

---

## Angular Framework

---

### Introduction :

Angular is a Javascript-based open source front-end framework that allows to create the new generation web applications. These web application are reactive dynamic and responsive. Angular is a Single page application.

A **single-page application (SPA)** is a web application that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server. This approach avoids interruption of the user experience between successive pages, making the application behave more like a desktop application.

For example : Gmail, Google Maps, Facebook...

Angular is owned by google and maintained by the developer community to keep improving the SAPs development approach.

---

### History :

The first version of Angular was Angular1.0 called AngularJS which was released in 2010.

In 2014 Angular 2.0 was first introduced it was a complete rewrite of Angular so, the drastic changes in the 2.0 version created controversy among developers. After months of development beta testing and reviews. The final version was released on September 2016. Since this version we no longer call it AngularJS but Angular.

Angular 3.0 was just skipped

Angular 4 version was announced on 13 December 2016 and a stable version released on 23 March 2017.

Angular 5 was released on 1 Nov, 2017. It provided some improvements to support for progressive web apps, also provides improvements related to Material Design.

Angular 6 was released on 4 may, 2018. It was a major release which provides many new features.

Angular 7 was released on October 18, 2018.

Angular 8 was released on Mai 28, 2019.

---

## Angular 8

Angular is a framework which is used to build client applications in HTML and TypeScript.

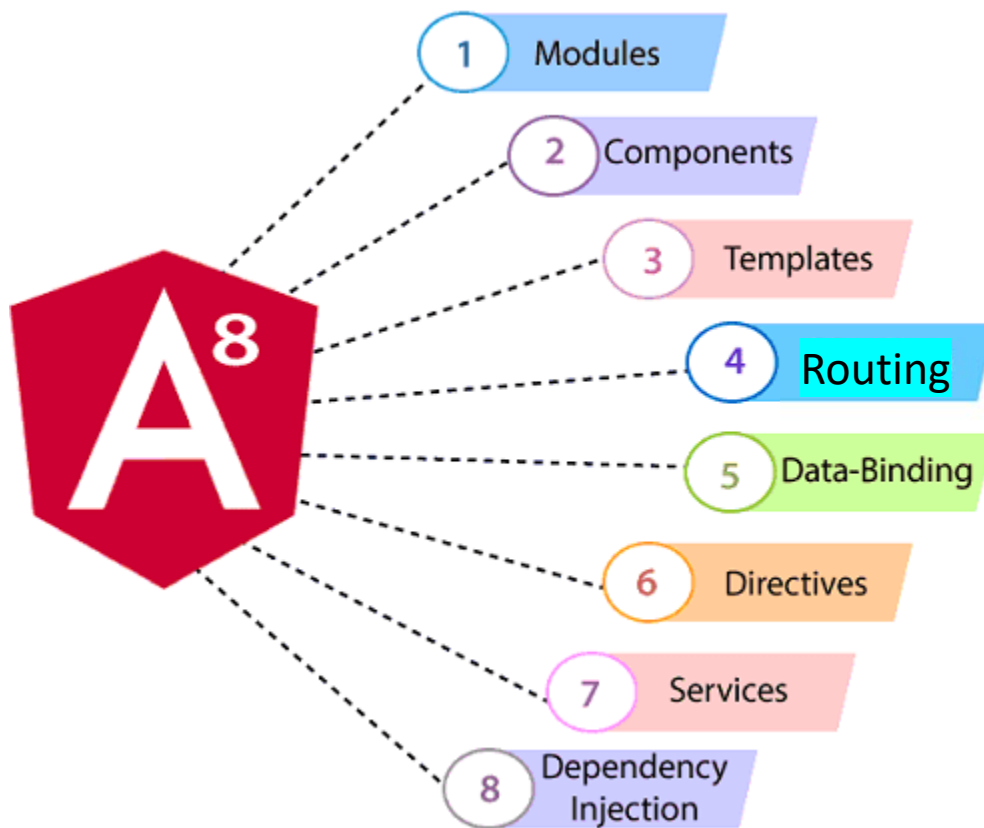
Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you can import into your apps.

**TypeScript** is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language.

TypeScript is designed for development of large applications and transcompiles to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs. TypeScript may be used to develop JavaScript applications for both client-side and server-side (Node.js) execution.

For this course we will be using angular version 8.

These are the different pillars of Angular that we will see them in current course, we will go back in details about every item in the coming lectures.



## Tools and dependencies

### Visual Studio code

Visual Studio Code is a lightweight and powerful source code editor available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go).

<https://code.visualstudio.com/>

### Node (npm : node package manager)

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

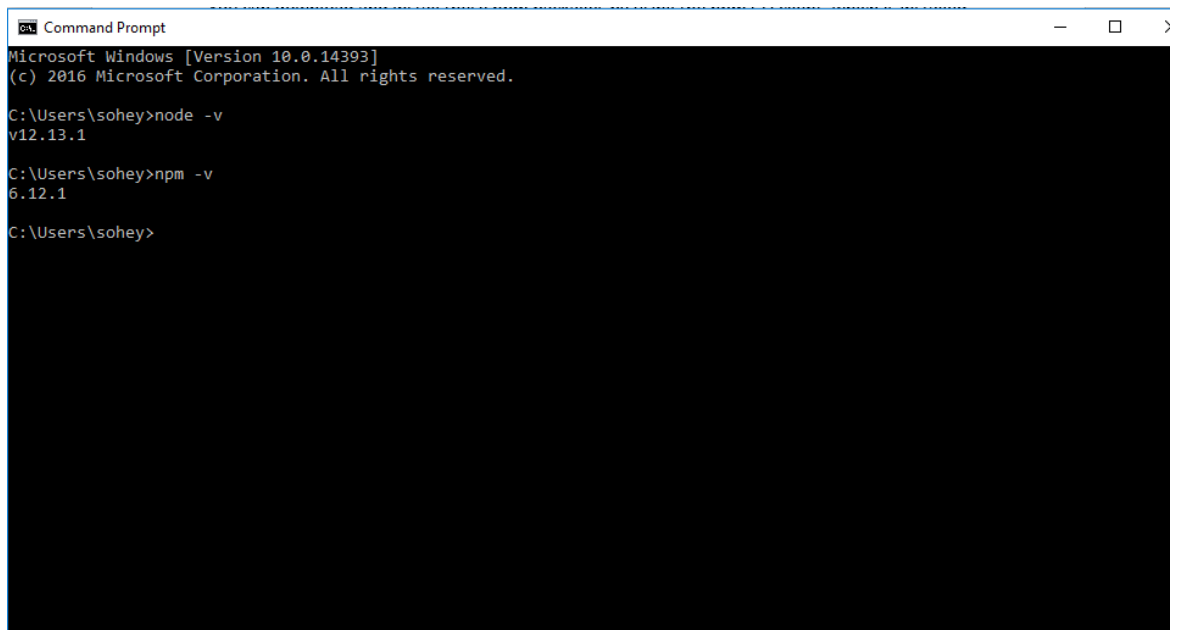
The Angular Framework, Angular CLI, and most of the components used by Angular applications are packaged as npm packages and distributed via the npm registry.

