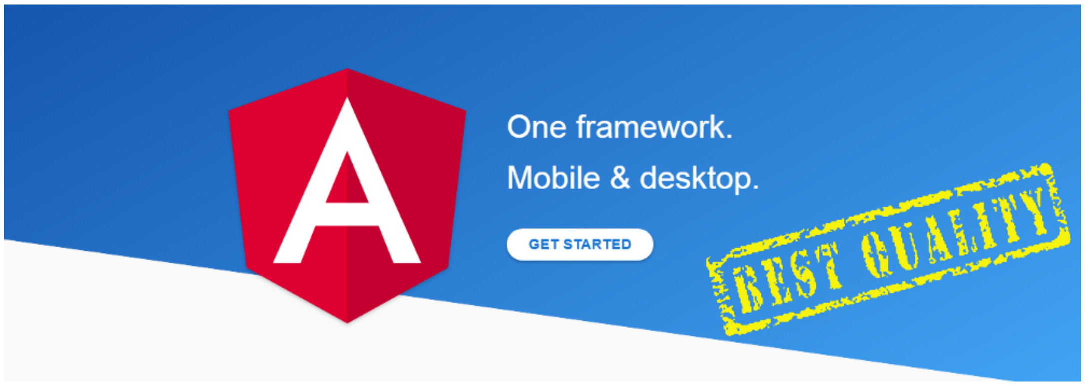
Best-practices learnt from delivering a quality Angular4 application



Best Practices for developing with Angular Framework

Back in Sept 2016, just when Angular team bit the bullet and released Angular2 Final, I was able to convince my customer to use Angular2 for it's one of bigger applications.

As some of you might recall, Angular2 went through an unusually long Alpha, Beta and RC stages. It seemed as if the entire Angular2 was re-written since the first Alpha release. So at the time of 2.0 final release, the entire Angular scene was ***very***chaotic. There were hardly any good tutorials or resources which were working with 2.0.0.Final release.

I also did not have AngularJS(1.x) background. I had just delivered a huge SPA using Backbone-Marionette-Rivets.js stack. In the hindsight, it was a good thing to not have baggage from AngularJS!

> *All in all, we took a leap of faith! I placed my faith on Angular developers, community and my ability to adapt to new framework and jumped into the valley without a parachute!*

Me and my core team members—we had about 3–4 weeks time to create the first spike and we all ran fence to fence; fell often but learnt a lot. My overall experience of JavaScript development scene also came handy when we scaled from 4 people to about 18 people team in couple of months.

Looking back, after 6–8 months of development and product delivery of the application, I can see that some good practices saved the day. This post summarizes them for everyone's benefit. Without further ado, here are some of the best practices that you might want to adopt to deliver a quality Angular application…

Best practices for Absolute beginners

Become comfortable with ES2015

Most of the initial curve for angular is just about getting comfortable with ES2015. So ensure all the developers on the team have READ and actually TRIED ES2015 and ES2016 flavors of JavaScript. There is A LOT to learn here but it will just make ready to face the external world tutorials which often makes use of these syntax. E.g. syntax like `() => { }` or `[…a, b,]` should not trip you. Or usage of `import` , `class` , `let` , `const` , etc should be first nature to your developers.

Embrace Typescript and Visual Studio Code (VSCode).

Most of the code snippets for Angular you will find online are in Typescript.. which is a superset of ES2015. I will highly recommend that you use this so that code snippets online will make sense. Also as a companion, use Visual Studio for Code as your IDE, TSLint as your linter and TSLint plugin in VSCode to ensure you get best static code analysis experience. Plus—by using TS, you don't need Babel. **Bonus**: Also add Angular Language Service plugin to VSCode. This gives far better angular experience especially in the Angular templates.

Master npm ecosystem.

Along-side ES2015, Angular is also all about being comfortable with Node and NPM Ecosystem. Any serious example will make use of package.json (npm) and node to build and run their example. Virtually EVERY angular component out there will give you instructions about how to install it using npm. So make absence of Npm and VSCode deal-breaker for your teams. Either your developers are using these tools or they are not on your team! Seriously!

Angular Application Development Best Practices

Eat, Sleep, Breath Components!

Angular is all about components. Design the components first, before starting to code. By design, I mean –

> *Draw outlines on the Visual-Designs to clearly demarcate which screen area will be owned by which component.*

> *Make the components small enough so that they can be reused at many places But large enough that making them any smaller makes no sense. It takes a bit of time to get used to creating this this logical grouping but you can naturally do this in 2–3 sprints. I insist on my entire team doing this for EVERY story in EVERY sprint.*
>
> *Once you know your component, document the "inputs" and "outputs". I have a small design-checklist which I make every developer fill-up as a short design documentation for each story. Please see* ***Design Narrative*** *section at the bottom of below this post if you want to adapt it in your project.*

Design each component with Re-usability in mind. Try to create commonly used UI element as separate component and re-use them in the screens.

