Configuring linting and adding autoformatting

We are going to take our Create React App project to the next level in this lesson by fully setting up linting and also adding a tool that will autoformat the code.

WE'LL COVER THE FOLLOWING

Re-open the project in the correct folder

Before we start, we need to open the project we created in the last lesson in the root folder. The root folder should have the same name as your app and should contain the `package.json` file.

Linting

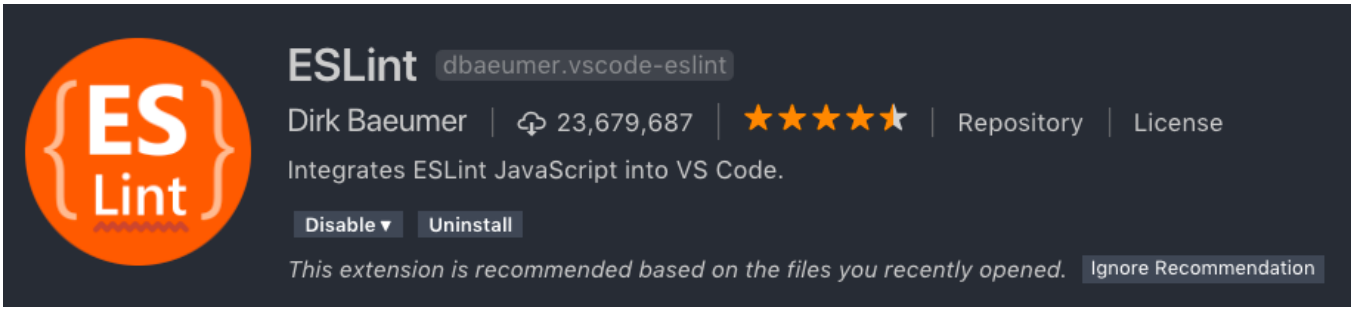
Linting is the process of running a program that checks for potential errors. The most popular linter in the JavaScript space at the moment is **ESLint**. CRA has already installed and configured ESLint to check our React and TypeScript code - neat!

> Note that **TSLint** has been a popular alternative to ESLint for linting

> TypeScript code but is now deprecated. More information can be found here.

Configuring ESLint in Visual Studio Code

We can install and configure an ESLint extension in Visual Studio Code to lint and highlight potential errors. The extension is called **ESLint** and is published by Dirk Baeumer.



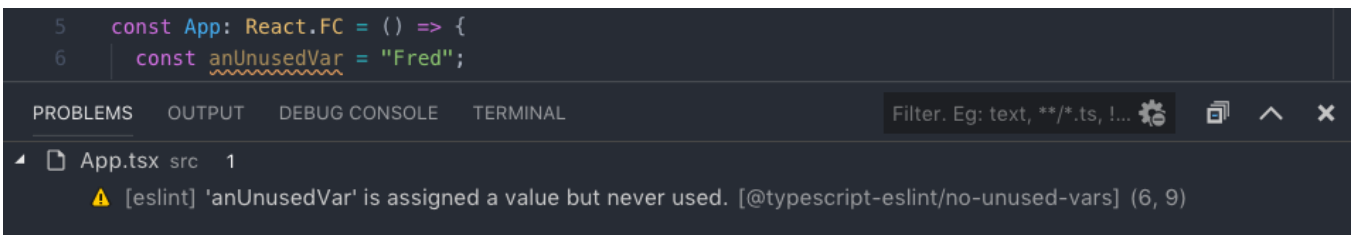
Note that you may need to start Visual Studio Code for the extension to start working.

We need to tell VS Code to lint TypeScript code:

- First, create a folder called `.vscode` in the root of our project.
- Then create a file called `settings.json` in the `.vscode` folder with the following content:

```
{
  "eslint.validate": [
    "javascript",
    "javascriptreact",
    { "language": "typescript", "autoFix": true },
    { "language": "typescriptreact", "autoFix": true }
  ]
}
```

Linting errors will be underlined and appear in the *Problems* list:



Configuring ESLint rules

The rules that ESLint runs when it does its checks can be changed in a configuration file. There are several ESLint configuration file formats; we'll use a file called `.eslintrc.json` in the project root folder. We can add the file with the following content to inherit from the ESLint configuration set up by CRA:

```
{
  "extends": "react-app"
}
```

We can then add and override rules using a `rules` field:

```
{
  "extends": "react-app",
  "rules": {
    "no-debugger":"warn"
  }
}
```

After we save our updated `.eslintrc.json` file, the new rules will take effect on our project.