You can download and install these npm packages by using the npm CLI client, which is installed with and runs as a Node.js application.

<https://nodejs.org/en/>

To check the installed version execute these cmd (command) in the cli (command line interface):

- `node -v`
- `npm -v`



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\sohey>node -v
v12.13.1

C:\Users\sohey>npm -v
6.12.1

C:\Users\sohey>
```

## Tools and dependencies

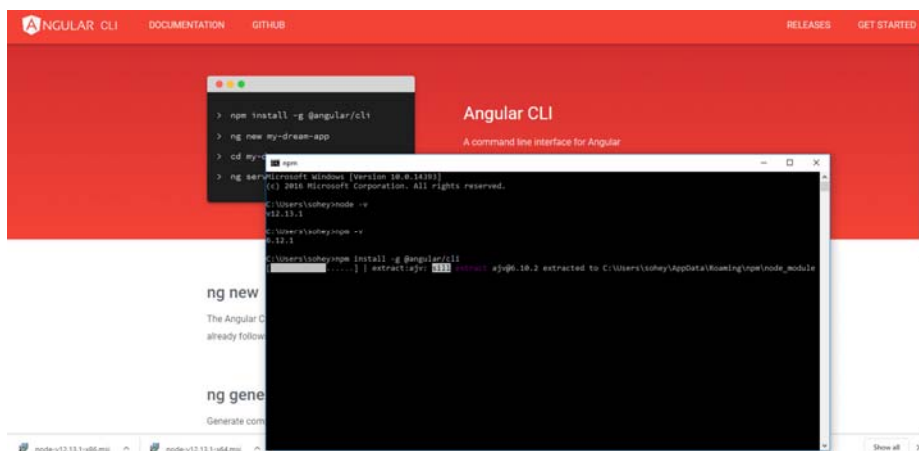
### Angular CLI

The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications. (you can see more information on the link below):

<https://cli.angular.io/>

To install Angular cli execute the cmd:

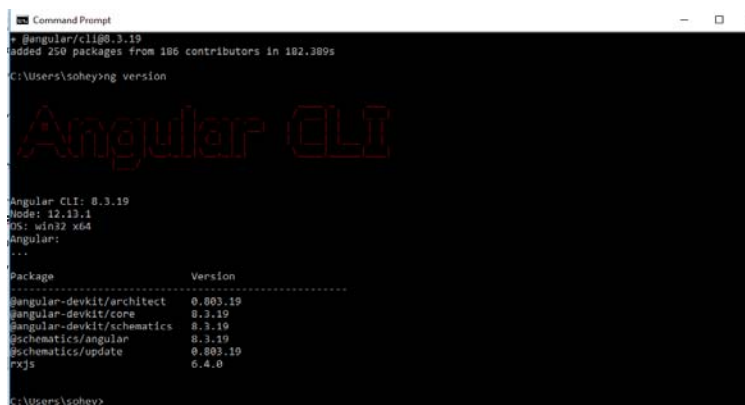
- `npm install -g @angular/cli`



You can find the source code of angular cli in github : <https://github.com/angular/angular-cli>

To check the installed version, execute these cmd (command) in the cli (command line interface):

❖ `ng version`



To get the help of the available options in ng, execute this cmd :

`ng help`

```
Command Prompt
@schematics/angular 8.3.10
@schematics/update 0.803.19
rxjs 6.4.0

C:\Users\sohey>ng help
Available Commands:
  add Adds support for an external library to your project.
  analytics Configures the gathering of Angular CLI usage metrics. See https://v8.angular.io/cli/usage-analytics-gatheri
ng-
  build (b) Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from
within a workspace directory.
  deploy (d) Invokes the deploy builder for a specified project or for the default project in the workspace.
  config Retrieves or sets Angular configuration values in the angular.json file for the workspace.
  doc (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.
  e2e (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor.
  generate (g) Generates and/or modifies files based on a schematic.
  help Lists available commands and their short descriptions.
  lint (l) Runs linting tools on Angular app code in a given project folder.
  new (n) Creates a new workspace and an initial Angular app.
  run Runs an Architect target with an optional custom builder configuration defined in your project.
  serve (s) Builds and serves your app, rebuilding on file changes.
  test (t) Runs unit tests in a project.
  update Updates your application and its dependencies. See https://update.angular.io/
  version (v) Outputs Angular CLI version.
  xi18n Extracts i18n messages from source code.

