Best Practices for a Clean and Robust Angular Application

This post outlines some of the best practices we can use in our application to support robust and a highly scalable Angular framework. This write-up is related to Angular, Typescript and RxJs. Additionally, the blog discusses some general coding guidelines to make the application cleaner. Let us begin by evaluating the various features available in the Angular application framework.

strict:true

If you are starting a new project, it is worth setting strict:true in the tsconfig.json file to enable all strict type checking options.

Example

Inside tsconfig.json file add the below entry in compiler options:

```
"compilerOptions": {
"strict": true,
}
```

ViewEncapsulation

Use the ViewEncapsulation option based on the requirements. At some occasions, you might want to adopt the parent styles for child components.

Example

```
@Component({
selector: 'my-app',
template: `<h1>Hello World!</h1>`,
styles: [`:host { display:block; } h1{ color:blue; }`],
encapsulation: ViewEncapsulation.None
})
```

Here is a link for your reference.

Try Catch

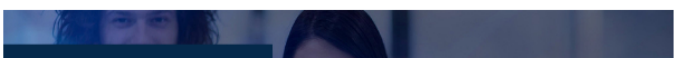
Whatever code you write in ts file, try to place it in the catch block.

Explanation

To avoid application breakage. In case, you have missed a null check or accessing some object which is not defined, during the run time the application will break, and the user will not be able to perform any action.

Example

```
Try {
//your code goes here
} catch(e){
Console.log("Error occurred"+e);
}
```



Type keyword(s) here

Search

Categories

.Net
Application Security
Automation Testing
Automation Testing
Azure
Big Data
Blockchain
Bonita BPM
Big Data Process Management
Blockchain
Bonita BPM
Business Process Management
Code Review
Covid19
Cyber Security
Data Science
Development Tools
DevOps
Digital Business
Digital Marketing
Docker
E-Commerce
Enterprise Mobility
Evoke
Evoke Technologies
Hadoop
Java
JavaScript Frameworks
Microsoft
Mobile Application Development
Mobility
Open Source
Open Source Frameworks
Oracle
Oracle EBS
Payroll Management
Pega
Performance Testing
PHP
Programming Languages
Project Management
Quality Assurance
R Language
RPA
Salesforce
SharePoint
Software Development

Routing (Route Guard, Auth Guard)

Use this option, when there is a need to switch between different pages within the application. This makes the code look cleaner and easy to understand. Make maximum use of the Route Guard and Auth Guard.

Lazy Load

Wherever possible, try to Lazy Load the modules in your Angular application. Lazy loading means, you load something only when it is used. For instance, loading a component only when it is to be seen.

Explanation

This will reduce the size of the application to be loaded and can improve the application boot time by not loading the modules that are not required.

Example

Before:

```
{
path: 'not-lazy-loaded', component: NotLazyLoadedComponent
}
```

After:

```
{
path: 'lazy-load', loadChildren: 'lazy- load.module#LazyLoadModule'
}
```

ChangeDetectionStrategy

Use this strategy carefully based on your requirements. You might run into issues sometimes by (always) allowing the changedetectionstrategy mostly in the development mode.

Let, Const, Lambda or Arrow Functions

When declaring variables, use const when the value is not been reassigned.

Explanation

Using let and const appropriately makes the intention of the declarations clear. It will also help in identifying issues when a value is reassigned to a constant accidentally, by throwing a compile-time error. Further, it also helps to improve the readability of the code.

Example

Before: let empId = “12345”;
After: If empId is always constant then change it to const empId=”12345”;

trackBy

When using ngFor to loop over an array in templates, use it with a trackBy function which will return a unique identifier for each item.

Explanation

When an array changes, Angular re-renders the whole DOM tree. But if you use trackBy, Angular application framework will know which element has changed and will only make DOM changes for that particular element.

Example

```
Before: <li *ngFor="let item of items;">{{ item }}</li>
After: <li *ngFor="let item of items; trackBy: trackByFn">{{ item }}</li>
// in the component
trackByFn(index, item) { return index // or item.id; }
```

No Function Calling in NgIf

Do not call functions in NgIf.

Explanation

NgIf executes on each DOM change and there are chances that the browser might reach the maximum call stack size and throw a script error.

Example

Software Development
Software Quality
Software Testing
Test Factory
Tutorial
UI/UX
Uncategorized
Web Application Development
Web Application Frameworks
Web Design
Website Development
Workflow Testing

ARCHIVES

August 2021
July 2021
June 2021
December 2020
November 2020
October 2020
September 2020
August 2020
July 2020
May 2020
April 2020
January 2020
November 2019
July 2019
May 2019
April 2019
March 2019
February 2019
January 2019
December 2018
November 2018
September 2018
June 2018
April 2018
March 2018
February 2018
January 2018
December 2017
November 2017
October 2017
September 2017
August 2017
July 2017
June 2017
May 2017
March 2017
February 2017
December 2016
November 2016
October 2016
September 2016
August 2016
July 2016
June 2016
May 2016
March 2016

```
Before: <div *ngIf=”showData()”></div>
After: <div *ngIf=”showData”></div>
//inside component
public showData: boolean = false;
show() { this.showData = true; }
```

Do Not Repeat Yourself

Make sure you do not have the same code copied at different places in the codebase. Extract the repeating code and use it in place of the repeated code.

