ESLint

Haider Zia
19th April 2022

WHY SHOULD I USE A LINTER?

Javascript is notorious for being frustrating to debug. When your code doesn't compile and you receive a cryptic error message that you don't understand, you may spend hours finding and fixing a tiny, seemingly simple error.

Catching and fixing errors is a pain to beginwith, and it is only exacerbated by Javascript being a dynamically typed language because that increases the likelihood of errors. It is difficult for a developer to keep track of the type of a variable throughout the program, and that may introduce errors that would never arise in a statically typed language.

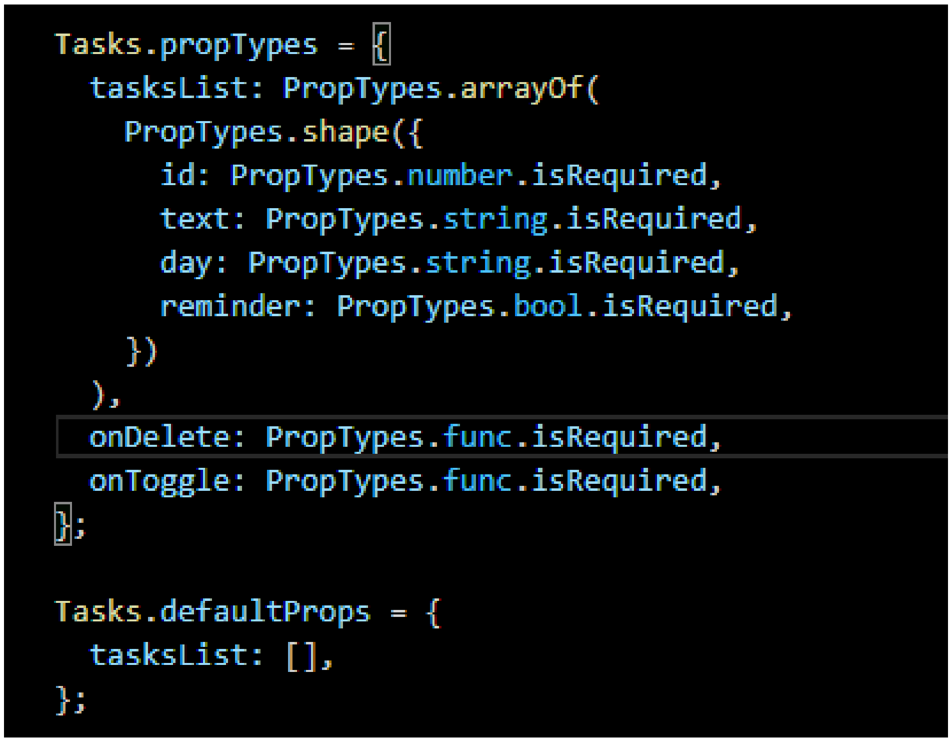
The problem, however, is not only the introduction of errors in the code, but also that these errors may be very elusive. Your code may compile perfectly at the time only for there to be a lot of bugs after it is deployed. That is a lot worse than running into errors during development.

Errors aren't the only problems in code, however. You may have perfectly running, error-free code, but it may be useless when working with other people if it is unreadable. This unreadability may significantly slow down a team when team members have to decipher each other's code before attempting to add to or modify it.

It is because of all of these problems that using a Linter is a good idea for Javascript developers. So how are these problems addressed?

Error Prevention

By strictly enforcing certain rules before the code is compiled, the likelihood of run-time errors is reduced considerably. An example of this from my own code is strictly enforcing Prop Type Validation, and Default Props for those props that are not required as ".isRequired" in React JS components. This prevents developers from running into type-related errors during run-time.



```
Tasks.propTypes = {
  tasksList: PropTypes.arrayOf(
    PropTypes.shape({
      id: PropTypes.number.isRequired,
      text: PropTypes.string.isRequired,
      day: PropTypes.string.isRequired,
      reminder: PropTypes.bool.isRequired,
    })
  ),
  onDelete: PropTypes.func.isRequired,
  onToggle: PropTypes.func.isRequired,
};

Tasks.defaultProps = {
  tasksList: [],
};
```

There are, of course, many other rules enforced such as no unused variables, requiring super() call inside constructors, and so on. In addition to that, warnings are also given for console logs so that such things may be removed before putting the code in production.

