Angular Fundamentals

.NET CORE

Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps.

HTTPS://ANGULAR.IO/DOCS

TS/Angular Workspace SetUp

https://angular.io/guide/setup-local
https://code.visualstudio.com/docs/typescript/typescript-compiling
https://angular.io/tutorial/toh-pt0#create-a-new-workspace-and-an-initial-application

---

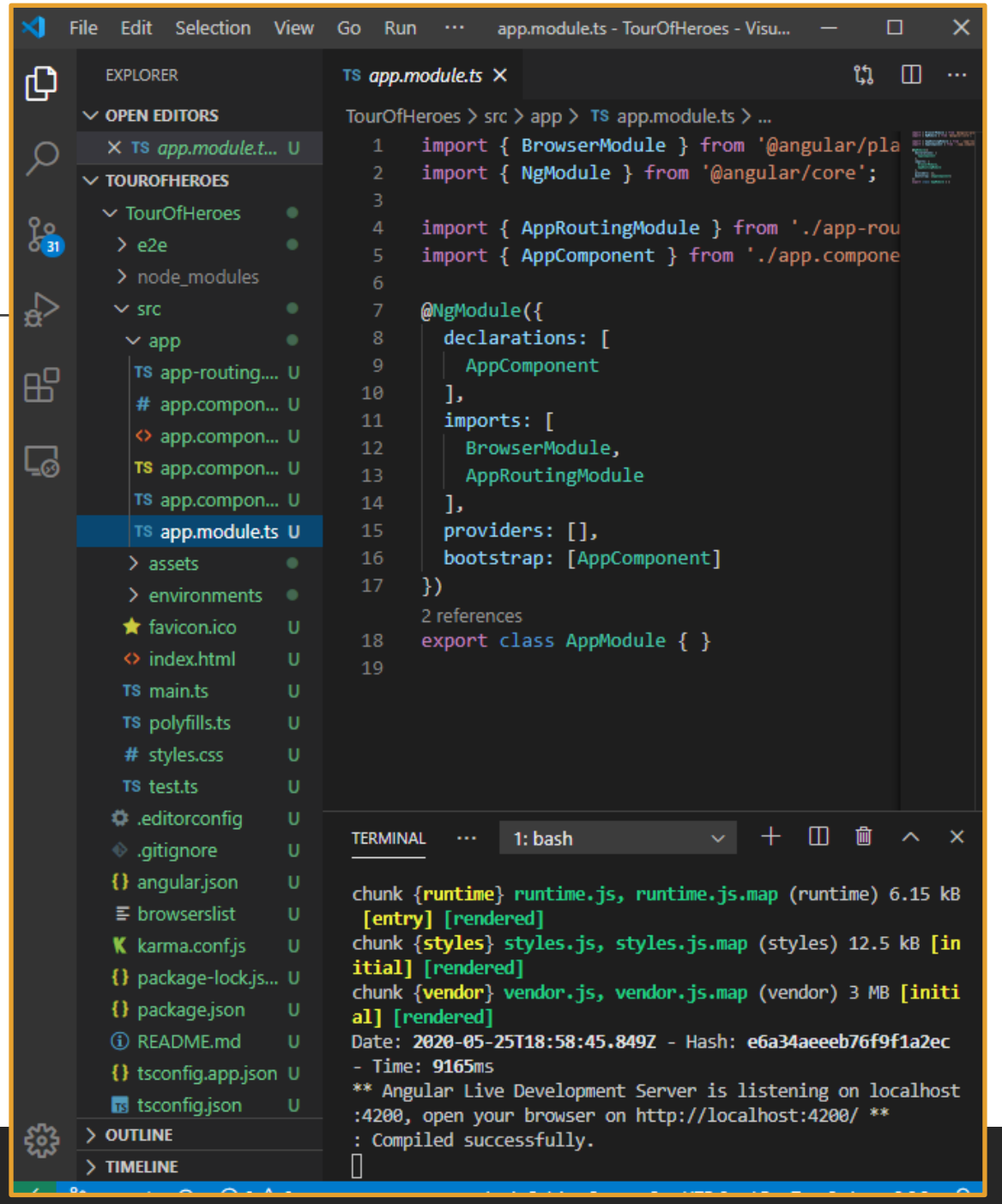
Following the steps from here to create your first Angular App.

