
  country: string
): void {
  console.log(
    `${firstName} ${lastName} lives at ${number}, ${stre
  );
}

printUser(
  'John',
  'Doe',
  48,
  '998 Primrose Lane',
  53718,
  'Madison',
  'United States of America'
);
```

If like me, you use VSCode and have the auto-fix on save option for ESLint on, the issue will look like this:



```
function printUser(
    firstName: string,
    lastName: string,
    number: number,
    street: string,
    code: number,
    city: string,
    country: string
): void {
    console.log(
        `${firstName} ${lastName} lives at ${number}, ${street}, ${code} in ${city}, ${country}`
    );
}

printUser(
    'John',
    'Doe',
    48,
    '998 Primrose Lane',
    53718,
    'Madison',
    'United States of America'
);
```

Fixing the conflict by disabling all ESLint formatting rules of the added plugin

Now, a common mistake a lot of people make is try to fix this issue or any other issue regarding conflicts with Prettier after the addition of a new plugin by overriding the rule that generates the conflict. In this case it is the indent rule of the `@typescript-eslint` plugin. To do this, they add this rule to the `rules` array:

```
"@typescript-eslint/indent": ["error", 2]
```

This effectively solves the problem by forcing the `typescript-eslint` indent rule to match the indentation rule of prettier. **But two problems appear**:

- Now **both ESLint and Prettier will be responsible for formatting our code**. This is not the pattern we want to follow

If we follow the pattern described above, no `code formatting` should be done by ESLint and the newly added plugin is no exception to this pattern. **We therefore need to disable all the `code formatting` rules of the added plugin** by adding `prettier/@typescript-eslint` to our `extends` array.

```
{
  "parser": "@typescript-eslint/parser",
  "extends": ["plugin:@typescript-eslint/recommended",
  "env": {
    "es6": true,
    "node": true
  },
  "rules": {
    "prettier/prettier": "error"
  },
  "plugins": [
    "prettier"
  ]
}
```

Please keep in mind that the `.eslintrc.json` `extends` array's order is very important. Basically each time a new configuration is added to the array, it will override the previous configurations. It is therefore of the utmost importance that `prettier` and `prettier/@typescript-eslint` are at the end of the array.

To sum up this common problem:

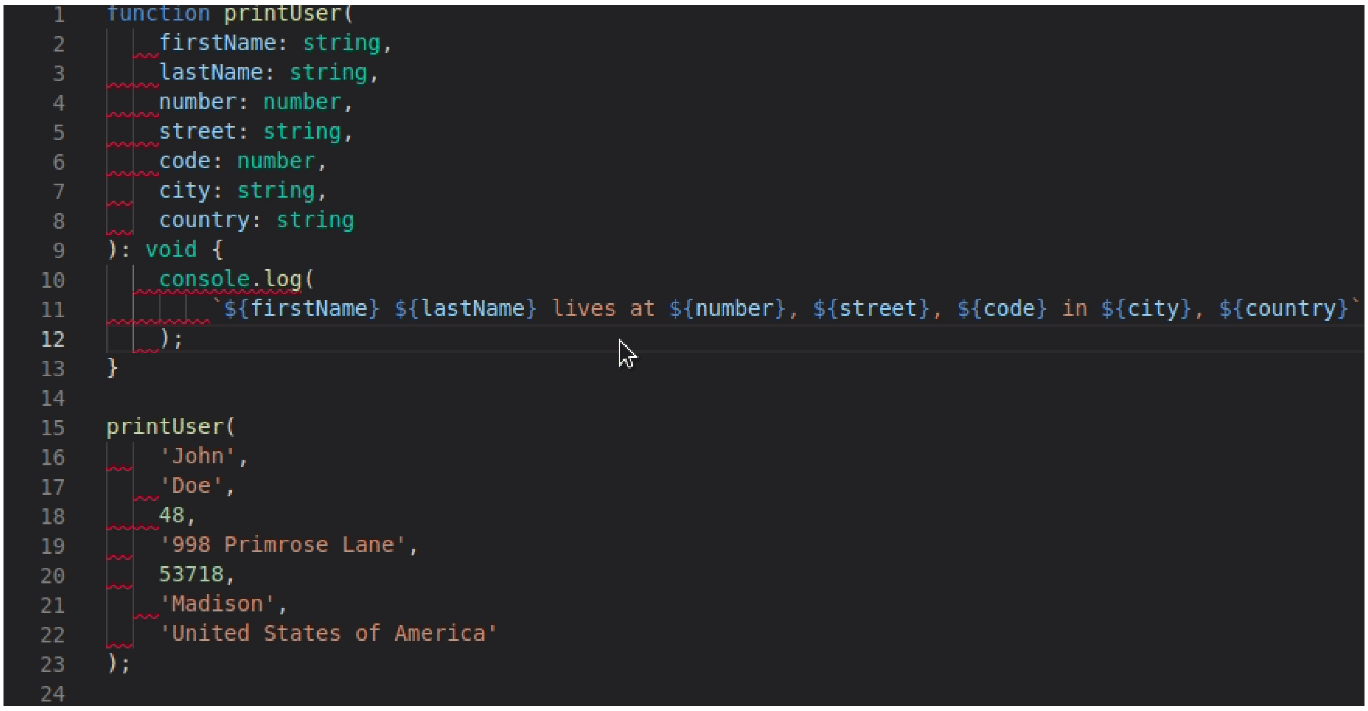
- Whenever you add a plugin in ESLint, **you have to think about the `code formatting` rules that it adds** and add a `prettier` configuration (if available) to disable them all. In our case, we used `prettier/@typescript-eslint` but we could have used `prettier/react` or `prettier/vue`. Check out the **eslint-config-prettier documentation** to get the list of supported ESLint plugins.
- **Do not try overriding the formatting rules by yourself** in the `.eslintrc.json` file (it is not ESLint's role)
- If you see your code being formatted in two different ways with Prettier and ESLint conflicting, it means you have a useless ESLint formatting rule generating this conflict and that you do not follow the pattern described above

Adding a custom rule

For our project above, the team is not comfortable with 2 space indentation and wants to switch to 4. **A common mistake is to forget our ESLint-Prettier pattern and apply this rule within the `.eslintrc.json` file** like so:

