



Angular Fundamentals

.NET

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

[HTTPS://ANGULAR.IO/DOCS](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>

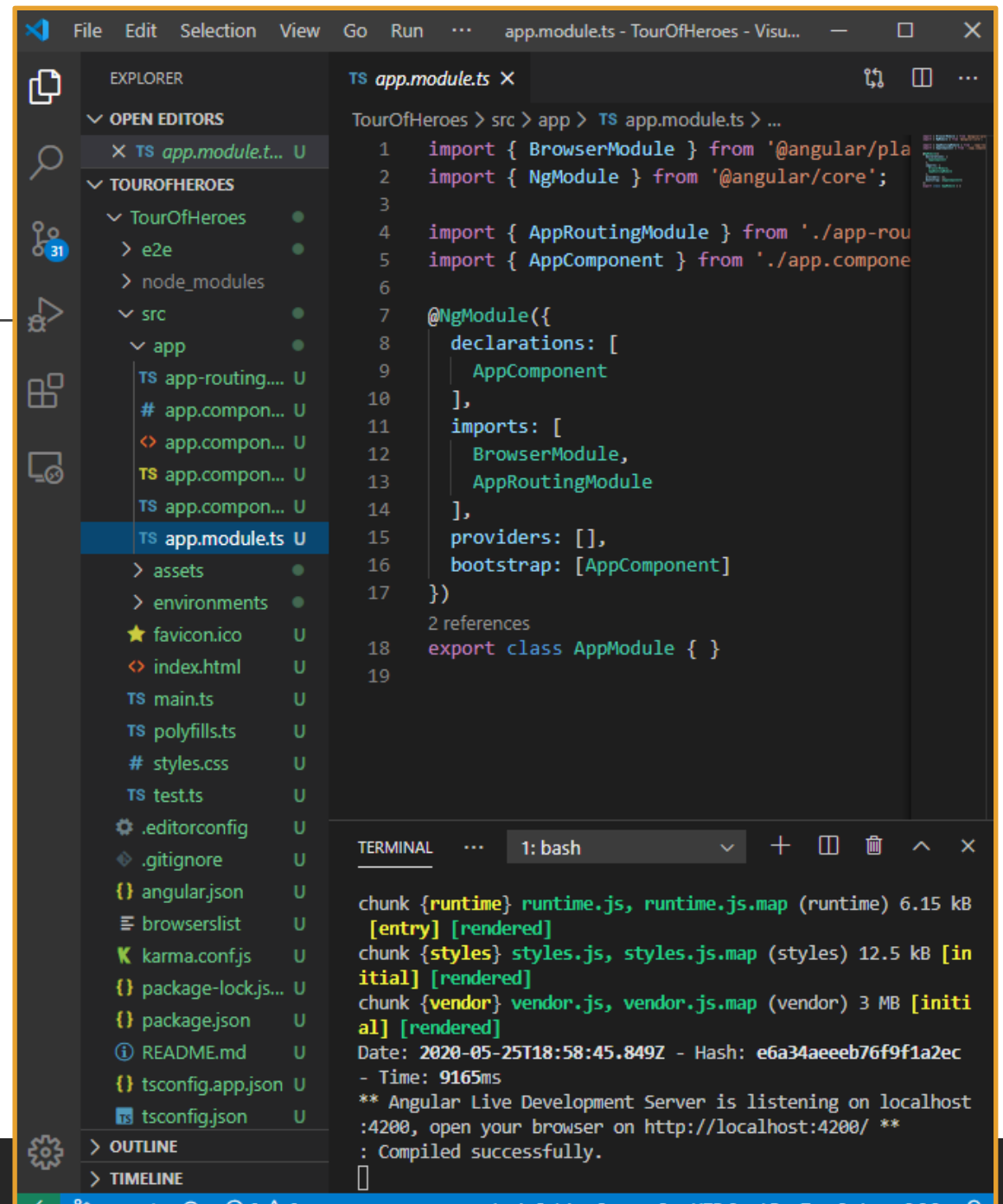
Following the steps from [here](#) to create your first Angular App.