Explanation

Having the same code in multiple places means that if we want to make a change to the logic in the code, we have to do it at multiple places. This makes it difficult to maintain and prone to bugs. Further, it takes time to make changes to the logic. In those cases, extract the repeating code and use it instead, which means there would be only one place to change.

Clean up Subscriptions

Destroy the Subscriptions to avoid any memory leak. When subscribing to Observables, always make sure you unsubscribe them by using operators such as take, takeUntil, etc.

Explanation

Failing to unsubscribe from the Observables will lead to unwanted memory leaks if the Observable stream is left open. Potentially, even after a component has been destroyed or the user has navigated to another page.

Example

Before:

```
anObservable.pipe(map(value =>value.item)).subscribe(item => this.textToDisplay = item);
```

After:

```
private _destroyed$ = new Subject(); anObservable.pipe(map(value => value.item),takeUntil(this._destroyed$)
```

Avoid Including Subscriptions Inside Subscriptions

Sometimes you may want values from more than one Observable to perform an action. In such cases, avoid subscribing to one Observable in the Subscribe block of another Observable. Instead, use appropriate chaining operators. Chaining operators run on Observables from the operator before them. Some chaining operators are withLatestFrom and CombineLatest.

Example

Before:

```
firstObservable$.pipe().subscribe(firstValue => {
secondObservable$.pipe().subscribe(secondValue => {
console.log(`Combined values are: ${firstValue} & $ {secondValue}`); }); });
```

After:

```
firstObservable$.pipe(withLatestFrom(secondObservable$),first()).subscribe(([firstValue, secondValue]) => {
```

Subscribe in Template (Angular 5 and Above)

Avoid subscribing to Observables from components. Instead, subscribe to the Observables from the template.

Explanation

Async Pipes unsubscribe themselves automatically making the code simpler by eliminating the need to manually manage subscriptions. Further, it reduces the risk of accidentally forgetting to unsubscribe a subscription in the component, which would possibly cause a memory leak.

Example

Before:

```
// template <p>{{ textToDisplay }}</p>
// component someObservable.pipe(map(value => value.item)).subscribe(item => this.textToDisplay = item);
```

After:

```
// template <p>{{ textToDisplay$ | async }}</p>
// component this.textToDisplay$ = someObservable.pipe(map(value => value.item));
```

AOT (Ahead-of-Time Compilation)

With AOT, the browser downloads a pre-compiled version of the application. The browser loads executable code, so it can render the application immediately, without waiting to compile the app first.

Explanation

Faster rendering, fewer asynchronous requests, smaller Angular application framework download size.

Unit Testing

If possible, write unit test cases (code coverage) using Jasmine or any other unit testing tool.

Explanation

This will ensure that your code is not breaking anywhere. Additionally, it also ensures that you don't have any unused imports. Here's a link for your reference.

Note

- Avoid NgIf for SVG tags, Internet Explorer web browser throws an error for this syntax (i.e. for older versions of Angular, below Angular 4).
- Do not use NgFor and NgIf on the same element, Angular will not allow it.
- Use ngContainer to loop NgFor and use NgIf on the immediate element.

Additionally, you can go through the official Angular style guide to learn about the best practices.

Conclusion

Building web applications and scaling them is a continuous exercise, and there's always scope to improve the way we write code and build apps. This list of optimizations is a great place to start and applying these things to your project will make your application clean, less buggy and enhance the angular application performance.

Evoke's UI/UX Services

With over 15 years of experience, Evoke provides a wide-ranging UI/UX services that delight end-users. We can help your enterprise by creating a custom UI/UX road-map that aligns with your business goals and objectives. With an experienced UI/UX practice and years of experience in designing apps both web and mobile, we enable your enterprise to gain a clear business-edge.

Our UI/UX team works with you to understand your business needs and creates an intuitive and attractive UI/UX solution that not only optimizes your end-users experience but adds great business value. To learn more about our UI/UX solutions, contact us online or call us at +1 (937) 660-4923.

Tags: angular | angular application | angular best practices | angular frameworks | angular tutorial | angular typescript | angular web application

May 3, 2019 1

SHARE TWEET SHARE PIN IT

WRITTEN BY ABHINAV MALLADI

Abhinav Malladi is a UI developer at Evoke Technologies and holds expertise in various UI technologies like JavaScript, jQuery, AngularJS, Angular7, Html5, CSS3. He loves spending his free time exploring latest technologies and binge-watching movies.

‹ Previous post

Next post ›

1 Comment

Write a comment

DWIGHT SPENCER

May 7, 2019 at 6:03 pm

REPLY

Unit testing is a fool's errand and heavily discredited. Rephrased: It is a waste of time and money and does not do true QA. But it sold a lot of books to managers. Get up to date by having a good discussion with James Coplien and other SDLC process leaders.