1. Make sure you have Node.js with node –v in Command Line. If not, go to nodejs.org to get it.
2. Install Angular CLI globally with npm install -g @angular/cli in Command Line.
3. Use ng new <my-app-name> to create a ***WorkSpace*** for your app and install the default starter app.
4. Press enter to accept the defaults.
5. ng new installs the Angular ***npm*** packages needed.
6. Navigate in the CLI to your app folder. (cd <my-app-name>).
7. Use ng serve –open (2 dashes) to launch the server and open the browser with the default sample project.
8. In VS Code, install the ***Angular Extension Pack*** to get goodies!
9. Use this Angular Cheat Sheet for quick reference!

WorkSpace

https://angular.io/tutorial/toh-pt0#set-up-your-environment

A workspace contains all the files for one or more projects. A project is the set of files that comprise an app, a library, or end-to-end (e2e) tests.



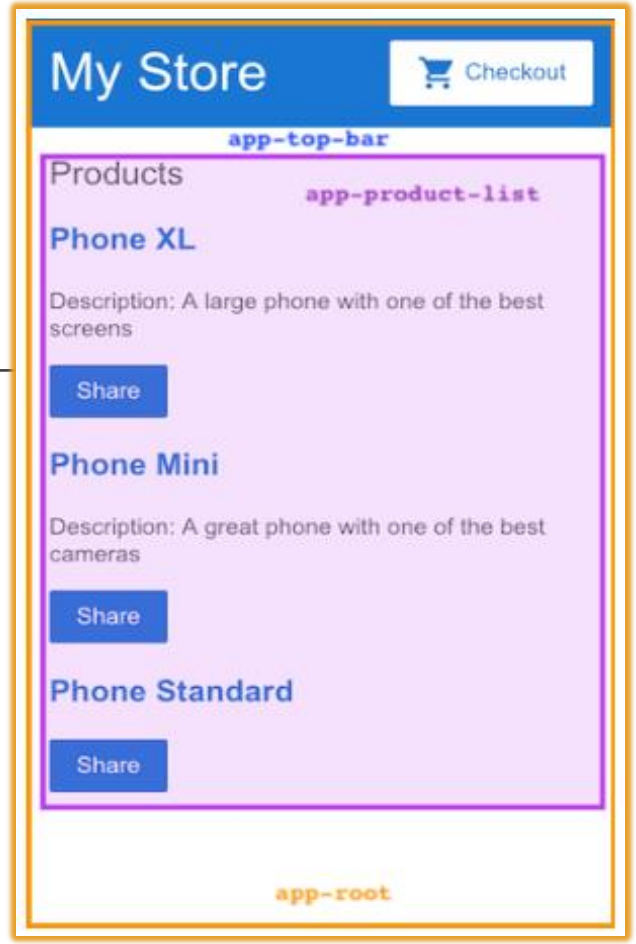
Components

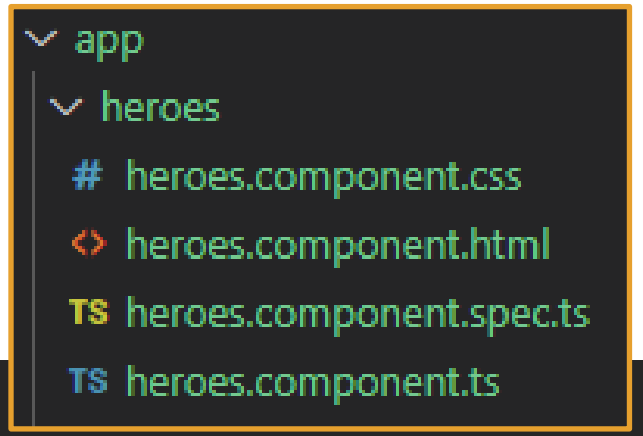
https://angular.io/tutorial/toh-pt0#set-up-your-environment
https://angular.io/guide/component-interaction

Components are the fundamental building blocks of ***Angular*** applications. They display data on the screen, listen for user input, and take action based on that input.

