****

The purpose of this document is to outline the setup of this project's GitHub organization, repositories and the uses for each. This document also describes coding environment settings to ensure the code uploaded to github is valid and of the correct format.

## Table of Contents

[**Table of Contents**](#_sivvkfdgradb)2

[GitHub Organization](#_maxr3iw6l7tx) 3

[GitHub Repositories](#_q4vohtcwhgja) 5

[Repository Rules](#_9ux94bkkaa0d) 5

[Change Type](#_euepevnnou8o) 5

[New Branches](#_e305wijgp0nf) 5

[Merging changes to the main branch](#_orvwkpm7qjmh) 5

[Commit Messages](#_crb0tl61fmg2) 6

[Branch Rules](#_y2bwzih4k8y4) 7

[Frontend](#_ebp4lwd9pm0l) 9

[Static-Frontend](#_ebp4lwd9pm0l) 11

[Documents](#_a41n606tizk3) 12

[Backend](#_rb4bs0lnk0vv) 13

[**Coding Environment**](#_hiobj2zi455s)14

[Coding Editors](#_3pe9wa60ry2v) 14

[Visual Studio Code](#_3007yfbx1woz) 14

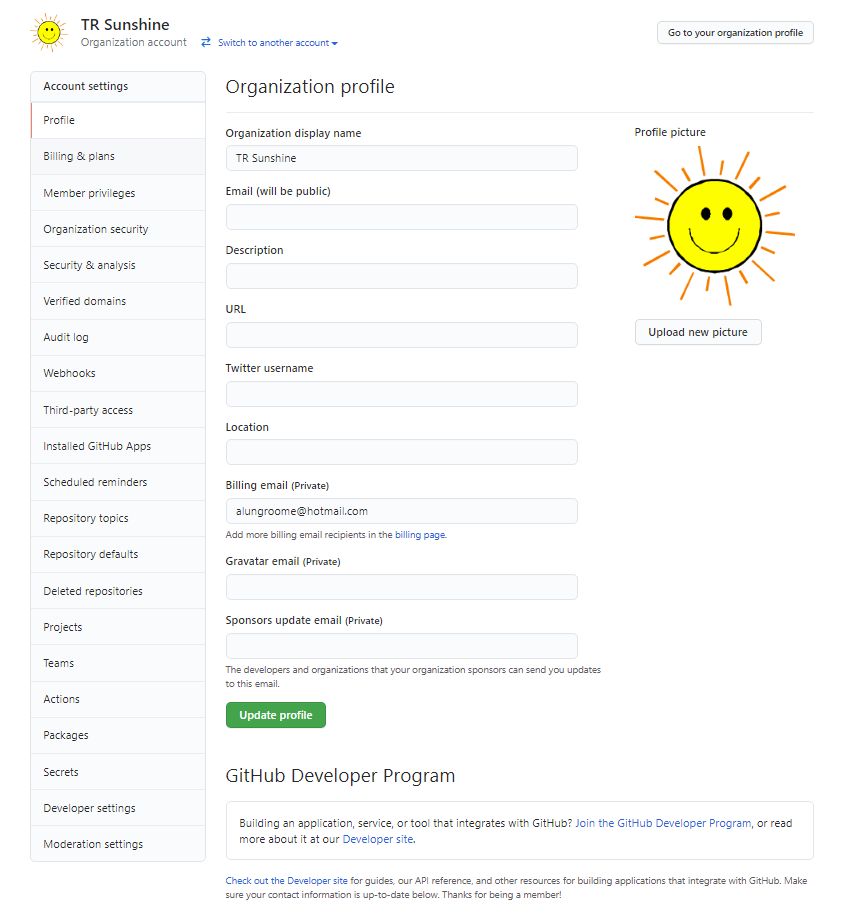
[ESLint](#_l7tpjxv824e7) 15

## GitHub Organization

A GitHub organization consists of shared accounts where businesses and open-source projects can collaborate. Owners and administrators can manage member access to the organization's data and projects.

The GitHub Organization name for this project is ‘TR Sunshine’.

Setup

1. Create the organization for the project group named ‘TR\_Sunshine’.
2. Invite all members of the project team to the GitHub organization.
3. Assign all project members ‘Owner’ Access to the organization.

## GitHub Repositories

A GitHub repository contains all of your project’s files and each file’s revision history. You can use repositories to manage your work and collaborate with others. One of the most common uses for GitHub repositories is to version control source code.