For more detailed help run "ng [command name] --help"
C:\Users\sohey>
```

## ❖ Available Commands:

**Add** -> Adds support for an external library to your project.

**build (b)** Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.

**deploy (d)** Invokes the deploy builder for a specified project or for the default project in the workspace.

**config** Retrieves or sets Angular configuration values in the angular.json file for the workspace.

**doc (d)** Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.

**e2e (e)** Builds and serves an Angular app, then runs end-to-end tests using Protractor.

**generate (g)** Generates and/or modifies files based on a schematic.

**help** Lists available commands and their short descriptions.

**lint (l)** Runs linting tools on Angular app code in a given project folder.

**new (n)** Creates a new workspace and an initial Angular app.

**run** Runs an Architect target with an optional custom builder configuration defined in your project.

**serve (s)** Builds and serves your app, rebuilding on file changes.

**test (t)** Runs unit tests in a project.

**update** Updates your application and its dependencies. See <https://update.angular.io/>

**version (v)** Outputs Angular CLI version.

**xi18n** Extracts i18n messages from source code.

For more detailed help run "ng [command name] --help"

## Angular project setup

- *ng new myFirstApp*

The ng new cmd creates an Angular workspace folder and generates a new app skeleton.

A newly generated app contains the source files for a root module, with a root component and template. Each app has a src folder that contains the logic, data, and assets.

You can edit the generated files directly, or add to and modify them using CLI commands. Use the ng generate command to add new files for additional components and services, and code for new pipes, directives, and so on. Commands such as add and generate, which create or operate on apps and libraries, must be executed from within a workspace or project folder.

- *ng serve (usually for development mode)*

Builds and serves your app, rebuilding on file changes.

<https://angular.io/cli/serve>

- ✓ *ng build (for production mode)*

Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.

<https://angular.io/cli/build>

---

