Best Practices

1. File Structure Convention

The guideline uses the shortcut **employee.component.ts|html|css|spec** to represent those various files. Using this shortcut makes this guide's file structures easier to read and more terse.

1. Single responsibility

Apply the single responsibility principle (SRP) to all components, services, and other symbols. This helps make the app cleaner, easier to read and maintain, and more testable.

* Do define one thing, such as a service or component, per file.
* Consider limiting files to 400 lines of code.
* The key is to make the code more reusable, easier to read, and less mistake prone.

1. Small functions

* Consider limiting to no more than 75 lines.
* Small functions are easier to test, especially when they do one thing and serve one purpose.
* Small functions help avoid hidden bugs that come with large functions that share variables with external scope, create unwanted closures, or unwanted coupling with dependencies.

1. Service Names

* Do use consistent names for all services named after their feature.
* Do suffix a service class name with Service. For example, something that gets data or heroes should be called a DataService or a HeroService.

1. Coding Convention
   1. Classes
      1. Do use upper camel case when naming classes.
   2. Constant
      1. Do declare variables with const if their values should not change during the application lifetime.
      2. Conveys to readers that the value is invariant.
      3. TypeScript helps enforce that intent by requiring immediate initialization and by preventing subsequent re-assignment.
      4. Consider spelling const variables in lower camel case.
   3. Interfaces
      1. Do name an interface using upper camel case.
      2. Consider naming an interface without an I prefix.
      3. Consider using a class instead of an interface.
      4. A class alone is less code than a *class-plus-interface*.
   4. Properties and methods
      1. Do use lower camel case to name properties and methods.
      2. Avoid prefixing private properties and methods with an underscore.
   5. Import Line Spacing
      1. Consider leaving one empty line between third party imports and application imports.
      2. Consider listing import lines alphabetized by the module.
      3. The empty line separates *your* stuff from *their* stuff.
      4. Alphabetizing makes it easier to read and locate symbols.
2. Application structure and NgModules

LIFT

* Do structure the app such that you can Locate code quickly, Identify the code at a glance, keep the Flattest structure you can, and Try to be DRY.
* LIFT Provides a consistent structure that scales well, is modular, and makes it easier to increase developer efficiency by finding code quickly. To confirm your intuition about a particular structure, ask: *can I quickly open and start work in all of the related files for this feature*?
  1. Locate
     1. Do make locating code intuitive, simple and fast.
     2. To work efficiently you must be able to find files quickly, especially when you do not know the file names.
     3. Keeping related files near each other in an intuitive location saves time. A descriptive folder structure makes a world of difference to you and the people who come after you.
  2. Identity
     1. Do name the file such that you instantly know what it contains and represents.
     2. Do be descriptive with file names and keep the contents of the file to exactly one component.
     3. Avoid files with multiple components, multiple services, or a mixture.
     4. Spend less time hunting and pecking for code, and become more efficient. Longer file names are far better than *short-but-obscure* abbreviated names.
  3. Flat
     1. Do keep a flat folder structure as long as possible.
     2. Consider creating sub-folders when a folder reaches seven or more files.
     3. No one wants to search for a file through seven levels of folders. A flat structure is easy to scan.
  4. T-DRY (Try to be DRY)
     1. Do be DRY (Don't Repeat Yourself).
     2. Avoid being so DRY that you sacrifice readability.
     3. Being DRY is important, but not crucial if it sacrifices the other elements of LIFT. That's why it's called *T-DRY*.