JavaScript/TypeScript Code Reviews

Style Guide

Developers should use prettier to do code formatting for JavaScript.

Using an automated code formatting tool like Prettier enforces a well accepted style guide that was collaboratively built by a wide range of companies including Microsoft, Facebook, and AirBnB.

For higher level style guidance not covered by prettier, we follow the AirBnB Style Guide.

Code Analysis / Linting

eslint

Per guidance outlined in Palantir's 2019 TSLint road map, TypeScript code should be linted with ESLint. See the typescript-eslint documentation for more information around linting TypeScript code with ESLint.

To install and configure linting with ESLint, install the following packages as dev-dependencies:

```
npm install -D eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin
```

Add a `.eslintrc.js` to the root of your project:

```
module.exports = {
  root: true,
  parser: '@typescript-eslint/parser',
  plugins: [
    '@typescript-eslint',
  ],
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/eslint-recommended',
    'plugin:@typescript-eslint/recommended',
  ],
};
```

Add the following to the `scripts` of your `package.json`:

```
"scripts": {
    "lint": "eslint . --ext .js,.jsx,.ts,.tsx --ignore-path .gitignore"
}
```

This will lint all `.js`, `.jsx`, `.ts`, `.tsx` files in your project and omit any files or directories specified in your `.gitignore`.

You can run linting with:

```
npm run lint
```

Setting up Prettier

Prettier is an opinionated code formatter.

Getting started guide.

Install with `npm` as a dev-dependency:

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

Add `prettier` to your `.eslintrc.js`:

```
module.exports = {
  root: true,
  parser: '@typescript-eslint/parser',
  plugins: [
    '@typescript-eslint',
  ],
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/eslint-recommended',
    'plugin:@typescript-eslint/recommended',
    'prettier/@typescript-eslint',
    'plugin:prettier/recommended',
  ],
};
```

This will apply the `prettier` rule set when linting with ESLint.

Auto Formatting with VSCode

VS Code can be configured to automatically perform `eslint --fix` on save.

Create a `.vscode` folder in the root of your project and add the following to your `.vscode/settings.json`:

```
{
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
}
```

By default, we use the following overrides should be added to the VS Code configuration to standardize on single quotes, a four space drop, and to do ESLinting:

```
{
    "prettier.singleQuote": true,
    "prettier.eslintIntegration": true,
    "prettier.tabWidth": 4
}
```

Setting Up Testing

Playwright is highly recommended to be set up within a project. its an open source testing suite created by Microsoft.

To install it use this command:

```
npm install playwright
```

Since playwright shows the tests in the browser you have to choose which browser you want it to run if unless using chrome, which is the default. You can do this by

Build Validation

To automate this process in Azure Devops you can add the following snippet to your pipeline definition yaml file. This will lint any scripts in the `./scripts/` folder.

```
- task: Npm@1
  displayName: 'Lint'
  inputs:
    command: 'custom'
    customCommand: 'run lint'
    workingDir: './scripts/'
```

Pre-Commit Hooks

All developers should run `eslint` in a pre-commit hook to ensure standard formatting. We highly recommend using an editor integration like vscode-eslint to provide realtime feedback.

1. Under `.git/hooks` rename `pre-commit.sample` to `pre-commit`
2. Remove the existing sample code in that file
3. There are many examples of scripts for this on gist, like pre-commit-eslint
4. Modify accordingly to include TypeScript files (include ts extension and make sure typescript-eslint is set up)
5. Make the file executable: `chmod +x .git/hooks/pre-commit`

As an alternative husky can be considered to simplify pre-commit hooks.

Code Review Checklist

In addition to the Code Review Checklist you should also look for these JavaScript and TypeScript specific code review items.

Javascript / Typescript Checklist

- [ ] Does the code stick to our formatting and code standards? Does running prettier and ESLint over the code should yield no warnings or errors respectively?
- [ ] Does the change re-implement code that would be better served by pulling in a well known module from the ecosystem?
- [ ] Is `"use strict";` used to reduce errors with undeclared variables?
- [ ] Are unit tests used where possible, also for APIs?
- [ ] Are tests arranged correctly with the **Arrange/Act/Assert** pattern and properly documented in this way?
- [ ] Are best practices for error handling followed, as well as `try catch finally` statements?
- [ ] Are the `doWork().then(doSomething).then(checkSomething)` properly followed for async calls, including `expect`, `done`?
- [ ] Instead of using raw strings, are constants used in the main class? Or if these strings are used across files/classes, is there a static class for the constants?
- [ ] Are magic numbers explained? There should be no number in the code without at least a comment of why it is there. If the number is repetitive, is there a constant/enum or equivalent?

- [ ] If there is an asynchronous method, does the name of the method end with the `Async` suffix?
- [ ] Is a minimum level of logging in place? Are the logging levels used sensible?
- [ ] Is document fragment manipulation limited to when you need to manipulate multiple sub elements?
- [ ] Does TypeScript code compile without raising linting errors?
- [ ] Instead of using raw strings, are constants used in the main class? Or if these strings are used across files/classes, is there a static class for the constants?
- [ ] Are magic numbers explained? There should be no number in the code without at least a comment of why it is there. If the number is repetitive, is there a constant/enum or equivalent?
- [ ] Is there a proper `/* */` in the various classes and methods?
- [ ] Are heavy operations implemented in the backend, leaving the controller as thin as possible?
- [ ] Is event handling on the html efficiently done?

Last update: August 22, 2024