## Angular project structure

angular.json	Default CLI configuration for all projects in the workspace, including configuration options for build, serve, and test tools that the CLI uses, such as TSLint, Karma, and Protractor.  For the details : <a href="https://angular.io/guide/workspace-config">https://angular.io/guide/workspace-config</a>
package.json	Configures npm package dependencies that are available to all projects in the workspace.  For the details : <a href="https://docs.npmjs.com/files/package.json">https://docs.npmjs.com/files/package.json</a>
package-lock.json	Provides version information for all packages installed into node_modules by the npm client.  For details : <a href="https://docs.npmjs.com/files/package-lock.json">https://docs.npmjs.com/files/package-lock.json</a>
src/	Source files for the root-level application project.
node_modules/	Provides npm packages to the entire workspace. Workspace wide node_modules dependencies are visible to all projects.

For the details : <https://angular.io/guide/file-structure>

## Angular project structure (src)

app/	Contains the component files in which your application logic and data are defined.
assets/	Contains image and other asset files to be copied as-is when you build your application.
environments/	Contains build configuration options for particular target environments. By default there is an unnamed standard development environment and a production ("prod") environment. You can define additional target environment configurations.
index.html	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any <script> or<link> tags here manually.
main.ts	The main entry point for your application. Compiles the application and bootstraps the application's root module (AppModule) to run in the browser.
polyfills.ts	Provides polyfill scripts for browser support.

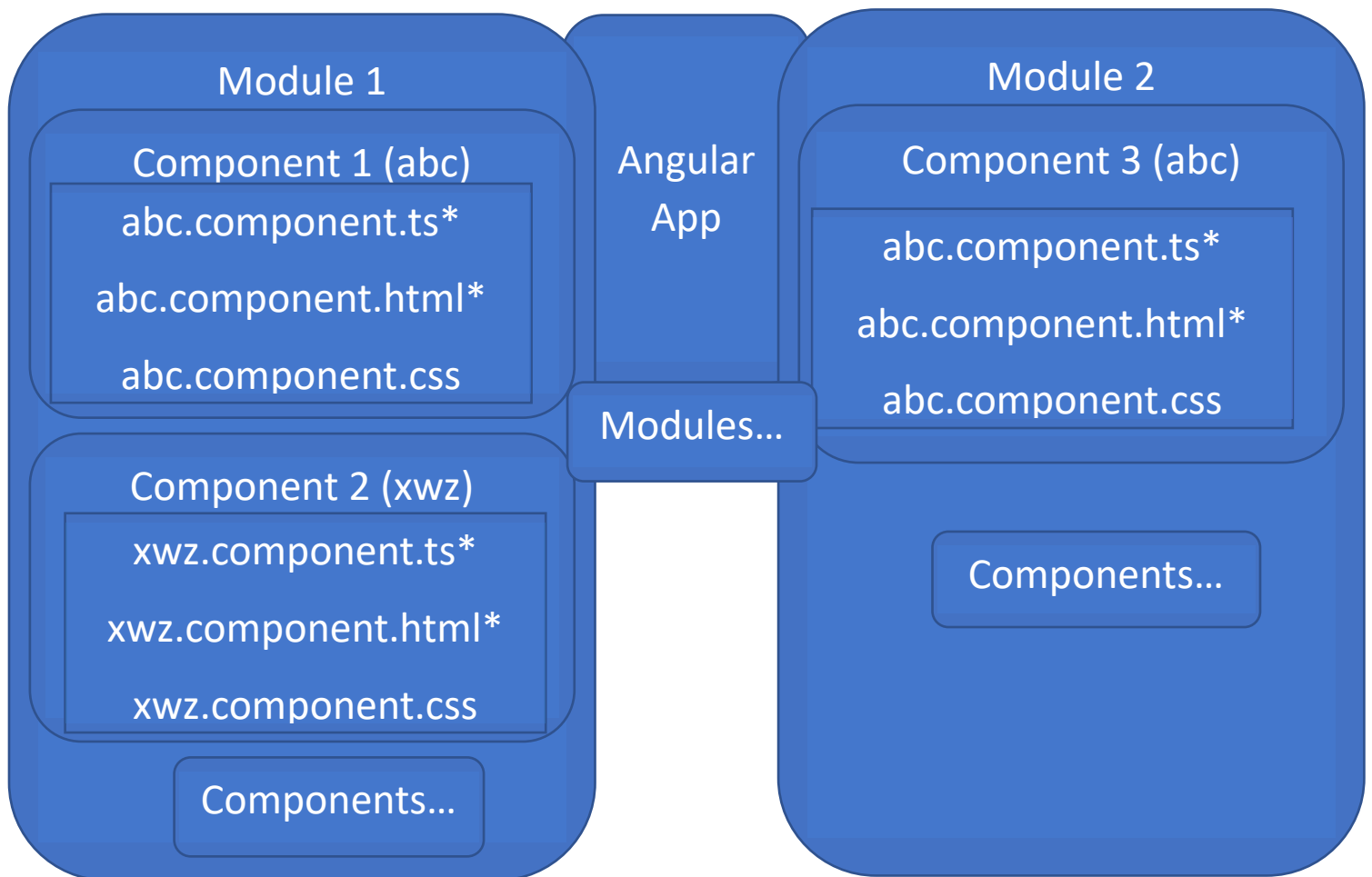
For the details : <https://angular.io/guide/file-structure>

## Angular project structure (src/app)

app/app.component.ts	Defines the logic for the app's root component, named AppComponent. The view associated with this root component becomes the root of the view hierarchy as you add components and services to your application.
app/app.component.html	Defines the HTML template associated with the root AppComponent.