An ***Angular*** application comprises a tree of ***components***, in which each ***Angular component*** has a specific purpose and responsibility. In this example there are 3 components displayed:

- ***app-root*** (orange box) is the application shell. This is the first component to load and the parent of all other components. You can think of it as the base page.
- ***app-top-bar*** (blue background) is the store name and checkout button.
- ***app-product-list*** (purple box) is the product list.





Angular Component

https://angular.io/tutorial/toh-pt1#create-the-heroes-component

The *CLI* creates a new folder for each ***component*** and generates a .css, .ts, and .html, inside it. Use either the Angular helper (R-click the app folder) or the command ng generate component [name] to create a new ***component***.

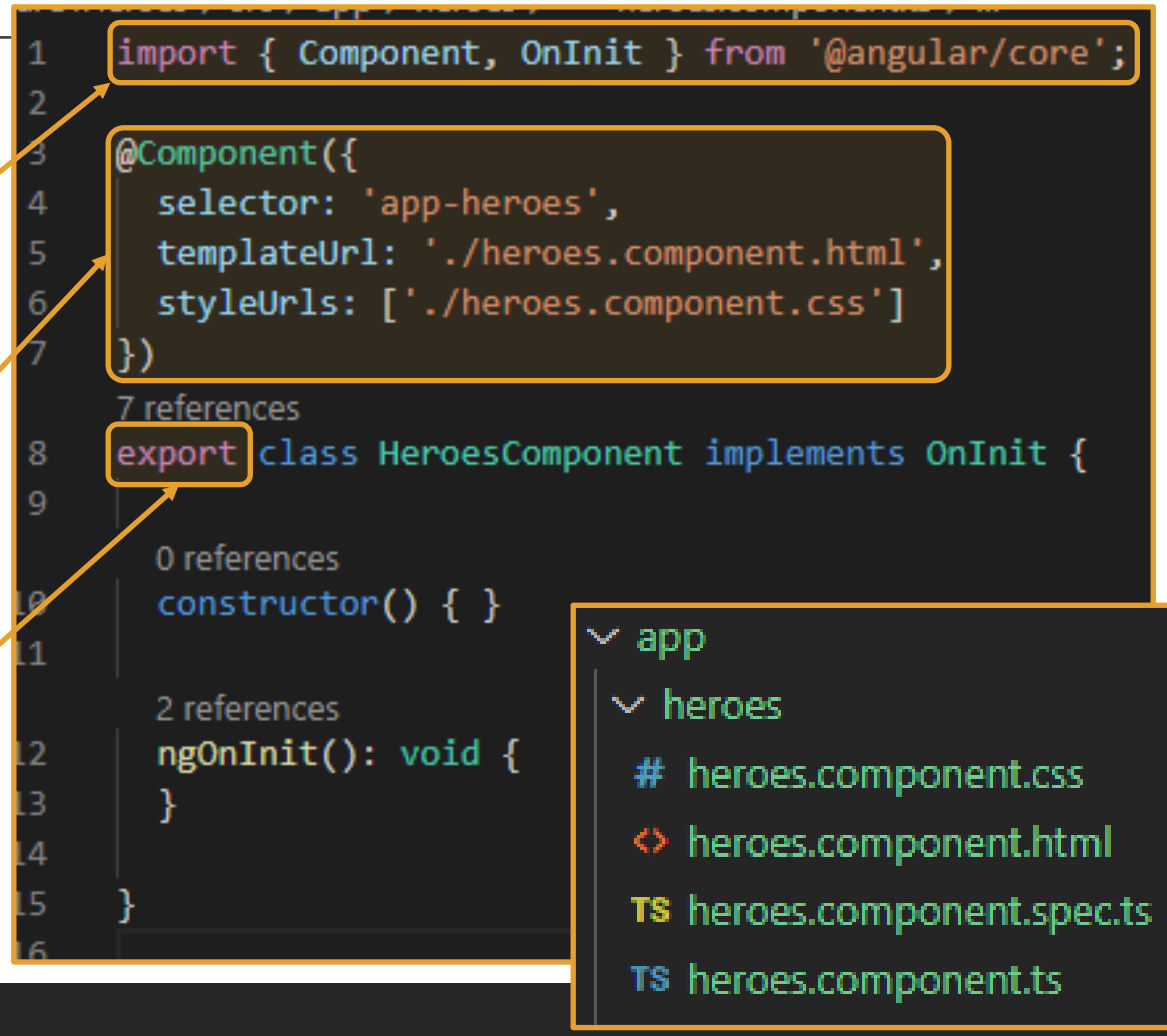
Always import { Component, OnInit } from @angular/core; library and annotate the ***component class*** with @Component().