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.



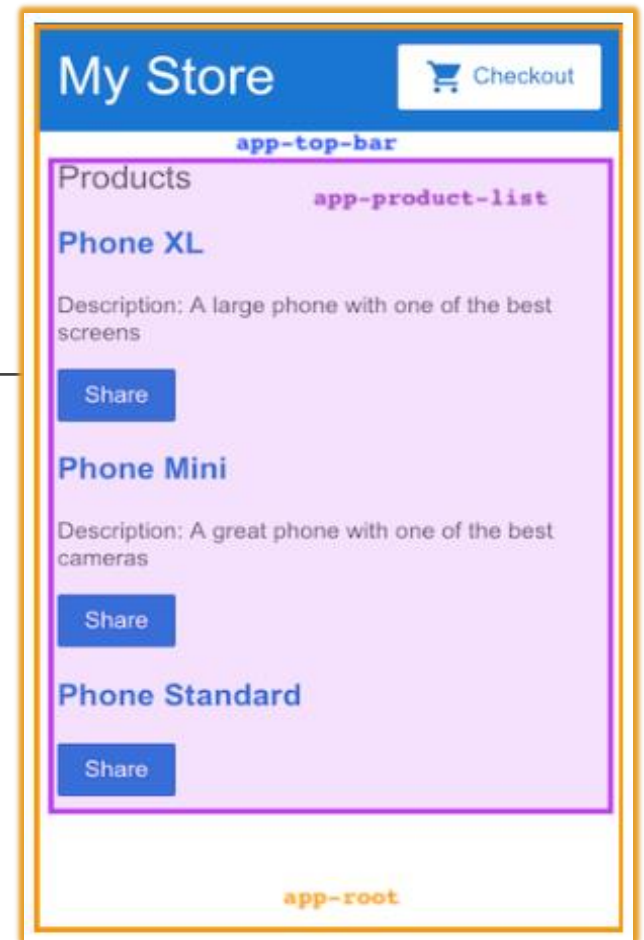
Components

<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.



```
✓ app
  ✓ heroes
    # heroes.component.css
    <> heroes.component.html
    TS heroes.component.spec.ts
    TS heroes.component.ts
```


Angular Component

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

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

Always `import { Component, OnInit } from @angular/core;` library.

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 the best place for `@Component` initialization logic, such as getting current data from a **Service** or initializing variables.

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-heroes',
5   templateUrl: './heroes.component.html',
6   styleUrls: ['./heroes.component.css']
7 })
8 export class HeroesComponent implements OnInit {
9
10  0 references
11  constructor() { }
12
13  2 references
14  ngOnInit(): void {
15  }
16 }
```

7 references

0 references

2 references

app

heroes

- # heroes.component.css
- heroes.component.html
- TS heroes.component.spec.ts
- TS heroes.component.ts

Connect a new Component

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

Every *component* must be declared in `@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.

```
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 into components where needed.

Create an **interface** with **ng generate interface <ComponentName>**.