app/app.component.css	Defines the base CSS stylesheet for the root AppComponent.
app/app.component.spec.ts	Defines a unit test for the root AppComponent.
app/app.module.ts	Defines the root module, named AppModule, that tells Angular how to assemble the application. Initially declares only the AppComponent. As you add more components to the app, they must be declared here.

For the details : <https://angular.io/guide/file-structure>

## Angular project composition



## Key concepts

These are some key concepts we recommend that you spend time to discover them.

Typescript

MVC and MVVM architectures

Object Oriented programming

Classes & Objects

Namespaces & packages

Git



---

## Angular Framework – Lab Project Planning

---

### Introduction :

During this course all the students will be asked to be split on a group of 2 or individual, in order to work on application lab.

This lab will be a graded (25%). In this lab the students are asked to apply all what they will learn on the class.

### Steps :

- ✓ Students will give to the instructor the list of groups with the lab subject.
- ✓ The students and the instructor will discuss the project and define the features.
- ✓ After each and every lecture the students will apply new lesson in the project.

### Project development, collaboration & evaluation

In order to work on the project as a team you will need to use Github as a collaborative development tool.

So each team members needs to create an account on github website : <https://github.com/>

Only 1 of the team members needs to initiate the repository, then invite his team mate if applicable and also the instructor.

After that all the contributors in the project will be able to work together.

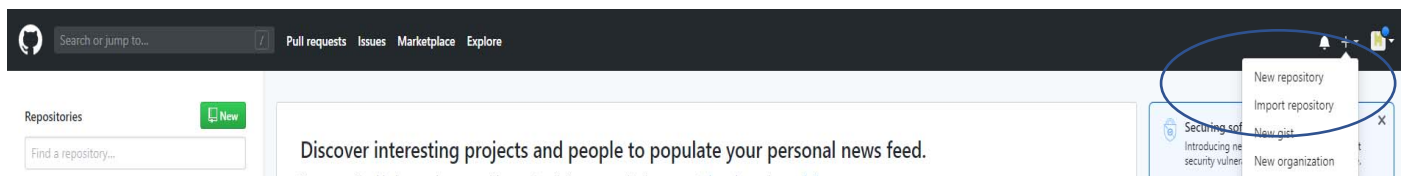
### create account, login, create repository

Create an account on : <https://github.com/>

After that, connect with your new credentials on the same website.

Then create a new repository

Then fill this form: (New Repository)




- Repository name needs to be unique

- Check public
- Check initialize this repo with a README
- Add a license select Apache

Then click on create repository Button

Owner **Repository name \***

 nirou8 /

Great repository names are short and memorable. Need inspiration? How about **fuzzy-telegram**?

