Article
Sergei Sarkisian · Jul 1 8m read

Angular. General tips

Before we start with some intermediate and advanced topics, I would like to sum up some more general points. They are subjective, of course, so I will be happy to discuss them if you have other opinion or better arguments for any of them.

The list is not comprehensive and this is intended, cause I will cover some topics in future articles.

Tip 1. Follow official styleguide

Angular is quite strict in terms of limiting the possible architecture of an application, but there are still many places in it that allow you to do things your own way. The imagination of the developers is limitless, but sometimes it makes the job difficult for those who work on the project with you or after you.

The Angular team does a good job by maintaining the Angular architecture itself and it's libraries, so they definitely know how create stable and supportable codebase.

I recommend follow their official styleguide and deviate from it only if things don't work that way. This will make things easier when you will come to new project or if anyone will come into your project.

Remember, supportable, stable and easy to understand code and app architecture is more important than clever but cryptic solutions than nobody will able to catch up (possibly even you in the future).

Official Angular styleguide: https://angular.io/guide/styleguide

Tip 2. Consider to buy Angular Ninja book

You may think that this is ad, but here is the thing: that's really good book with all the main concepts of Angular covered and the price is up to you. You can also choose how much money will go to writers and how much will go to charity.

One of the authors of this book is a member of Angular teem, so this is definitely the most reliable source of information about Angular after the official documentation. You can see the chapters of the book on the book's page and read sample chapters to decide if it's worth your buck.

Moreover, the book is updated with the release of a new version of Angular and you get all updates to the book for free.

The Ninja Squad blog itself is very good source of information about Angular, with articles and news about new versions, best practices, experimental features and more.

Angular Ninja book: https://books.ninja-squad.com/angular

Tip 3. Read official documentation

Before you dive in into the writing of some code of your app it's a good idea to read through official documentation

and guides, especially if you start your project on a version of Angular you hadn't use before. Things keep getting deprecated all the time, tutorials and guides in the Internet could be outdated and you may end up with growing your technical debt instead of using new best practices and functionality.

That's a good habit also to check for which version the guide was written for. If it's 2+ version behind the one you are using then it's better to check if things have changed since then.

Official documentation: https://angular.io

Tip 4. Consider of using Angular CDK even if you don't use Angular Material

I will cover it better in future articles, but I know that many Angular Developers don't even know about Angular CDK.

Angular CDK is a library of useful directives and base classes that can help you with developing better applications. For example, it has such things as FocusTrap, Drag & Drop, VirtualScroll and more which can be easily added to your components

Angular CDK: https://material.angular.io/cdk/categories

Tip 5. Fix your dependencies in package.json

It's not particularly related to Angular and it might be important in any project. When you make npm install --save <something> , it will be added to your package.json with ^ or ~ in the beginning of package version. It means that your project will be able to use any minor/patch version of the same major version of the dependency. This will go wrong in the future with almost 100% probability. I faced this issue in different projects so many times. Time passing by and new minor version of some dependency coming out and your app won't build anymore. Because an author of this dependency updated it's dependencies in minor (or even patch) version and now they are in conflict with your build. Or may be there is a new bug presented in new minor version of your dependency and you have no problems in your code (cause you installed your dependencies before the new version came up) but anyone else trying to install and build your project from repository will face the bug you even don't aware of (and you will not be able to reproduce it on your machine).

package-lock.json exists to solve this problem and to help developers have the same set of dependencies across the project. Ideally it should be committed to repository, but many developers are deciding to add this file to .gitignore. And id it's gone, you may end up with problems above. So better do not blindly trust your dependencies resolution and fix it on the specific version, regularly scan it for vulnerabilities (with npm audit) and then update and test it manually.

Tip 6. Try to upgrade your app to the new Angular version as soon as you can

Angular is evolving continuously and new versions are releasing every 6 months or so. Keeping your app updated will allow you to use all the new features, including faster builds and other optimisations. Also, upgrading to the next major version of Angular should not be a big problem in this case. But if you are 5 versions behind and your application is mid or large size, the process of updating to last version can be tough as Angular has no schematics for upgrading apps skipping intermediate Angular versions. You will need to upgrade through all the intermediate versions one-by-one, checking that the application works on each version. The direct update without Angular CLI schematics is possible, but could also be tricky, so I'm suggesting keeping your app fresh and updated.