### Repository Rules

#### Change Type

Throughout the project there will be changes that fall into different categories depending on the piece of work that is going to be delivered. Each change type is categorised as below:

* FEAT: The new feature you're adding to a particular application
* FIX: A bug fix
* STYLE: Feature and updates related to styling
* REFACTOR: Refactoring a specific section of the codebase
* TEST: Everything related to testing
* DOCS: Everything related to documentation

#### New Branches

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history.

The below naming convention and rules should be used when creating a new branch:

1. The new branch name must be created with the change type (listed above) in capital letters, followed by underscore, followed by a short description of what you are developing all in lowercase
2. If the feature name is more than one word then each word must be separated with a \_

Usage

1. For creating a new branch the git command would be similar to below:
   1. git checkout -b FEAT\_sunshine\_api
2. For pushing a change to the remote branch the git command would be similar to the below commands:
   1. git add .
   2. git commit -m “FEAT: Individual\_walk”
   3. git push origin br\_sunshine\_api

#### Merging changes to the main branch

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

Usage

1. Merging to the main branch should happen by each team member once per day
2. When a member of the team has completed work in their branch they should first initiate a pull from the remote main branch to their local main branch following the below commands (FEAT\_sunshine\_api represents the name of the branch that was created which contains the code changes):
   1. git add .
   2. git commit -m “FEAT: Individual\_walk”
   3. git checkout main
   4. git pull origin main
   5. git checkout FEAT\_sunshine\_api
   6. git merge main
   7. git push origin FEAT\_sunshine\_api
3. Before merging the changes to the remote main branch you should initiate a pull request via the pull request menu at the top of the repository in GitHub.
4. Set the base branch as **main**
5. Set the compare branch to the branch containing the change
6. Press “Create pull request”
7. Specify a pull request name
8. Add a descriptive comment about each change you are wanting to merge
9. Assign two members in the team to the list of reviewers on the right hand side
10. Each reviewer will receive a notification to review the merge. Once each member in the team has performed a review and approve the final approver should perform the action of merging the change by pressing the Merge button which will merge the changes to the main branch.

#### Commit Messages

The **commit** command is used to save changes to a local repository after staging in Git.

The below naming convention and rules should be used when committing any change to any repository in this project:

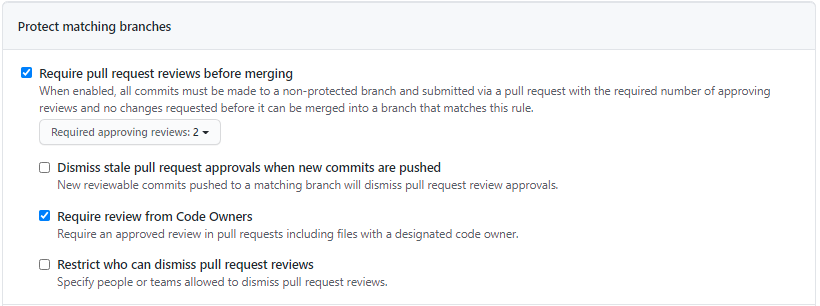
1. The first word of the commit should be the keyword from the ‘Change Type’ list above in this document with capitalization of each character.
2. Followed by a semicolon and space character
3. Followed by a brief description of the change where the first character of the sentence is capitalized
4. Total length of the commit message is limited to 50 chars
5. Commits to the repository should be little and often