@Component is a ***decorator*** function that specifies the Angular metadata for the ***component***:

1. The selector name to use for CSS and if importing this component into a .html page.
2. The relative .html location.
3. The relative .css location.

Use export to make the class available for import by other components.

ngOnInit() is a lifecycle hook. It's a good place for component initialization logic like getting current data from a ***Service***.



Connect a new Component

https://angular.io/tutorial/toh-pt1#show-the-heroescomponent-view

Every ***component*** must be declared in the ***NgModule*** to function.

When you declare a new ***component***, ***Angular CLI*** automatically imports the new component into ***app.module.ts*** and declares it under the @NgModule.declarations array on generation.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms'; // <-- NgModel lives here

import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';

@NgModule({
  declarations: [
    AppComponent,
    HeroesComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Create an interface

https://angular.io/tutorial/toh-pt1#create-a-hero-interface

Interfaces are useful for when you want to define a class or object (with its types), then import it in various comonents.

Create an ***interface*** with ng generate interface [name], or R-click the app folder => choose another schematic.

Then import that ***interface*** into the ***Component*** from the relative file location in which you want to use it.

src/app/hero.ts

```
export interface Hero {
  id: number;
  name: string;
}
```

```
1  import { Component, OnInit } from '@angular/core';
2  import { Hero } from '../hero';
```

TypeScript Modules

https://www.typescriptlang.org/docs/handbook/modules.html

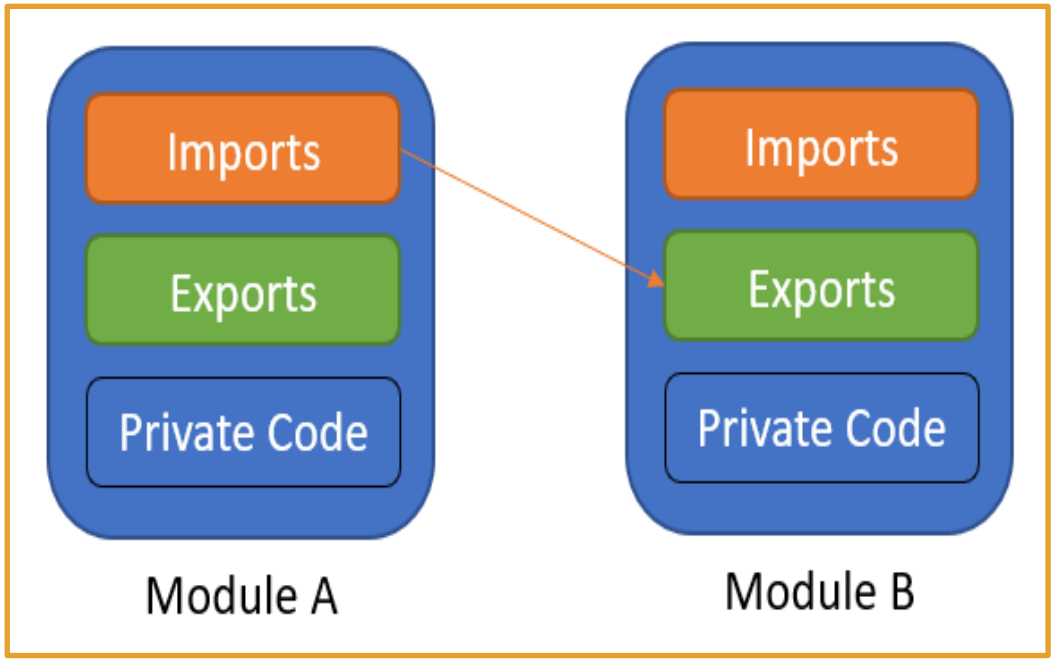
TS shares the *JS* concept of ***Modules***.

