Coding Standards Document

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - **Martin Golding**

# Angular

## Angular Application Folder Structure

Official Angular Style guide for best practices and Naming conventions for creating components, services, guards, modules, directives, pipes, etc.

<https://angular.io/guide/styleguide>

## Best Practices

1. Use Angular CLI for structured project folder and for creating components, services, module, guards, etc.
2. Prevent Memory Leaks in Angular Observable (Use ngOnDestroy to unsubscribe from observables)
3. Avoid having subscriptions inside subscriptions – I,e Avoid Nested Subscriptions/callback hell
4. Do not ever use Impure pipe.
5. Lazy load feature modules for faster loading application.
6. If the values of the variables are intact, declare it with ‘const’.
7. CDK Virtual Scroll – to load large data set with better performance.
8. Create Reusable components for frequently used layouts ex, modals, page title/app title
9. Write Readable Code - Variable- and function names
10. Write Small pure functions – Easy to write Unit Tests.
11. Remove unused code, avoid code comments (Add comments only if necessary)
12. Using Immutability - Objects and arrays are the reference types in JavaScript. If we want to copy them into another object or an array and to modify them, the best practice is to do that in an immutable way.
13. Avoid logic in templates as it will be difficult to Unit Tests.
14. Use Template Driven Forms for Small Size forms.
15. Use Reactive Forms – If form is big, needs dynamic field population, cross field validation (Reactive forms are easy to Unit Test)
16. Understand the Difference: Smart vs Dumb Components:

Smart components are used in manipulating data, calling the APIs, focusing more on functionalities, and managing states.

Smart components focus on how things work.

Dumb components are all about cosmetics: they focus more on how they look.

Dumb components are UI focused and more into reusability.

Dumb components help accept properties and to allow them to be dynamic and reusable as much as possible.

1. Use of Service Workers for the Fastest Responsive Apps:

Service workers make it even easier for users to interact with web apps. They convert your single-page apps into progressive web apps by adding native features to it. You can use service workers with any version of the Angular framework above five.

They augment/enhance user experience by allowing users to access and interact with native features of a web app without having to wait for internet connectivity.

**Advantages of Service Workers:**

* Offers native app experience, completely caches the version on refreshing the web pages.
* Loads latest cached code in the new tabs.
* Downloads the resources only upon the changed contents.
* Serves the previous version of application until the stable version is being served.

1. Use Async Pipes for Saving the Memory Bytes:

Async Pipes are built-in attributes. They save you a boatload of memory bytes when you build a large-scale application frequented by thousands of users. They subscribe to the observables and return the value it has omitted.

1. Security Best Practices in Angular apps:

it’s not about application-level security, but the built-in features that the framework offers. Let’s have a quick look into the security aspects in Angular.

* Sanitization and security contexts.
* DOM sanitization service.
* Context security policy.
* Offline template compiler.

1. Use RxJS

RxJS library is a powerful tool that helps to optimal application logic and keeps code clean.

1. Testing

Testing enables us to make sure that certain parts of the application work exactly as you expect them to. This to some extent saves the existing code from breakdowns, helps to clarify – how it will work in various cases. And, in the end, allows detecting code weaknesses.

# TypeScript

## Best Practices

1. Don’t ever use the types Number, String, Boolean, Symbol, or Object These types refer to non-primitive boxed objects that are almost never used appropriately in JavaScript code.
2. Do use the types number, string, boolean, and symbol.
3. Instead of Object, use the non-primitive object type.
4. Declare Variable Type Instead of Using ‘any’.
5. Use Optional Parameters - Don’t write several overloads that differ only in trailing parameters.
6. Do use optional parameters whenever possible.