Usage

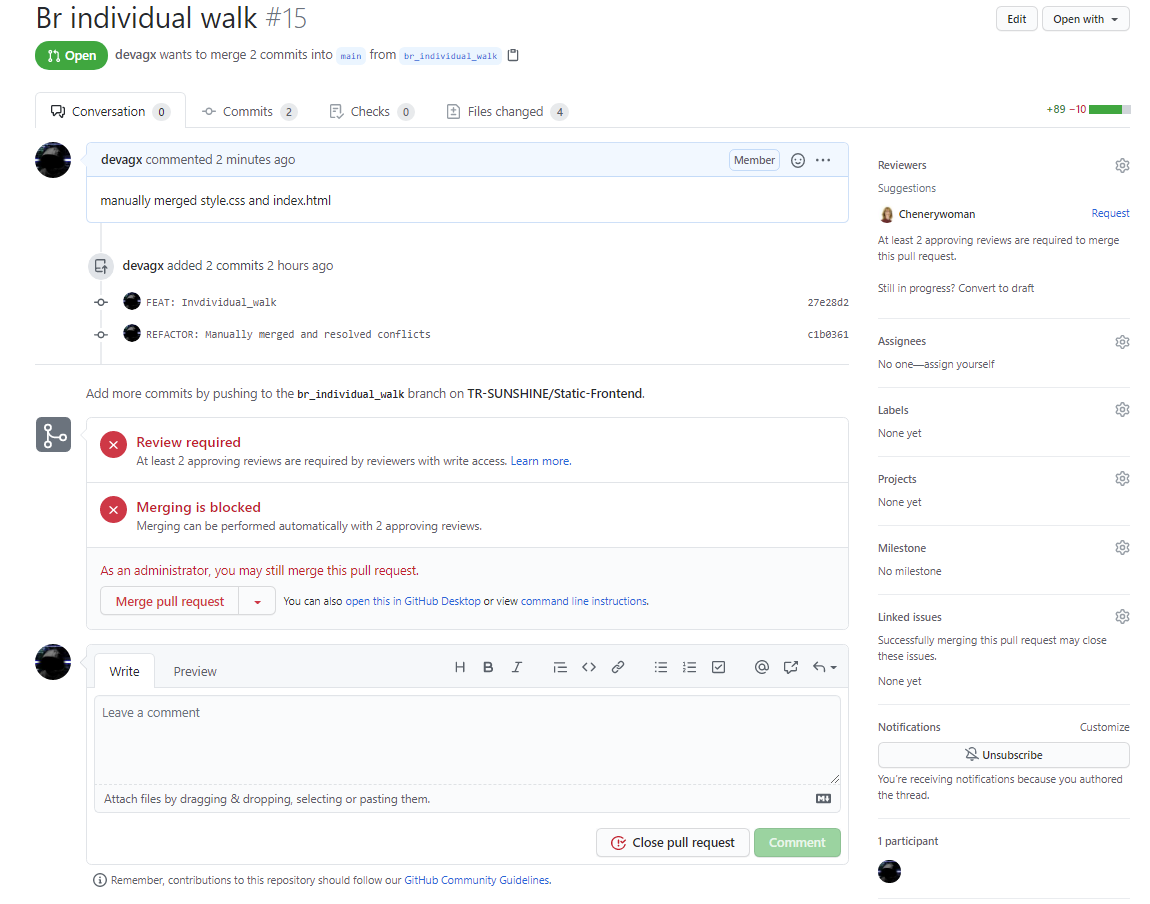
1. For fixing a bug in code the git command would be similar to below:
   1. git commit -m “FIX: Null Exception in Function GetWeather”

#### Branch Rules

The following branch rules have been applied to each coding repository to enforce a workflow for each branch

1. A two step review and one step approval is required to merge from the remote branch to the remote repository.

Below is an example illustrating the pull request requires 2 reviews and an approval:



### **Frontend**

As part of this project the Frontend repository will be used to version control the source code for the projects Frontend which is developed using React and Bootstrap.