All the available ESLint rules can be found here.

In our project, add an ESLint rule to warn us when we use a `console.log` statement.

Show Answer

Adding automatic code formatting

Some of the linting rules that deal with the format of our code (such as semicolons at the end of statements) can be automatically dealt with by a tool like **Prettier**.

Installing Prettier

We can install Prettier into our project by executing the following npm

command in the Terminal:

```
npm install --save-dev prettier
```

Making Prettier play nice with ESLint

Prettier can take responsibility for the style rules off ESLint by using the `eslint-config-prettier` and `eslint-plugin-prettier` ESLint plugins.

- `eslint-config-prettier` disables ESLint rules that conflict with Prettier
- `eslint-plugin-prettier` is an ESLint rule that formats code using Prettier

These can be installed using the following npm command in the Terminal:

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

The ESLint prettier plugin and the prettier rule also need to be defined in `.eslintrc.json`:

```
{
  "extends": ["react-app","prettier"],
  "plugins": ["prettier"],
  "rules": {
    "prettier/prettier": "error",
    ...
  }
}
```

Configuring Prettier

There are several Prettier configuration file formats; we'll use a file called `.prettierrc`. Let's add a file called `.prettierrc` to the root of our project with the following content:

```
{
 "printWidth": 100,
 "singleQuote": true,
 "semi": true,
 "tabWidth": 2,
 "trailingComma": "all"
}
```

Here we have specified that:

- Lines should wrap at 100 characters.
- Single quotes should be used rather than double quotes.
- Semicolons should be placed at the end of statements.
- The indentation level will be two spaces.
- A trailing comma will be added to multi-line arrays and objects.

More information on the configuration options can be found here.

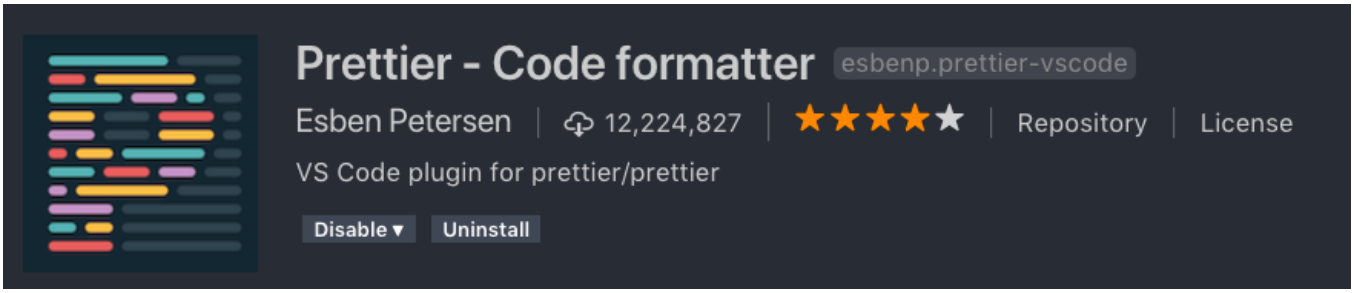
After we save our updated `.prettierrc` file, the new rules will take effect on our project.

As an exercise, change the number of indentation spaces to 4.

Show Answer

Configuring Prettier in Visual Studio Code

The most popular VS Code Prettier extension is **Prettier – Code formatter** and is published by Esben Petersen:



Note that you may need to start Visual Studio Code for the extension to start working.

We can get VS Code to automatically format code in a file when we save it by adding the `editor.formatOnSave` flag to our `settings.json` file:

```
{
  ...,
  "editor.formatOnSave": true
}
```

That’s it! Our code will now be nicely formatted when we save it.

Wrap up

The project feels great now; our code is being formatted nicely for us, and we have increased protection from making mistakes.

We learn how to run the tests in the project and produce a production-ready build in the next lesson.