**ESLint, JSLint, TSLint:** These three tools work in the same basic way. They have a set of rules which they use to analyze and report problems in JavaScript files. A good linting tool can help make sure a project adheres to coding standard.

**JSLint:** is easy to use. The downside is JSLint is not configurable and extensible. You can’t add custom rules and it is difficult to tell which rule is causing which error

**TSLint:** can only be used for Typescript

**ESLint**: supports both JavaScript and Typescript. It is easy to understand. It is flexible and extensible. It has a good support for both ES6 and JSX. ESLint is already included as a dependency with create-react-app

Installing ESLint:

yarn add eslint –save-dev (preferred)

**Configuring ESLint:** Create a .eslintrc file at the root of your project. ESLint can be run on any file from this file.

OR

It is also possible to define ESLint configuration inside a package.json file. You will need your configuration object in a eslintConfig key.

**Setting up ESLint to work with Typescript:**

Install the 3 required dev dependencies.

yarn add eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin –dev

1. eslint: The core ESLint linting library
2. @typescript-eslint/parser: The parser that will allow ESLint to lint Typescript code
3. @typescript-eslint/eslint-plugin: A plugin that contains a bunch of ESLint rules that are Typescript specific

If using create-react-app to bootstrap a project, eslint is already included as a dependency through react-scripts and therefore not required to install it with yarn.

Add .eslintrc.js file in the root project directory. Here you will need to configure Typescript.

If using Typescript with React, the eslint-plugin-react dev dependency should be installed

**Adding Prettier to our project:** Prettier is used to handle code formatting. Install the following dependency to get prettier working with ESLint.

yarn add prettier eslint-config-prettier eslint-plugin-prettier –dev

1. Prettier: is the core prettier library.

2- Eslint-config-prettier: Disble ESLint rules which conflict with prettier

3- Eslint-plugin-prettier: Runs prettier as ESLint rule

To configure prettier .prettierrc.js file needs to be created at the root project directory

The basic .eslintrc with prettier enabled and the recommended plugin look like this:

{

    "root": true,

    "parser": "@typescript-eslint/parser",

    "plugins": [

      "@typescript-eslint",

      "prettier"

    ],

    "extends": [

      "eslint:recommended",

      "plugin:@typescript-eslint/eslint-recommended",

      "plugin:@typescript-eslint/recommended",

      "prettier"

    ],

    "rules": {

      "no-console": 1,       // Means warning

      "prettier/prettier": 2 // Means error

    }

**Formatting Code Automatically**: To format our code whenever we commit code in git we need to add the following dependencies:

yarn add –save husky lint-staged prettier

1. Husky: makes it possible to use githooks as if they are npm scripts
2. Lint-staged: allows us to run scripts on staged files in git

Add these line to package.json to make sure files are formatted correctly

"husky": {

  "hooks": {

  "pre-commit": "lint-staged"

  }

}

+ "lint-staged": {

  "src/\*\*/\*.{js,jsx,ts,tsx,json,css,scss,md}": [

  "prettier --write"

  ]

},

Whenever we make a commit Prettier will format the changed files automatically.

VSCode Extensions: Here are some useful VS Code extensions ESLint, Path Intellisense, Prettier

Prettier: Formatting shortcut shift + alt + f