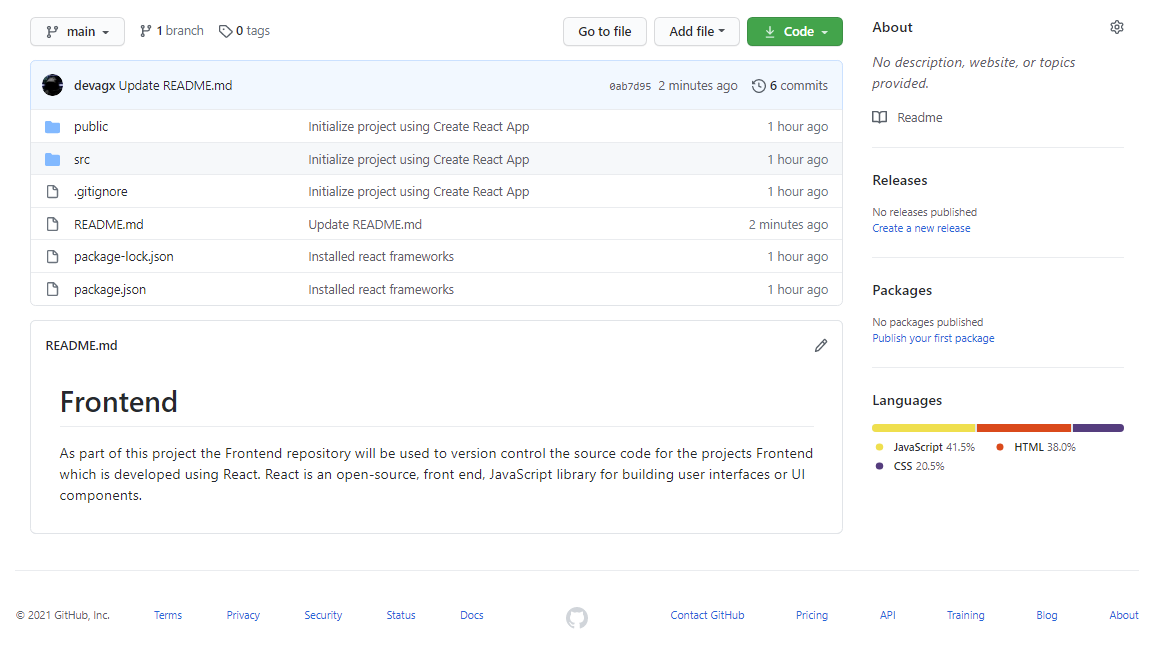
React is an open-source, front end, JavaScript library for building user interfaces or UI components.

Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases.

Initial Setup

The below outlines the steps needed for the initial setup. These steps should not be performed in an attempt to clone a repository.

1. Create a new public repository named ‘Frontend’
2. Create Initial react code base via installation under under main frontend repository
   1. npx create-react-app tr\_sunshine\_react\_application
3. Link GitHub Repository to local code
   1. Navigate to project directory
      1. cd tr\_sunshine\_react\_application
   2. Link GitHub repository to local code
      1. git remote add origin https://github.com/TR-SUNSHINE/Frontend.git
      2. git branch -M main
      3. git push -u origin main
4. Install additional frameworks and components
   1. React Routers
      1. npm install react-router-dom
   2. Material UI
      1. npm install @material-ui/core
   3. React Bootstrap
      1. npm install react-bootstrap bootstrap
5. Sync to GitHub
   1. git add .
   2. git push -u origin main



Steps to clone the repository

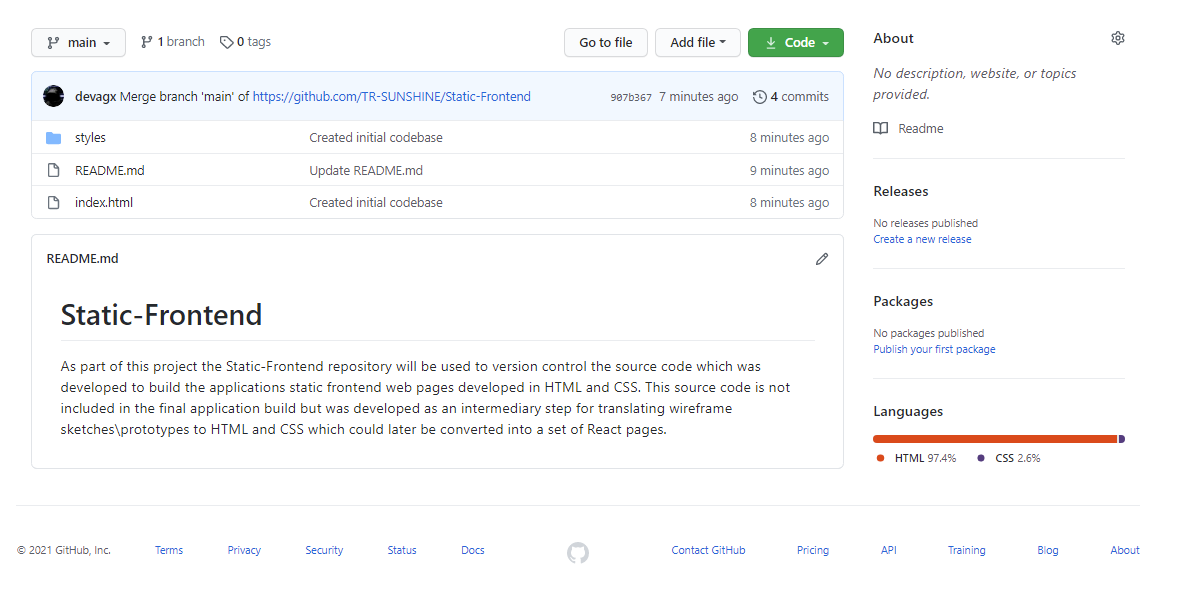
1. Navigate to working directory
2. git clone <https://github.com/TR-SUNSHINE/Frontend.git>
3. Install additional frameworks and components
   1. React Routers
      1. npm install react-router-dom
   2. Material UI
      1. npm install @material-ui/core
   3. React Bootstrap
      1. npm install react-bootstrap bootstrap