Tip 7. Try to reduce your dependency list

It's easy to fall in habit of bringing new dependency in your app as you need new functionality. But with each dependency the complexity of your app is growing and the risk of sudden technical debt avalanche is increasing.

If you decided to add new dependency in your app think about this:

- How well supported the dependency is?
- How many people using it?
- How big the development team?
- How quickly they are closing GitHub issues?
- How well documentation of the dependency is written?
- How quickly it is updated after new major of Angular came out?
- How big an impact of this dependency to performance and bundle size of your application?
- How difficult will be to replace this dependency if it will be abandoned or deprecated in the future?

If some of this questions have negative tone answer, consider to get rid of it or at least to replace it for something more mature and well-supported.

Tip 8. Do not use <any> as your "temporary" type

Again, it's easy to start using any as you write your business logic, cause writing proper types could be time-demanding, and you need finish your task in this sprint. Types can be added later, of course. But it is slippery slope to ramp up technical debt.

An architecture of your app and types should be defined before your write any business logic. You should clearly understand what objects you will have and where they will be used. Specification before the code, right? (Tom Demarco wrote a book about it before it went mainstream: https://www.amazon.com/Deadline-Novel-About-Project-Management/dp/0932633390).

If you are writing the code without pre-defined types, you might end up with worse app architecture and functions which use very similar but different objects. So you will need either create different types for each function (which will do things even worse) or spend your time for writing specification, types **AND** refactoring which is a waste of time compared to if it had been done before.

Tip 9. Spend your time to understand the build process

Angular does a great job to facilitate the work of the developer regarding the building of the project. But the default options not always best for every case.

Spend your time to understand how build process works in Angular, what the differences between Development and Production builds, which options Angular has for builds (like optimisations, source maps, bundling and more).

Tip 10. Investigate what's inside your bundle

Not all libraries provide tree-shaking and not always we do imports right way, so there is always a chance that something redundant get bundled with our app.

So it's good habit to investigate the content of your bundle from time to time.

There are good articles describing the process using webpack-bundle-analyzer , so I won't cover it here, but here is a link for one of them: https://www.digitalocean.com/community/tutorials/angular-angular-webpack-bundle-analyzer

I will cover this topic more detailed later in the series.

Tip 11. Use services providedIn property instead of importing the service in the module

Many Angular code examples on the Internet use importing the services in the modules. But it is not preferred way of declaring services since Angular 6 or 7. We should use @Injectable decorator property providedIn for enabling tree-shaking and better bundling of our applications. Angular is smart enough to understand in which bundle the service should be included in, when it should be initialized and how many instances of the service should be created.

There are three values which providedIn accepts. Most of the time root is enough, but there are two more:

- root: the service will be singleton in scope of application
- any: one instance of the service will e created for all the eagerly loaded modules along, with different instance created for every lazy module
- platform: the service will be singleton for all the applications running on the same page

Tip 12. Do not forget main performance rules

- Use trackBy for collections to reduce redrawing operations and JavaScript execution
- Use onPush change detection strategy in every place you can (I will cover it in the dedicated article)
- Perform heavy calculations outside of ngZone
- Use throttle and debounce patterns with your events to prevent unnecessary server calls and event flooding
- Use Virtual Scrolling for displaying big data sets
- Use pure pipes for data transformation in your templates
- Use AoT compilation
- Lazy load the parts of your app which not necessary for application to start
- Avoid computations and conditional function calls in your templates (function calls should be used only for events)

Thanks for reading! Hope that some of this tips were useful for you. If you have any comments and remarks, please, let me know in the comments, I will be glad to discuss

See you!

#Angular #Angular2 #Coding Guidelines #Frontend #UI Development #Other

Source URL: https://community.intersystems.com/post/angular-general-tips