Description (optional)

---

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

---

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.


Add .gitignore: **None** ▾ | Add a license: **None** ▾ ⓘ

---

**Create repository**

## create repository

After creating the repository you will be redirected to the repo overview page

 nirou8 / **Test** Unwatch ▾ 1 ★ Star 0 🍴 Fork 0


[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)



No description, website, or topics provided. [Edit](#)


[Manage topics](#)

1 commit 1 branch 0 releases 1 contributor

Branch: master ▾ [New pull request](#) [Create new file](#) [Upload files](#) [Find file](#) [Clone or download ▾](#)

 nirou8 Initial commit Latest commit 623988e now

 LICENSE	Initial commit	now
 README.md	Initial commit	now

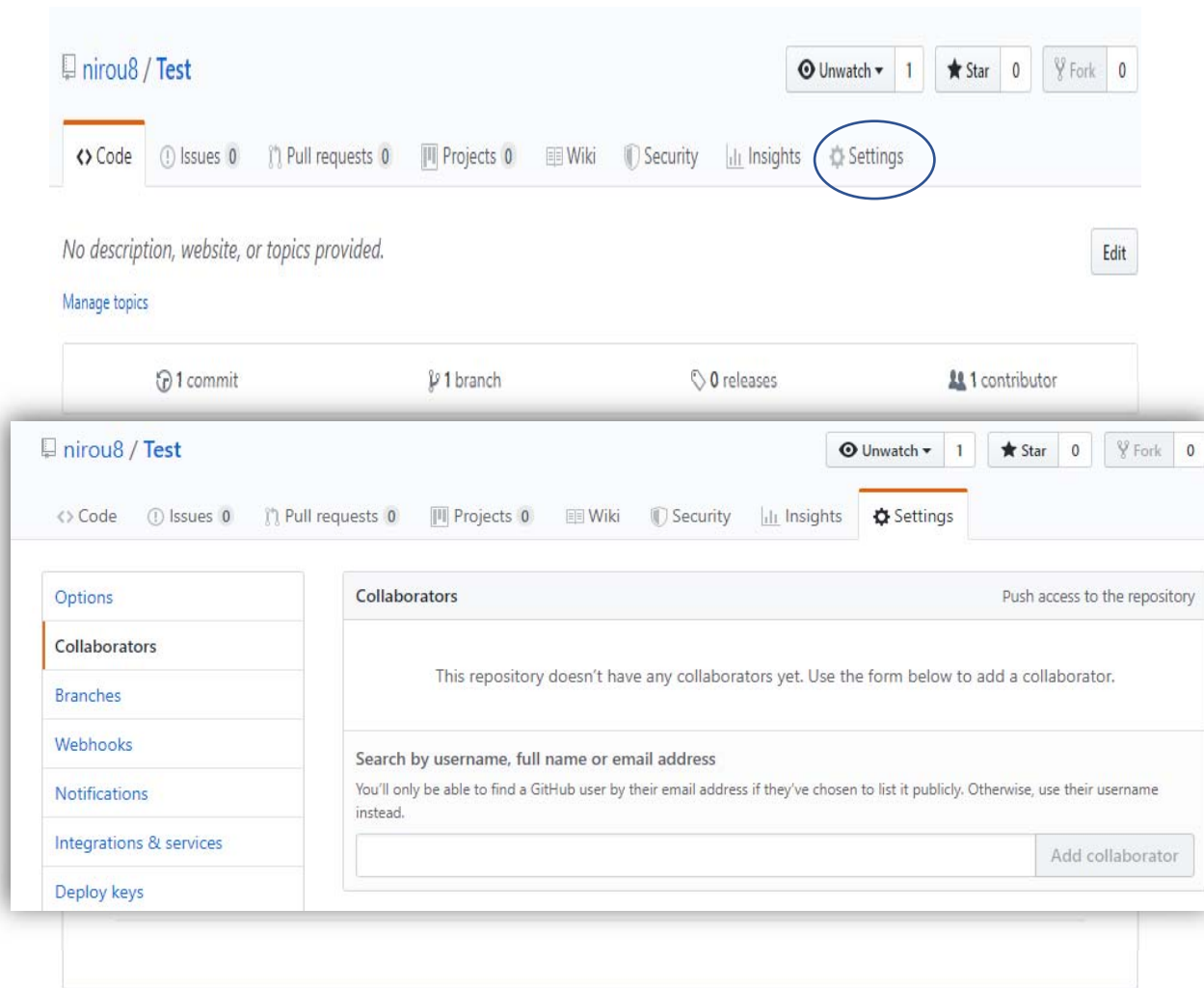
 README.md [Edit](#)

Test

## add teammate to a repository

Now to add the team mate to the project, click on settings

Then type the username of Your teammate and click on Add collaborator button.



## install git bash

Now to use your repository on your machine, you need to install first Git bash tool:

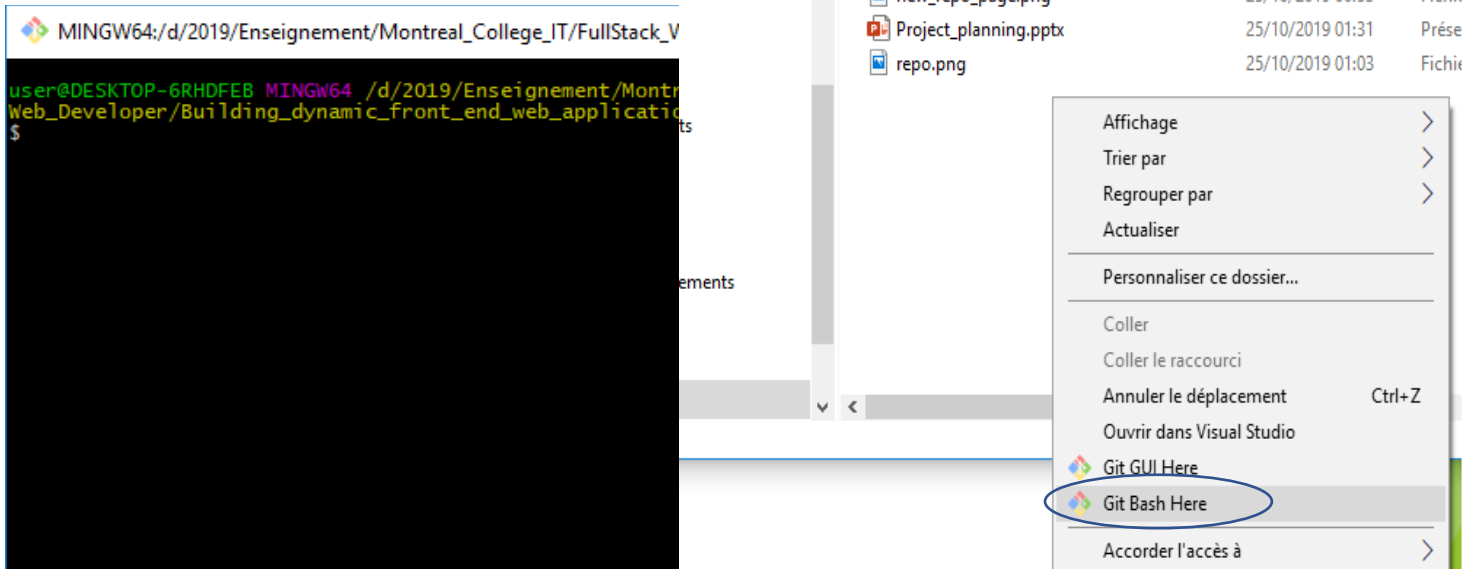
<https://gitforwindows.org/>

Go to this website, download and install the tool.

Git bash is a CLI tool that help us to execute the git commands.

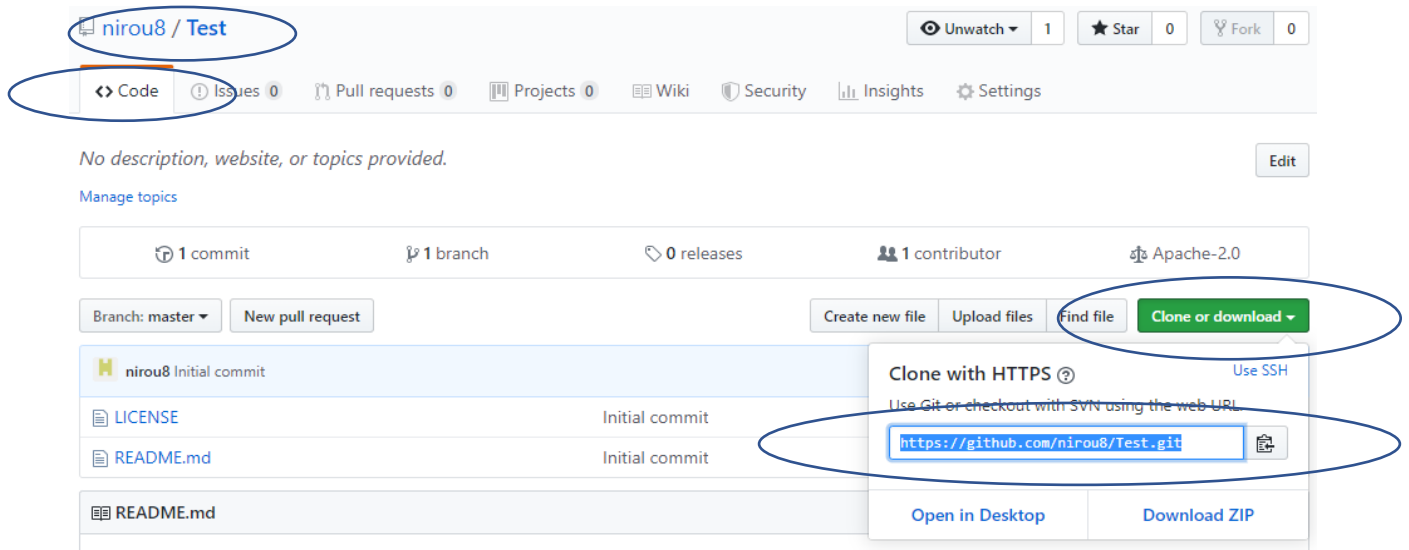
To launch Git bash go to the location in which you want to clone your project from git

Right click and the click on « Git Bash here »



## clone repository

Now you need to clone the repository in your local machine :



- ✓ Copy the [url] of the repo (4)
- ✓ Then in git bash CLI write : git clone [url]
- ✓ After that access the [project folder] in the git bash CLI : cd [project folder]
- ✓ Now you can initiate the Angular project inside this folder using: ng new [project folder]

- ✓ Now we will move the [project folder]/[project folder]/ under [project folder]/

Example : we move Test/Test/ to Test/

---

### add, commit and push

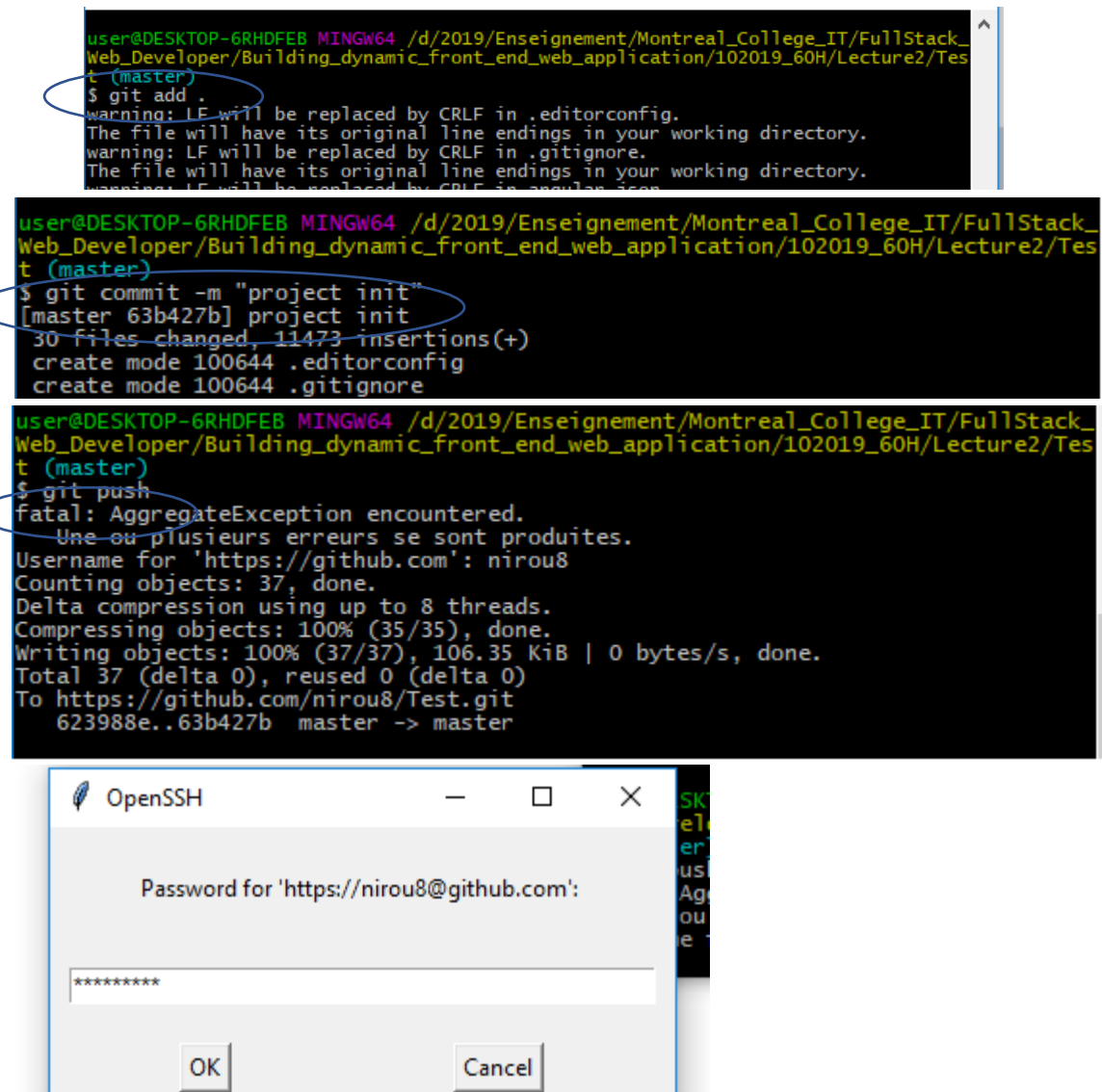
Now to synchronize the local changes with the remote repository you need to execute these commands below :

git add .

git commit -m "project init"

git push

After that maybe it will ask you to enter your credentials git username / password



