Type Script 2.0

Coding Standard

Contents

[1. Introduction 3](#_Toc524364445)

[2. Variable and function 3](#_Toc524364446)

[3. Interface 3](#_Toc524364447)

[4. Namespace 3](#_Toc524364448)

[5. Classes 4](#_Toc524364449)

[6. Constants 4](#_Toc524364450)

[7. Single responsibility 4](#_Toc524364451)

[8. Small functions 5](#_Toc524364452)

[11. Service names 6](#_Toc524364453)

[12. Custom prefix for components 6](#_Toc524364454)

[13. Custom prefix for directives 6](#_Toc524364455)

[14. Unit test file names 7](#_Toc524364456)

[15. *12.End-to-End* (E2E) test file names 7](#_Toc524364457)

[16. Angular NgModule names 7](#_Toc524364458)

### Introduction

This document outline coding standards which uses TypeScript 2.0 as primary language for development.

Developers are recommended to use TSList. TSLint is an extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors. It is widely supported across modern editors & build systems and can be customized with your own lint rules, configurations, and formatters

### Variable and function

Use camelCase for variable and function names

Example:

var fooVar;

function barFunc() { }

### Interface

Use PascalCase for interface name.

Example:

interface FooBar {

}

### Namespace

Use PascalCase for namespace name

Example:

namespace fooBar {

}

### Classes

Use PascalCase for class names

Example:

class FooBar {

}

Use camelCase of class members and methods

class FooBar {

isPrimeNumber: number;

getPrimeNumber() { }

}

### Constants

**Do** declare variables with const if their values should not change during the application lifetime

**Consider** spelling const variables in lower camel case.

Example:

export const mockHeroes = ['Sam', 'Jill']; // prefer

export const heroesUrl = 'api/heroes'; // prefer

export const VILLAINS\_URL = 'api/villains'; // tolerate

### Single responsibility

Apply the [*single responsibility principle* (SRP)](https://wikipedia.org/wiki/Single_responsibility_principle) to all components, services, and other symbols. This helps make the app cleaner, easier to read and maintain, and more testable.

Example :

app.component.ts

app.component.html

app.component.css

app.component.spec.ts

### Small functions

Do define small functions

Consider limiting to no more than 75 lines.

Why? Small functions are easier to test, especially when they do one thing and serve one purpose.

Why? Small functions promote reuse.

Why? Small functions are easier to read.

Why? Small functions are easier to maintain.

1. **Naming conventions**

Separate file names with dots and dashes.

Do use dashes to separate words in the descriptive name.

Do use dots to separate the descriptive name from the type.

Do use conventional type names including .service, .component, .pipe, .module, and .directive. Invent additional type names if you must but take care not to create too many.

Why? Type names provide a consistent way to quickly identify what is in the file.

Why? Type names make it easy to find a specific file type using an editor or IDE's fuzzy search techniques.

Why? Unabbreviated type names such as .service are descriptive and unambiguous. Abbreviations such as .srv, .svc, and .serv can be confusing.

1. **Symbols and file names**

Do use consistent names for all assets named after what they represent.

Do use upper camel case for class names.

Do match the name of the symbol to the name of the file.

Do give the filename the conventional suffix (such as .component.ts, .directive.ts, .module.ts, .pipe.ts, or .service.ts) for a file of that type.

|  |  |
| --- | --- |
| **Symbol Name** | **File Name** |
| @Component({ ... })  export class AppComponent { } | app.component.ts |
| @Component({ ... })  export class HeroesComponent { } | heroes.component.ts |

### Service names

**Do** use consistent names for all services named after their feature.

**Do** suffix a service class name with Service. For example, something that gets data or heroes should be called a DataService or a HeroService.

|  |  |
| --- | --- |
| **Symbol Name** | **File Name** |
| @Injectable()  export class HeroDataService { } | hero-data.service.ts |
| @Injectable()  export class CreditService { } | credit.service.ts |

### Custom prefix for components

**Do** use a hyphenated, lowercase element selector value (e.g. admin-users).

**Do** use a custom prefix for a component selector. For example, the prefix admin represents an admin feature area.

**Do** use a prefix that identifies the feature area or the app itself.

Example

App/heroes/hero.component.ts

@Component({ selector: 'hero' }) export class HeroComponent {}

### Custom prefix for directives

**Do** use a custom prefix for the selector of directives (e.g, the prefix toh from **T**our **o**f **H**eroes).

**Do** spell non-element selectors in lower camel case unless the selector is meant to match a native HTML attribute.

@Directive({ selector: '[tohValidate]' }) export class ValidateDirective {}

### Unit test file names

**Do** name test specification files the same as the component they test.

**Do** name test specification files with a suffix of .spec.

|  |  |
| --- | --- |
| **Test Type** | **File Names** |
| Components | heroes.component.spec.ts  hero-list.component.spec.ts  hero-detail.component.spec.ts |
| Services | logger.service.spec.ts  hero.service.spec.ts  filter-text.service.spec.ts |
| Pipes | ellipsis.pipe.spec.ts  init-caps.pipe.spec.ts |

### 12.End-to-End (E2E) test file names

**Do** name end-to-end test specification files after the feature they test with a suffix of .e2e-spec.

|  |  |
| --- | --- |
| **Test Type** | **File Names** |
| End-to-End Tests | app.e2e-spec.ts  heroes.e2e-spec.ts |

### Angular NgModule names

Do append the symbol name with the suffix Module.

Do give the file name the .module.ts extension.

Do name the module after the feature and folder it resides in.

|  |  |
| --- | --- |
| **Symbol Name** | **File Name** |
| @NgModule({ ... })  export class AppModule { } | app.module.ts |
| @NgModule({ ... }) export class AppRoutingModule { } | app-routing.module.ts |