# C#

## Best Practices

1. **Naming Conventions**

**1. Pascal Case  
Use pascal Case for Naming Classes/Interfaces, Class Properties, Class Methods, Enums, Constants.**

**2. Camel Case**Use camel case for variables, method parameters.

1. **Always begin Interface Name with letter ‘I’.**
2. **Keep Methods short and Use Single Responsibility Design Principle.**
3. **Avoid too many parameters for method.**
4. **Use Switch when using if-else-if-elseif statement for better readability.**
5. **Use Enums/Constants to avoid hardcoding strings or numbers.**
6. **Write only one statement and declaration per line.**
7. **Use regions to group related pieces of code.**
8. **Use Meaningful names for Class, Interface, Property, Method, Variables and avoid comments as much as possible.**
9. **Always check for objects and complex objects for null values before accessing any of its property.**
10. To check for null or empty conditions, use the following:

**//Avoid**

**var varName = "test"; if (varName != null && varName != "") { }**

**//Do**

**var varName = "test"; if (!string.IsNullOrEmpty(varName)) { }**

1. **Collections**
2. **Return empty collection instead of null.**
3. **Check Any() operator instead of checking count > 0 or checking null.**
4. **Use foreach instead of traditional for loop.**
5. **Use IList<T>, ICollection<T>, IEnumerable<T> instead of concreate classes like List<T>.**
6. **Use object initializers to simplify the object creation.**
7. **Group the name spaces I,e put framework namespaces first and then application specific namespaces.**
8. **Do ensure to close connections like Database, file streams, sockets etc.**
9. **Use using statement to automatic garbage collection instead of try-catch-finally.**
10. **Use StringBuilder class instead of string when manipulating strings in loop.**

# SQL

## Best Practices

## Database Objects

1. Use well defined and consistent names for tables and columns.
2. Use singular for table names. Table represents a collection of entities, there is no need for plural names.
3. Don’t use spaces for table names. Otherwise you will have to use ‘{‘, ‘[‘, ‘“’ etc. characters to define tables (i.e. for accessing table Student Course you'll write “Student Course”. StudentCourse is much better).
4. Don’t use unnecessary prefixes or suffixes for table names.
5. Use integer id fields for all tables. If id is not required for the time being, it may be required in the future (for association tables, indexing ...).
6. Choose columns with the integer data type (or its variants) for indexing. varchar column indexing will cause performance problems.
7. Use bit fields for boolean values. Using integer or varchar is unnecessarily storage consuming. Also start those column names with “Is”.
8. Provide authentication for database access. Don’t give admin role to each user.
9. Use an ORM (object relational mapping) framework (i.e. hibernate, iBatis ...) if application code is big enough. Performance issues of ORM frameworks can be handled by detailed configuration parameters.
10. Partition big and unused/rarely used tables/table parts to different physical storages for better query performance.
11. For big, sensitive and mission critical database systems, use disaster recovery and security services like failover clustering, auto backups, replication etc.
12. Use constraints (foreign key, check, not null ...) for data integrity. Don’t give whole control to application code.
13. Lack of database documentation is evil. Document your database design with ER schemas and instructions. Also write comment lines for your triggers, stored procedures, and other scripts.
14. Use indexes for frequently used queries on big tables. Analyzer tools can be used to determine where indexes will be defined. For queries retrieving a range of rows, clustered indexes are usually better. For point queries, non-clustered indexes are usually better.
15. Database server and the web server must be placed in different machines. This will provide more security (attackers can’t access data directly) and server CPU and memory performance will be better because of reduced request number and process usage.
16. End foreign key column names with suffix ‘Ref’.

## Queries

1. Do not use SELECT \* FROM TABLE/VIEW, instead select the columns that are only needed for better performance.
2. Use table aliases when your query involves more than one table/view.
3. Use column names in order by clause instead of its number.
4. Use ANSI standard Join clauses instead of old-style joins.

//Old Style

SELECT E.NAME, D.DEPT\_NAME FROM EMPLOYEE E, DEPT D WHERE E.DEPT\_ID = D.ID;

//ANSI standard

SELECT E.NAME, D.DEPT\_NAME FROM EMPLOYEE E INNER JOIN DEPT D ON E.DEPT\_ID = D.ID;

1. Use columns list in your insert statements.
2. Always check for existence before insert statement (Data Scripts).
3. Always use Uppercase letter for writing queries.
4. Try using bulk insert/update/delete, database engine performance is always better with bulk operations.
5. Unique ID
6. Multi Column primary key – Pros: Potentially fast, saves memory, Cons: Complex, requires upfront design and to ensure better performance, design will not be consistent across database.
7. Single Column Unique ID (Identity/Sequences) – Pros: Easy choice, Good performance, spreads load well.