### **Static-Frontend**

As part of this project the Static-Frontend repository will be used to version control the source code which was developed to build the applications static frontend web pages developed in HTML and CSS. This source code is not included in the final application build but was developed as an intermediary step for translating wireframe sketches\prototypes to HTML and CSS which could later be converted into a set of React pages.

Initial Setup

1. Create a new public repository named ‘Static-Frontend’.
2. Create template HTML and CSS code files.
3. Link Github repository to local code
   1. echo "# Static-Frontend" >> README.md
   2. git init
   3. git add README.md
   4. git commit -m "first commit"
   5. git branch -M main
   6. git remote add origin https://github.com/TR-SUNSHINE/Static-Frontend.git
   7. git push -u origin main
   8. git add .
   9. git push origin main



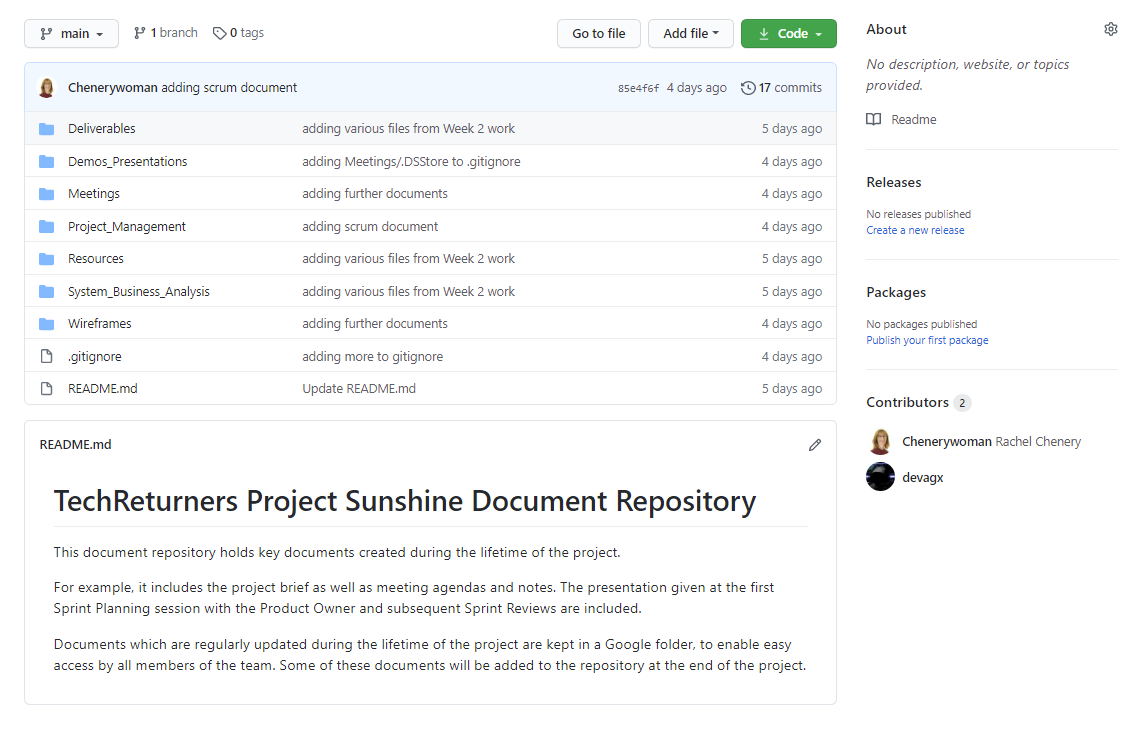
Steps to clone the repository

1. Navigate to working directory
2. git clone https://github.com/TR-SUNSHINE/Static-Frontend.git

### 

### **Documents**

As part of this project the Documents repository will be used to publicly store documentation which can be shared with TechReturners and associated businesses.