```
{
  "parser": "@typescript-eslint/parser",
  "extends": ["plugin:@typescript-eslint/recommended",
  "env": {
    "es6": true,
    "node": true
  },
  "rules": {
    "plugins": [
      "prettier"
    ]
  }
```

The generated error should be of no surprise to you if you followed everything that was said before. We have the exact same error as in the previous issue with a conflict between Prettier and ESLint:



```
function printUser(
    firstName: string,
    lastName: string,
    number: number,
    street: string,
    code: number,
    city: string,
    country: string
): void {
    console.log(
        `${firstName} ${lastName} lives at ${number}, ${street}, ${code} in ${city}, ${country}`
    );
}

printUser(
    'John',
    'Doe',
    48,
    '998 Primrose Lane',
    53718,
    'Madison',
    'United States of America'
);
```

There are two problems:

- Since our custom rule is defined within our own `rules` array of the `.eslintrc.json` file, it overrides the `prettier/@typescript-eslint` configuration that is responsible for disabling all formatting rules
- Once again, by adding this rule, we just forget about Prettier and do not follow our pattern

`.prettierrc`

```json
{
  "tabWidth": 4
}
```

Now Prettier will format your code using 4 spaces instead of the default 2 while the `.eslintrc.json` file should be exempt of any rules regarding indentation.

To sum up this common issue:

- Whenever you want to add a rule, **try and categorize the rule within the `code quality` or `code formatting` sets**. To make this easier, you just have to check whether the rule is supported and enforcable by prettier
- **Do not add custom formatting rules** to your `.eslintrc.json` file. These will almost certainly conflict with Prettier

Prettier and EditorConfig

Setting up the editor configuration

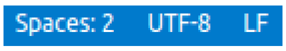
EditorConfig allows us to have the same editor configuration regardless of the editor used. We thus do not have to rely on Prettier to format our code with the team's conventions each time new code is written. It does require you to have the necessary **EditorConfig** plugin or extension installed on your IDE.

We add a custom editor configuration to our project:

`.editorconfig`

```
[*]
end_of_line = lf
charset = utf-8
indent_style = space
```

If you use **the EditorConfig VSCode extension** with this configuration, the editor will automatically know how to format your files. You will notice this at the bottom right of the editor:


Spaces: 2 UTF-8 LF

Avoiding redundant configuration between EditorConfig and Prettier

However, this means that **Prettier and EditorConfig share some configuration options that we do not want to repeat in two separate configuration files and keep them in sync** (e.g. end of line configuration). The latest versions of **Prettier** address this issue by parsing the `.editorconfig` file to determine what configuration options to use.

Those options are limited to:

```
end_of_line
indent_style
indent_size/tab_width
max_line_length
```

```
"endOfLine"
"useTabs"
"tabWidth"
"printWidth"
```

As it was the case for ESLint and Prettier, if you wish to change the configuration, **the rule is to check whether it is a EditorConfig or Prettier relevant configuration and then change it in the appropriate file**. The previous configuration options should be written only in the `.editorconfig`.

If you followed what we have done until now, you might notice a configuration error. We changed the indentation size in our Prettier configuration. Since this is an editor relevant configuration, we should add it to the `.editorconfig`file.

`.editorConfig`

```
tab_width 4
```

We remove the `.prettierrc.json` file and run ESLint on our unformatted code:

```
function printUser(
    firstName: string,
    lastName: string,
    number: number,
    street: string,
    code: number,
    city: string,
        `${firstName} ${lastName} lives at ${number}, ${
    );
}

printUser(
    "John",
    "Doe",
    48,
    "998 Primrose Lane",
    53718,
    "Madison",
    "United States of America"
);
```

The code's indentation is at 4. Now, whenever you open a new file with your editor, the indentation will already be configured at 4. Prettier will not have to format your code for this setting.

I hope this article will help you configure ESLint, Prettier and EditorConfig on your own projects. It took me some time to figure it all out. With this type of configuration, **I think it would be best not to adopt a setup once and forget approach for your linting and formatting experience**. If you wish to modify your configuration, you will have to remember following the pattern described above.

François Hendriks

Developer @Theodo. Fan of VueJS and TypeScript!

← Flutter Tutorial - Custom User Input Knob Using GestureDetector

Theodo

Buy stocks without paying commission with eToro
eToro

Elektrisk XC40. For en fast månedspris. Abonner nå på nett.
Care by Volvo

Sarpsborg: De fleste nordmenn kjenner ikke til disse lure triksene for å spare penger
Eco Experts

Kaldt hjemme? Gjør et vinterkupp på varmepumpe idag!
Tjenestetorget

Strøm (billigste leverandør)
Strøm.no

These Twins Were Named "Most Beautiful In The World," Wait Till You See Them Today
LuxuryLevels.com

Varmepumpe inkludert montering: Få 3 tilbud til ditt hus

8 Comments **Theodo** 1 **Login**

Recommend 3 Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

mohammed • 15 days ago
eslint padded block conflict with prettier and I did not find anyway to fix that !

Theodo

Rishabh Sharma • 3 months ago

Wow, this really helped a ton! Thanks!!

Reply • Share ›

Ali Saeed • 7 months ago

Thank you so much. Extremely helpful. Thanks for the detailed explanation and sharing your awesome separation of concerns model where formatting is done by prettier and quality checks by eslint.

Reply • Share ›

Mauro Oliveira • 9 months ago

Thanks a lot !
Very detailed explanation.
It helped me a lot!

Reply • Share ›

Tobias Kraus • 9 months ago

This was s helpful! I don't really like that there are so many libraries, plugins and IDE extensions needed to get the best result (especially in my project with React & TypeScript), but with this article I understand at least what's going on. I read many articles about this topics - but this one is the one I was looking for.

Would be great though if ESLint could cover in future everything what prettier does. Would simplify things a lot.

Reply • Share ›

Sean G. Wright • a year ago

I really appreciate the in depth explanation of how each tool/package interacts with the others. This kind of explanation is a lot more helpful than copying and pasting something into my prettier or eslint files and hoping it works!

Thanks!

Reply • Share ›

François Hendriks ➜ Sean G. Wright • a year ago

Thank you for you comment! This is exactly what I was trying to do! Too many times have I just copied and pasted configurations without actually knowing what

Theodo

Elektrisk XC40. For en fast månedspris. Abonner nå på nett.

Care by Volvo

Solcellepaneler - Skaff deg nå før subsidiene forsvinner!

Solkunder

Sarpsborg: De fleste nordmenn kjenner ikke til disse lure triksene for å spare penger

Eco Experts

Kaldt hjemme? Gjør et vinterkupp på varmepumpe idag!

Tjenestetorget

Strøm (billigste leverandør)

Strøm.no