2MUCHCOFFEE

2MUCHCOFFEE’S ANGULAR MANIFESTO

Welcome to 2muchcoffee’s professionals family - open source contributors with a distinct development concept, personal approach, and precisely measured results.
Enjoy your coffee while we’re building your app!

CONTENTS

We are honored to present an Angular Manifesto based on Angular's team recommendations, commonly used best practices, and expanded with our personal long-term experience and technical expertize.Enjoy your coffee while we're building your app!

FUNDAMENTAL ARCHITECTURAL POINTS

1. Use the Angular CLI for initializing, developing, building, scaffolding and maintaining Angular applications;

2. File structure - use commonly used practices declared by an Angular's team and follow the LIFT principle;

3 Use the Core, Shared and Feature modules for better code management and architecture:

3.1. The Core module:

- Should include only singletons: Services, Models, Interceptors, Resolvers, application-wide components that are being used only once in the AppComponent template;

- Prevent re-import of the Core module.

3.2. The Shared module:

- Should contain commonly used/re-used throughout the app Components, Directives, Pipes;

- Should contain third-party libraries modules that are used at least in 50% of modules across the app;

- Should not contain third-party libraries modules that are heavyweight unless it's used in most of the modules throughout the app;

- In case the main eager-loaded module, e.g. "HomeModule" doesn't need most of the Shared module imports, then you should import to it only specific modules/components etc.

4. Implement the lazy-load workflow where it’s possible.

5. Services, Components, and functions inside them should be readable, not bulky and primary follow the single-responsibility principle. Consider:

- Limiting files to 400 lines of code;
- Limiting functions to no more than 75 lines.

6. Extract templates and styles into a separate file.

7. Use libraries that are written specifically for Angular, otherwise create Angular wrappers for them.

BASIC ANGULAR-SCOPE POINTS

1. A Component should conform to the following structure:

- static properties;
- @Input properties;
- properties(public first, then private);
- accessors(public first, then private);
- constructor method implementation;
- lifecycle hook methods implementations;
- methods(public first, then private).

2. Appropriate naming:

- The file names and the element selectors of Components should be written in a dashed-case(kebab-case);
- Always give the filename the conventional suffix (such as .component.ts, .directive.ts, .module.ts, .pipe.ts, .service.ts or .spec.ts) for a file of that type;
- Suffix a Module/Component/Directive/Service class name with Module/Component/Directive/Service relatively, e.g. UsersComponent, DataService;
- Use consistent names for all Services and Pipes named after their feature;

- Use the lower camel case for naming variables, including “const”, and methods;
- Use the lower camel case and a custom prefix for naming the selectors of Directives;
- Use the upper camel case for naming the Classes and Interfaces;
- Consider using a class instead of an Interface, otherwise, consider naming an interface without an I prefix;
- Avoid prefixing private properties and methods with an underscore;
- In case an Accessor or a variable contains a stream, then its name should include the “$” sign in the end;
- In case a variable contains a boolean value - the name of such a variable should start from the “is” preposition, e.g. “isAdmin”;
- Don’t prefix the output properties with, e.g. the suffix “on”;
- The properties and functions names should be explicit and directly conform to its functionality;

3. Keep the presentation logic in the Component class, and not in the Template.

4. Limit logic in a Component to only that required for the View - delegate complex Component's logic to a Service.

5. Use Services for any server data processing, e.g. API requests, Actions dispatching etc.

6. The direct references to global objects, e.g. window, document are not allowed for permanent usages. To reference the window object you have to create a specific service that can be injected throughout the app. For example, in case you need to use the Document object, Angular provides it in the form of constant from @angular/common;

7. Preferably use pure pipes with only pure functions. In case you need to call a function in a template - make it memoized through a pipe.

MAIN RXJS POINTS

1. Use async pipe preferably.

2. Never subscribe in another subscribe - use appropriate RxJS operators instead, i.e the “switchMap” operator.

3. Don’t subscribe in methods - use for subscriptions Angular lifecycle hooks that are called only once after calling the constructor.

4. Avoid subscriptions in a Service if it’s not a singleton.

5. Always unsubscribe from a stream in the ngOnDestroy lifecycle hook in case it’s not possible to use the async pipe.

6. Preferably don’t combinate Promises or the RxJS operator “toPromise” with Observables.

7. Never use the setTimeout - use appropriate RxJS operators.

8. Don’t use RxJS timeline operators if you just need to run through the ngZone.

9. Use RxJS Subjects for temporary data saving and handling events;

CODE QUALITY POINTS

1. tslint.json - create/use a distinct, unified and consistent coding, naming, and whitespace conventions.

2. Use Typescript typing power - provide type definitions for all declarations across the app.

3. Consider using the “ordered-imports” tslint rule - files imports:

    3.1. Should be alphabetized;

    3.2. Should be structured according to a commonly used style guide, described in TSLint rules as a “groups” option:

    - Angular imports;
    - Third party libraries imports;
    - Parent level application imports;
    - Current level application imports;
    - Children level application imports.

    3.3. Each type of imports should be separated with an empty line.

4. Use the @Input and @Output() class decorators instead of the inputs and outputs properties of the @Directive and @Component metadata. Consider placing decorators on the same line as the property it decorates.

5. Use the @HostListener and @HostBinding instead of the host property.

6. Use current and supported libraries only.

OPEN SOURCE NGX-RESTANGULAR

Ngx-restangular is an Angular 2+ service that simplifies common GET, POST, DELETE, and PUT requests with a minimum of a client code. Ngx-restangular is responsible for communication between API and complex frontend web apps.

FEATURES

1. Self-linking elements support;

2. Supports both - Promises and Observables;

3. Send a request from/within an object - don’t create a new object for each request;

4. Supports nested RESTful resources;

5. Use meaningful names instead of URLs;

6. Provides an ability to create your own HTTP methods;

7. Send requests easily using different settings;

GITHUB

YOU’RE WELCOME TO CONTACT OUR CEO

DMITRIY MELNICHENKO

dmytro@2muchcoffee.com

2muchcoffee.com

2MUCHCOFFEE

sales@2muchcoffee.com

2muchcoffee.com