Readability and Synergy

In addition to preventing errors, having a linter enforce company-wide or team-wide code style consistency allows team members to easily read eachothers' code – at least in terms of style and conventions – like it is their own code. This reduces the time spent understanding eachothers' code and allows the team to work as one unit rather than multiple individuals, which in turn increases productivity.

For that purpose, linters, in conjunction with plugins like Prettier, can be used to automatically change the code to conform to style rules when the developer hits save. For example, you could use single quotes around some text and it would automatically convert to double quotes when you hit save.

WHY ESLINT?

These benefits may be a reason to use a linter, but let's discuss the reasons for using ESLint in particular in comparison to other common linters for JS

1. Unlike JSLint, ESLint gives you the ability to configure your rules. You can enable and disable rules that are shipped with ESLint per your need.
2. Unlike JS Lint's newer version JS Hint, ESLint gives you the ability to add your own custom rules.
3. Unlike JSCS, ESLint doesn't just detect code style violations, but also has rules against potential bugs.

ESLINT INSTALLATION AND CONFIGURATION

If all of this sounds like something that would be helpful for you or your team, here is how to install ESLint. For this example, we'll consider a React JS project in VSCode, and configure our project with Prettier and Airbnb style guide in addition to ESLint.

First, we navigate inside the project in the terminal. Then we install ESLint using the following command. Note that this command only installs it for your current project, not globally.

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

We then initialize it using the following command, which creates an ESLint configuration file inside your project

```
npm eslint --init
```

We will then be asked certain questions in the terminal about how we want to configure ESLint for our project. We can switch between these choices using arrow keys and select one by hitting enter. In our example, we choose the last option for the following question.

```
How would you like to use ESLint? …
  To check syntax only
  To check syntax and find problems
❯ To check syntax, find problems, and enforce code style
```

This will be followed by another question. We will choose JavaScript modules (imports/exports) from the available choices.

```
What type of modules does your project use? …
```

```
❯ JavaScript modules (import/export)

CommonJS (require/exports)

  None of these
```

For the next question, we will choose React JS.

```
Which framework does your project use? …

❯React

  Vue.js

  None of these
```

Our project does not use Typescript, so we will answer the next question accordingly.

```
Does your project use TypeScript? › No / Yes
```

We will make the following choice in the next question.

```
Where does your code run? …  (Press <space> to select, <a> to toggle all, <i> to invert
selection)

✔ Browser

  Node
```

The next question will include an option after choosing “Use a popular style guide” for the Airbnb styleguide, we’ll choose that from the available options.

```
✔ How would you like to define a style for your project? …

❯ Use a popular style guide

  Answer questions about your style
```

```
 Inspect your JavaScript file(s)
```

We'll choose JSON in the next question

```
What format do you want your config file to be in? …
  JavaScript
  YAML
❯ JSON
```

Finally, the required dependencies will be displayed on the terminal and we will be asked if we want to install them, in response to which we will choose yes.

```
Would you like to install them now with npm?  No / ›Yes
```

Once this is done, we can go ahead and install the VSCode plugins for ESLint and Prettier. After these installations are complete, we will return to the terminal and enter the following command.

```
npm i prettier eslint-config-prettier eslint-plugin-prettier -D
```

We will then go to the .eslintrc file and update the following section:

```
"extends": [ "airbnb", "plugin:prettier/recommended" ]
```

To allow ESLint to automatically correct your code according to the style guide when you hit save, we will navigate to VSCode settings using the command palette and choose: Preferences: Open Workspace Settings (JSON). Update the shown section as follows.

```
{
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
  "eslint.validate": ["javascript"]
}
```

By the end of this step, you’re ready to go!

WHAT NOW?

ESLint might throw a bunch of errors left and right before compilation for code that you would previously have considered fine, and that can be overwhelming (I know it was for me). Do not let that discourage you, because while all these new error messages may be some work to figure out at first, take satisfaction in knowing that the code you write now will be of higher quality and have lesser likelihood of having errors at run-time.