Cons: Memory wastage, require other columns to be useful in queries.

1. Use non clustered indexes to improve query performance however it comes with cost when doing insert/delete as indexes needs to update.
2. Image and blob data columns must not be defined in frequently queried tables because of performance issues. These data must be placed in separate tables and their pointer can be used in queried tables.
3. Normalization must be used as required, to optimize the performance. Under-normalization will cause excessive repetition of data, over-normalization will cause excessive joins across too many tables. Both will get worse performance.

# HTML

## Best Practices

1. Use proper document structure:

HTML documents will still work without elements such as <html>, <head>, and <body>. However, the pages will not render correctly in every browser so it's important to be consistent using the proper document structure.

1. Always declare document type.
2. Use meaningful title tag.
3. Use descriptive meta tags: this will be used by Search Engine spiders.
4. Use Semantic Elements:

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

1. Separate Content from Presentation: Your HTML is your content. CSS provides your content’s visual presentation. Never mix both.
2. Don't use inline styles.

It may seem like an easy route to place styling in line with the code instead of creating an external style sheet. However, inline styles are not a good coding practice because it makes it harder to update and maintain a website. Instead, keep your styles separate from your HTML mark-up.

1. Place external style sheets within the <head> tag.

Although external style sheets can be placed anywhere in the HTML document, it is best practice to place them within the <head> tag. This will allow your page to load faster.

1. Minify and Unify CSS.
2. Use the Right HTML Element at the Right Place.
3. Don’t Use Divs for Everything.
4. Use an Unordered List (<ul>) for Navigation.
5. Close Your Tags.
6. Use Alt Attributes with Images.
7. Use Title Attributes with Links (When Needed).
8. Use lowercase markup.
9. Reduce the number of elements on a page.

# CSS

## Best Practices

1. Make it Readable

Great readability of your CSS makes it much easier to maintain in the future, as you'll be able to find elements quicker.

Organize Code with Comments.

Write CSS Using Multiple Lines & Spaces.

Build Proficient Selectors.

1. Keep it Consistent.
2. Use a Reset

Most CSS frameworks have a reset built-in, but if you're not going to use one then at least consider using a reset. Resets essentially eliminate browser inconsistencies such as heights, font sizes, margins, headings, etc. The reset allows your layout look consistent in all browsers.

1. Organize the Stylesheet with a Top-down Structure.
2. Combine Elements

Elements in a stylesheet sometimes share properties. Instead of re-writing previous code, why not just combine them? For example, your h1, h2, and h3 elements might all share the same font and color:

h1, h2, h3 {font-family: tahoma, color: #333}

We could add unique characteristics to each of these header styles if we wanted (ie. h1 {size: 2.1em}) later in the stylesheet.

1. Create Your HTML First.
2. Understand the Difference Between Block vs. Inline Elements.
3. Make Use of Generic Classes

.left {float:left}

.right {float:right}

<div id="coolbox" class="left">...</div>

This way you don't have to constantly add "float:left" to all the elements that need to be floated.

1. Use Text-transform

Text-transform is a highly-useful CSS property that allows you to "standardize" how text is formatted on your site. For example, say you're wanting to create some headers that only have lowercase letters. Just add the text-transform property to the header style like so:

text-transform: lowercase;

Now all of the letters in the header will be lowercase by default. Text-transform allows you to modify your text (first letter capitalized, all letters capitalized, or all lowercase) with a simple property.

1. Avoid Extra Selectors.
2. Use ‘Hex’ Code Instead of Using Name ‘Color’ for Color property.
3. Use CSS Vendor Prefixed (when required).
4. Avoid inline styling.
5. Shrink CSS file size with CSS Compressors.
6. Use Hyphens for IDs or Class Names

BAD PRACTICE:

. left\_col {margin: 0; padding: 0;}

BEST PRACTICE:

. left-col {margin: 0; padding: 0;}

1. Use Practical ID & Class Values.