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>

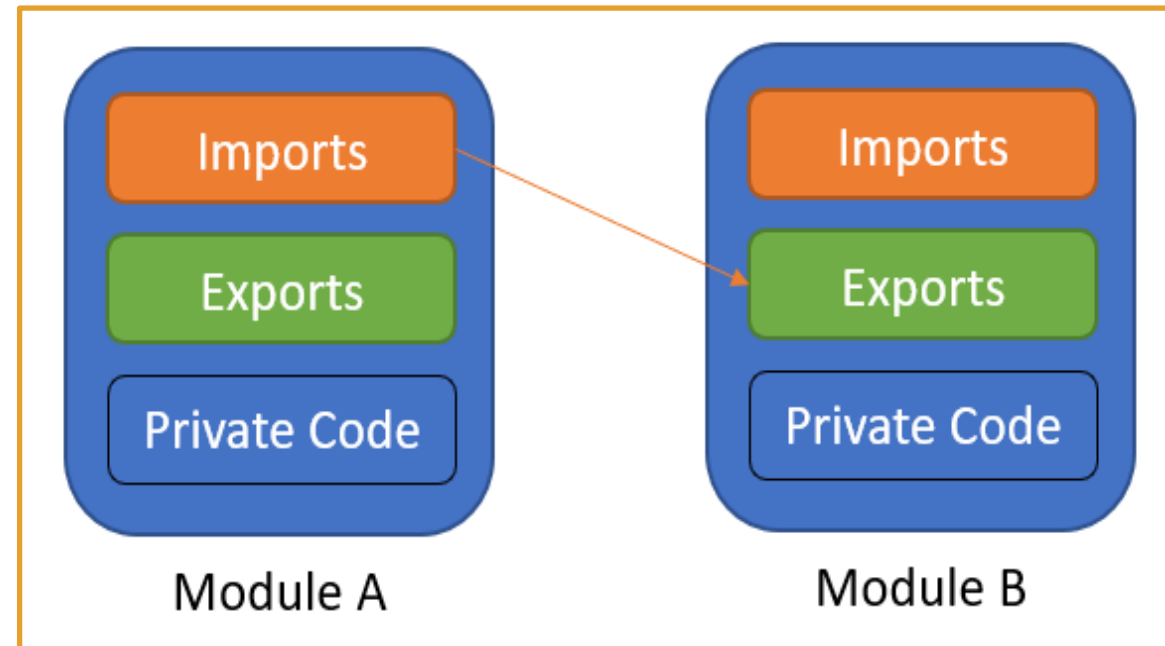
TS shares the *JS* concept of **Modules**. **Modules** in *TS* have their own scope. A module must be explicitly exported to make its members visible.

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 numberRegex = /^[0-9]+$/;  
  
export class ZipCodeValidator implements StringValidator {  
    isAcceptable(s: string) {  
        return s.length === 5 && numberRegex.test(s);  
    }  
}
```

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 integral to 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.

```
TourOfHeroes > src > app > TS hero.service.ts > HeroService
1  import { Injectable } from '@angular/core';
2  import { Hero } from './hero';
3  import { HEROES } from './mock-heroes';
4
5  @Injectable()
```

Dependency Injection – Services and Injectables

<https://angular.io/tutorial/toh-pt4#provide-the-heroservice>

<https://angular.io/guide/dependency-injection>

<https://angular.io/guide/architecture-services>

Providers of services must be registered with Angular's DI system before they can be injected into a **Component**.

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 './hero
3  import { HEROES } from './mo
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class HeroService {
9
10     0 references
11     getHeroes(): Hero[] {
12       return HEROES;
13     }
14
15     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**:
 - `ng generate service <serviceName>`.
2. Import the **Injectable** symbol into the **Service Component**
To allow the **Service** to be injected into **Components**:
 - `import { Injectable } from '@angular/core';`
3. Import the **Service** into the **Component** where it will be used:
 - `import { ServiceName } from '../relative.location';`
4. Inject the **Service** into the constructor of the **Component** where it will be used:
 - `constructor(private ServiceVariableName: ServiceName) {}`.

Use `ngOnInit()` to access and retrieve data from a service on instantiation of the **Component** instead of using the constructor.

```
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();
}
```

How to stop a running Angular Program

<https://anthonygiretti.com/2018/03/26/how-to-avoid-port-4200-is-already-in-use-error-with-angular-cli/>

1. In Command Line, use `netstat -ano | findstr :yourPortNumber`. (Usually it's 4200 with Angular.) to get your process number (PID). It's on the right or 'Listening'
2. In Command Line, use `taskkill [yourPID#]`.
3. In Command Line, use `ng serve --open` to recompile and reopen your app.