Modules in *TS* have their own scope. Anything declared inside a ***module*** is not visible outside that ***module*** unless it is explicitly ***exported***.

To consume a property ***exported*** from a different ***module***, it must be ***imported*** using an ***import*** method.

The relationships between ***modules*** are specified in terms of ***imports*** and ***exports*** at the file level.

In *TS*, any file containing a top-level ***import*** or ***export*** is considered a ***module***. A file without any top-level ***import*** or ***export*** declarations is treated as a script whose contents are available in the global scope (and therefore in ***modules*** as well).



TypeScript - Exporting a Declaration

https://www.typescriptlang.org/docs/handbook/modules.html#export

Any declaration (variable, function, class, type alias, interface) can be ***exported*** by adding the ***export*** keyword before the type keyword.

1. Use the ***export*** keyword to make a class, function, or variable available to other ***modules*** from within the ***module*** (***component***).
2. ***Import*** the class, function, or variable into the ***module*** (***component***) where you want to implement it.

```
export interface StringValidator {
  isAcceptable(s: string): boolean;
}
```

```
import { StringValidator } from "./StringValidator";

export const numberRegexp = /^[0-9]+$/;

export class ZipCodeValidator implements StringValidator {
  isAcceptable(s: string) {
    return s.length === 5 && numberRegexp.test(s);
  }
}
```

TS Function Parameter Types

https://www.typescriptlang.org/docs/handbook/functions.html#optional-and-default-parameters

- In *TS*, every function parameter is assumed to be ***required*** by the function.
- Make a parameter ***optional*** by placing a '?' behind the parameter name.
- ***Optional*** parameters must be last.
- Give parameters ***default*** values with '= "value"'.
- When the ***default*** parameter comes last, it is treated as ***optional***.
- ***Rest*** Parameters in *TS* are like ***args*** parameters in *JS*.
- ***Rest*** parameters are treated as ***optional*** parameters. The compiler builds an array of the additional arguments passed with the name given after the ellipsis (...). The ellipsis is also used to declare the type of the ***Rest*** parameters.

```
function buildName(firstName: string, lastName?: string) {
  if (lastName) return firstName + " " + lastName;
  else return firstName;
}
```

Optional parameters

```
function buildName(firstName: string, lastName = "Smith") {
  return firstName + " " + lastName;
}
```

Default parameters

```
function buildName(firstName: string, ...restOfName: string[]) {
  return firstName + " " + restOfName.join(" ");
}
```

Rest parameters

Dependency Injection – Services and Injectables

https://angular.io/guide/glossary#dependency-injection-di
https://angular.io/guide/dependency-injection