Use seed projects to hit the ground running!

Make use of some kickass starter seed projects because they would have done a great job at incorporating many features for you. I wholeheartedly recommend AngularClass webpack starteror BlackSonic's ES6 starter. This will get you running in no time with a great foundation for large project.

Or… Use Angular-CLI

Other option is to use Angular-CLI. Angular CLI is really good option for those who are finding entire ES2015+TypeScript+Angular a bit overwhelming. It abstract away quite a few things from you including entire webpack configuration. But that abstraction is also a downside since you cannot tinker around those abstracted parts. Thankfully, there is a `eject` option in Angular-CLI to eject most of abstracted things.

RIP SystemJS, Hello Webpack!

From the beginning, stop using SystemJS and switch over to Webpack. Webpack is far more powerful and versatile tool. Optimize webpack bundles effectively to ensure that you are not bundling same modules in multiple chunks. There are bundle-analyzers from webpack which do brilliant job of telling you about this. **Bonus**: Webpack Learning Slides and Step-by-step code

Use AoT FTW!

Usage of AoT (ahead of time) compilation is a great step towards performance gains at runtime. It also reduces your bundle by about 30kb (gzipped) which is a LOT of improvement. Angular 4.0+ brings about 30% improvement in app bundles due to how it generates the AoT code.

Understand Observables from RxJS.

A LOT of Angular work is about understanding what is ***Observable***. It's very important to understand how Observables work and becoming comfortable with RxJS library which helps you become Observable Ninja.

Lazy Loading the non-first-page routes

Lazy-load every route which don't need at 1st page hit. Webpack2 `import()` function will come handy for you. Also webpack's ng-router-loader will help automate the bundle creation for each lazy loaded module automatically..

Using Widgets and Libraries

Consider using standard widget library
like PrimeNG or ValorSoft. Try to avoid JQuery as it cannot be tree-shaken.

Debugging

Make use of `ng.probe()` in chrome console to do effective debugging / Or make use of Augury chrome extension—which wraps ng.probe for you.

Stay safe in Dark Corners of NgZone

NgZone and ZoneJS are some of the dark corners of Angular. When things don’t work fine even after you trying 100 different things for many days, you might be up against these two adversaries. I call them dark corners because no error will ever tell you that that error can be fixed if you fiddle around with NgZone. You must correlate your error and potential NgZone conflict yourself □. As such, NgZone is quite easy to use but I did not even know it existed for almost 5 months in my project.

Other wins via code structuring

> **Shared Modules**—Try to make use of shared module. Create a module and that should import & export all the commonly used modules & providers and import this in other modules.

> **Global vs local CSS**—Whenever writing the CSS, try to visualize if this kind of element might be used at lot of places and then write the style at application level instead of component, it will avoid the re-writing it again in new component. You can just override any small change is required in component level.

Theme File with SCSS—When using any CSS preprocessor, always define a file which has variables only related to color, font-size, etc. of the applications, it will help when you need to change the theme.

Typescript Inheritance for your help—Try to utilize the Inheritance in Typescript. If you have some view related functionality that might be required in many screens, you can create a base component with common functionality and then all other components can just extend it.

Use Services—Strive for complete segregation between the View implementation and service call. In the UI component, keep code only related to view, and delegate to a service to make the backend calls and for any functional logic.

General Web Developer Productivity Best Practices

Improve development workflow—Often times, developer do not think about finding shortcuts to improve their developer productivity. This includes, circumventing login locally, caching backend calls which are not required for your current work, making small code fixes to skip through 10 screens / clicks to arrive at the screen where they need to do their change. One should spend half an hour to twaek these things reach their current screen instantly and save time on every minor code update.

Mandatory Human Code Review—We have mandated that all developers must deliver code as pull_request in Git. Architect would review and approve the code before merging. This ensures that each line of code has been reviewed before merging. This help catching bugs and quality / performance problems which cannot be caught using Linters.

Design narrative:

One of the best thing that we implemented was—process of design elaboration from each developer for their story.

I insist that my developers follow this narrative.

To deliver story xyz,

Which component(s) would be required to be “created” or “modified”.

How will the component be accessed? From a topNav? Routing? From some user interaction on other components?

Which folder would those components belong?

What kind of @input, @output would be provided to / emitted from these components.

What would be your backend call requirement and it’s sequence

Any form validations?

Any spl technical things / libs required? E.g. moment, datepicker, modal, etc.

“Productivity improvement”? How will you reach your page fast—hardcoding? Proxying?

I have found that the developers were far more confident and their code quality improved once we established above design documentation process. Hopefully, you will find similar change in your team quality as well.

This post is adapted from my original post on my blog.

That’s it folks for now. Thank you for patiently reading till the end! If you liked the story—please follow me on twitter and hit ❤ symbol below the story.