```
user@DESKTOP-6RHDFFB MINGW64 /d/2019/Enseignement/Montreal_College_IT/FullStack_Web_Developer/Building_dynamic_front_end_web_application/102019_60H/Lecture2/Test (master)
$ git add .
warning: LF will be replaced by CRLF in .editorconfig.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in regular-ice-cream.txt
$ git commit -m "project init"
[master 63b427b] project init
30 files changed, 11473 insertions(+)
create mode 100644 .editorconfig
create mode 100644 .gitignore
$ git push
fatal: AggregateException encountered.
Une ou plusieurs erreurs se sont produites.
Username for 'https://github.com': nirou8
Counting objects: 37, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (35/35), done.
Writing objects: 100% (37/37), 106.35 KiB | 0 bytes/s, done.
Total 37 (delta 0), reused 0 (delta 0)
To https://github.com/nirou8/Test.git
623988e..63b427b master -> master
```

OpenSSH

Password for 'https://nirou8@github.com':

\*\*\*\*\*

OK Cancel

After the synchronization with the remote repository, if you check the repo under github.com you will see the updates

The screenshot shows the GitHub interface for a repository named 'nirou8 / Test'. At the top, there are buttons for 'Unwatch', 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Security', 'Insights', and 'Settings'. A message states 'No description, website, or topics provided.' with an 'Edit' button. Below this, statistics show '2 commits', '1 branch', '0 releases', '1 contributor', and 'Apache-2.0' license. A progress bar is visible. The 'Branch: master' dropdown is set, with a 'New pull request' button. Action buttons include 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A table lists repository files and their commit history:

File	Commit	Time
e2e	project init	16 minutes ago
src	project init	16 minutes ago
.editorconfig	project init	16 minutes ago
.gitignore	project init	16 minutes ago
LICENSE	Initial commit	2 hours ago
README.md	Initial commit	2 hours ago
angular.json	project init	16 minutes ago
package-lock.json	project init	16 minutes ago
package.json	project init	16 minutes ago
tsconfig.json	project init	16 minutes ago
tslint.json	project init	16 minutes ago

At the bottom, there is a 'README.md' file with an edit icon.

Now in order for the teammate to get whatever his colleague has done and pushed, he needs to pull :

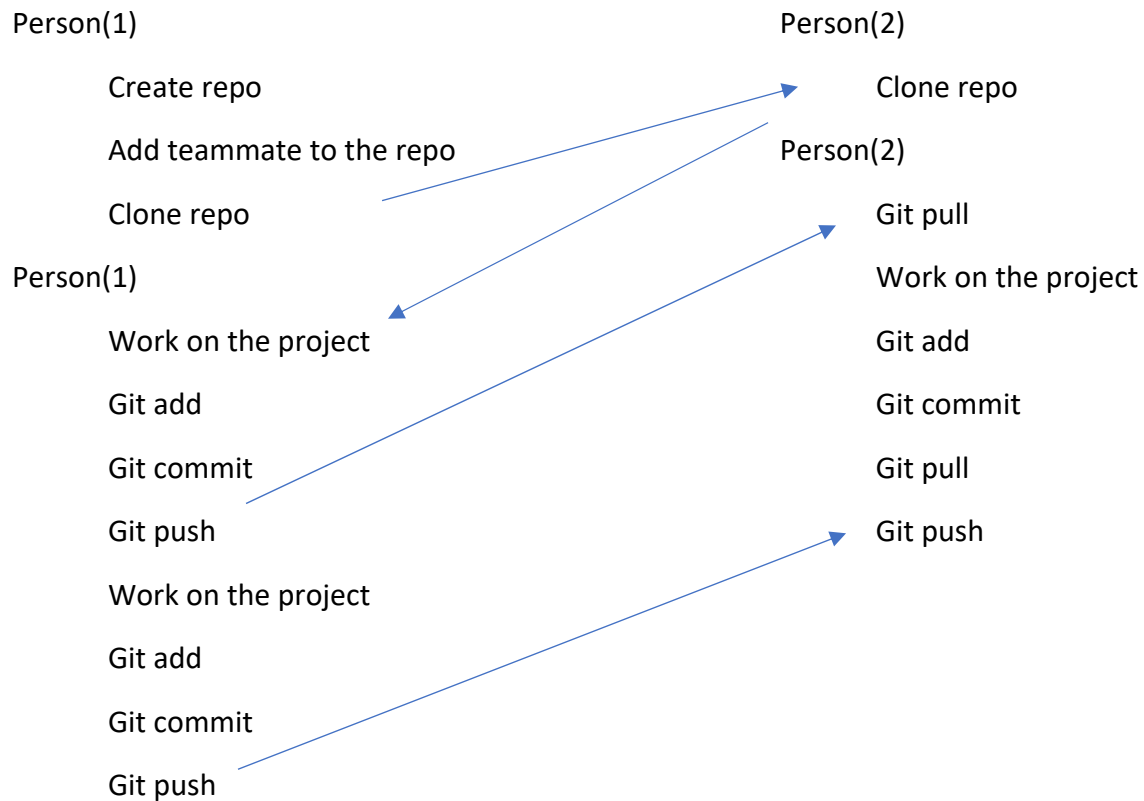
git pull

Then you will see the updates applied in your local project, now you can continue working on the project and in order to publish your account you need to (add, commit and push). In order for your colleague to get your work he needs to (pull).