Steps to clone the repository

1. Navigate to working directory
2. git clone https://github.com/TR-SUNSHINE/Documents.git

### **Backend**

TBD.

## Coding Environment

For this project each member of the team will use a common code editor with common settings applied to ensure the code uploaded to GitHub is valid and in the correct format.

### Coding Editors

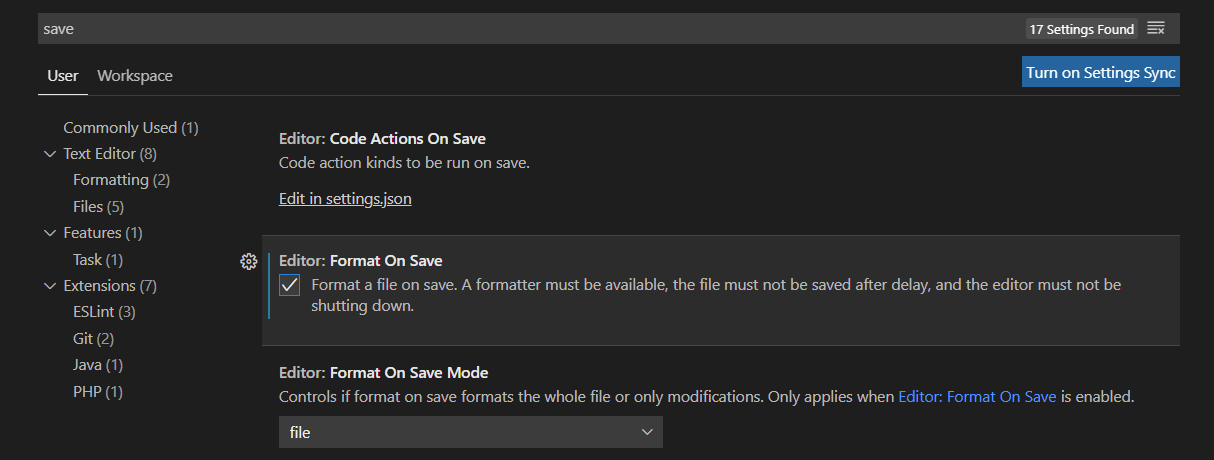
#### Visual Studio Code

Visual Studio code (VS code) will be used for the development of the following file types:

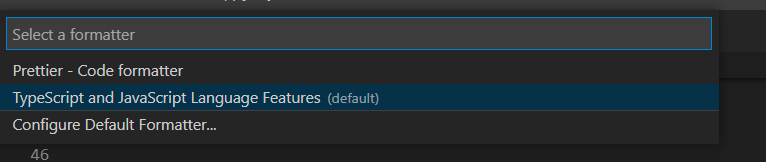
1. .js
2. .jsx
3. .css
4. .html

The below settings will be applied for each member of the team in VS Code to ensure all code files are formatted consistently to avoid code differences when merging to the main branch related to formatting:

**Format On Save** will be checked to ensure no team member forgets to format their code before committing their changes to remote branches.



**TypeScript and JavaScript Language Features** will be used as the default code formatter.



### ESLint

The JavaScript language is flexible in its coding style meaning quite often you can choose whether to use single or double quotes in strings or choose whether to end coding lines with a semicolon or not, very much like what you are enforced to do in the Java language. Because of this flexibility in the coding style this leaves room for inconsistent coding styles between different developers in the team which leads to unnecessary differences in code files when pushing to GitHub.. To ensure each team member codes consistently the below rules have been applied in the ESLint configuration in the packages.json file:

**semi** is used to enforce a semicolon at the end of coding lines. The rule accepts as its first parameter a number of 1 or 2. 1 is for warning and 2 is for error. In this project we want an error to be raised should a semicolon be missing at the end of a statement so we have chosen option number 2.

**quotes** is used to enforce consistent coding styles when defining strings and class names. The error parameter ensures the developer has no option but to use double quotes in these circumstances.

**jsx-quotes** is used to enforce consistent coding styles when defining strings and class names. The error parameter ensures the developer has no option but to use double quotes in these circumstances.

**allowTemplateLiterals** is used to allow strings to use backticks.