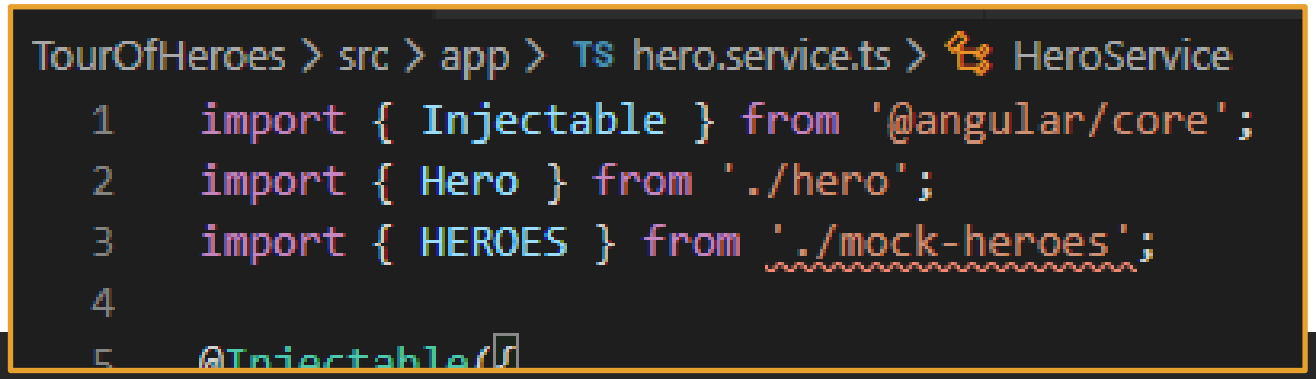
Components shouldn't fetch or save data directly. They should delegate data access to a ***Service***. A ***Service*** can get data from anywhere—a web service, local storage, or a mock data source.

Services are an integral part of Angular applications. In Angular, a ***service*** is an instance of a class that you can make available to any part of your application using Angular's ***dependency injection*** system.

Services are the place where you share data between parts of your application. The ***Service*** is your portal to persist data and have methods to access that data.

You can use ***services*** to share data across ***components.***

The @Injectable() decorator accepts a metadata object for the service, the same way the @Component() decorator does for component classes.



Dependency Injection – Services and Injectables

https://angular.io/tutorial/toh-pt4#provide-the-heroservice
https://angular.io/guide/dependency-injection

You must make the ***Service*** available to the ***dependency injection system*** before ***Angular*** can inject it into the ***Component*** by registering a ***provider***.

By default, ***the Angular CLI*** command ng generate service registers a ***provider*** with the ***root*** injector for your ***Service*** by including ***provider*** metadata that's provided in: 'root' in the @Injectable() ***decorator*** of the ***Service Component***.

When a ***Service*** is provided at the root level, Angular creates a single, shared instance of the ***Service*** and injects it into any class that asks for it.

Angular will also remove any unused ***Services***.

```
1   import { Injectable } from
2   import { Hero } from './her
3   import { HEROES } from './m
4
5   @Injectable({
6     providedIn: 'root'
7   })
    3 references
8   export class HeroService {
9
      0 references
10    getHeroes(): Hero[] {
11      return HEROES;
12    }
      0 references
```

Angular – How to Use DI to Get a Service

https://angular.io/tutorial/toh-pt4

To create a service to access your stored data,

1. Create a ***Service*** with
   - ng generate service [serviceName].
2. Import the ***Injectable*** symbol into the ***Service component***. This allows the ***Service*** to be injected into any other ***Component.***
   - import { Injectable } from '@angular/core';
3. Import the ***Service*** into the ***Component*** where it will be used with
   - import { ServiceName } from '../relative.location';.
4. Inject the ***Service*** into the constructor of the ***Component*** where it will be used with
   - constructor(private ServiceVariableName: ServiceName) {}.
5. Now you can access the ***Services*** functions with dot notation!

```
2   import { Hero } from '../hero';
3   import { HeroService } from '../hero.service';
4
```

```
0 references | 1 reference
constructor(private heroService: HeroService) {}

1 reference
getHeroes(): void {
  this.heroes = this.heroService.getHeroes();
}
6 references
ngOnInit(): void {
  this.getHeroes();
}
```

Best Practice is to use ngOnInit() to access and retrieve data from the service on instantiation of the Component instead of retrieving it in the constructor.