```
Users\DESKTOP-6RHFEB\MINGW64 /d/2019/Enseignement/Montreal_College_IT/FullStack/
Web_Developer/Building_dynamic_front_end_web_application/102019_60H/Lecture2/Tes
t (master)
$ git pull
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 4), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
From https://github.com/nirou8/Test
63b427b..243d215 master -> origin/master
Updating 63b427b..243d215
Fast-forward
 src/app/app.component.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Let's see here an example of two persons working on a same project together on git:

Person(1) will be the repository creator and the project initiator.



---

### **git commands flow**

To make it easy we recommend always for every user no matter what is the condition to do this flow :

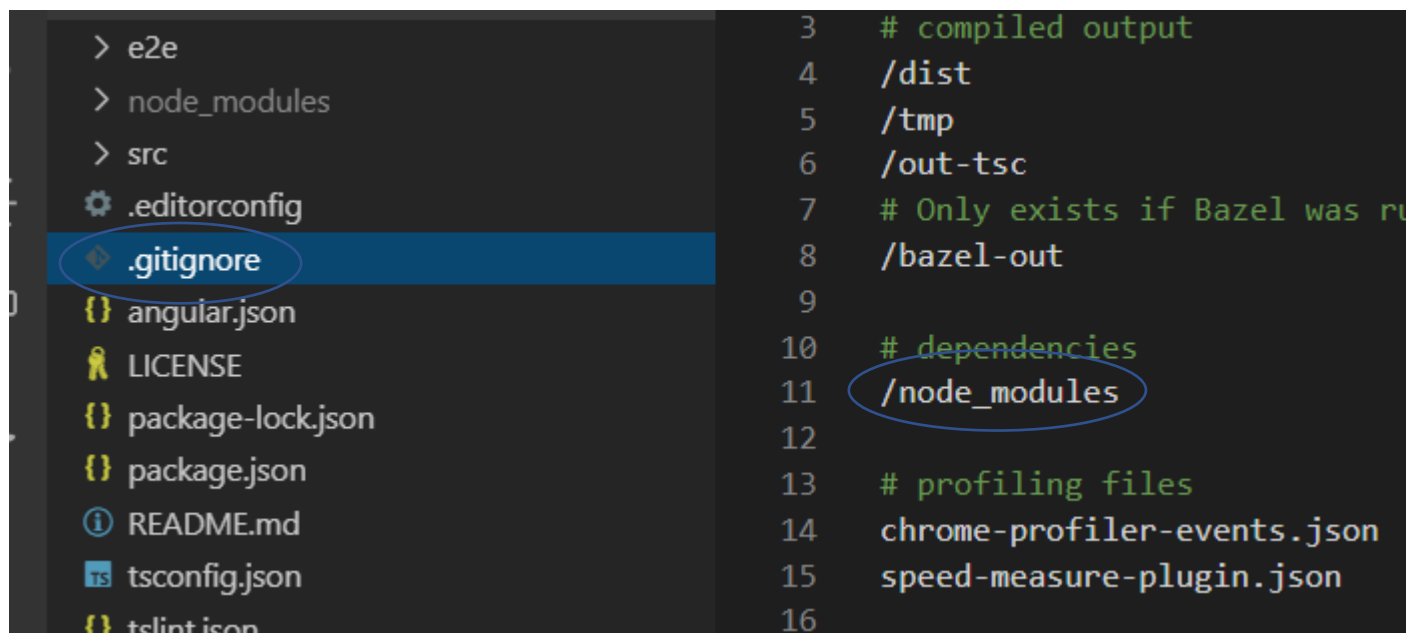
- 1) Clone repo
- 2) Work on the project
- 3) Git add
- 4) Git commit
- 5) Git pull
- 6) Git push

Then the process from (2) to (6) will be repeated on cycle

---

### **.gitignore)**

In the .gitignore file you can put all the files and/or folders that are not needed to be synchronized with the remote repository.



If you check the remote repository you will not find the `node_modules` folder in github, the reason is this folder has a big size and a lot of files. Anyway we can regenerate this folder at anytime by running the command : `npm install`

This command will install all the dependencies that are listed in the `package.json`

---

### git conflict resolving

When developers work on projects with github it happens most of the time that they will work on the same file and maybe change the same portion of code.

As a consequence there will be some conflicts.

So one of the developers needs to resolve the conflict manually.

We will see the example of resolving the conflict in the class.

You can also check this link as a reference.

<https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line>



---

## TypeScript

---

### Introduction :

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language.

TypeScript is designed for development of large applications and transcompiles to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs. TypeScript may be used to develop JavaScript applications for both client-side and server-side (Node.js) execution.

---

### TypeScript vs JavaScript

TypeScript has a feature known as Static typing but JavaScript does not have this feature.

(means “type” checking is done at compile-time, dynamic type, checking is done at run-time)

TypeScript does support optional parameter function but JavaScript does not support optional parameter function.

TypeScript is known as Object oriented programming OOP language whereas JavaScript is a scripting language.

TypeScript gives support for modules whereas JavaScript does not support modules.

TypeScript has Interface but JavaScript does not have Interface.

---

### TypeScript : static typing

In Javascript :

```
var name = "John";  
var age = 25;  
var isActive = true;  
var ages = array(14, 25, 17)
```

In Javascript : (OK)

```
var name = "John";  
name = 25;  
var ages = array(14, "15", 23)
```

In Typescript :

```
let name : string = "John";  
let age : number = 25;  
let isActive : boolean = true;  
let ages : Array<number> = [14, 25, 17]
```

In Typescript : (NOT OK)

```
let name : string = "John";  
name = 25;  
let ages : Array<number> = [14, "15", 23]  
* you cant change the type after..
```

## TypeScript : function with optional parameters

In Javascript :

```
function abc(a, b) {  
    var s = a + b;  
    console.log(s);  
}  
abc(2)
```

In Javascript : (OK)

```
function abc(a, b) {  
    var s = a + b;  
    console.log(s); }  
abc(2)
```

In Typescript :

```
function abc(a,b?){  
    let s = a + (b || 0)  
    console.log(s)  
}
```

In Typescript : (NOT OK)

```
function abc(a, b){  
    let s = a + b;  
    console.log(s);}  
abc(2)
```

---

## TypeScript : Object Oriented Programming (OOP) :

For the concept of OOP we will not deep dive in it in this course, but we will dedicate a focused lectures in the PHP course.

Here we will be interested only in understanding the syntax of what we call a « Class »

Basically a Class is a blueprint of an Object w can say also a Class is a description of the skeleton of an object.

### Class



### Object



JS

TS

In a JavaScript file we have: Variables & functions

Variables ↔ Attributes

In a TypeScript Class we will have: Attributes & Methods

Functions ↔ Methods

## TypeScript : Object Oriented Programming (OOP) :

Syntax :

```
class User{  
  fname : string  
  lname : string  
  age : number  
  constructor (fname, ln, age){  
    this.fname = fname  
    this.lname = ln  
    this.age = age  
  }  
  isAdultOrTeenager(){  
    if(this.age>=18){  
      return « I'm an adult »  
    }else if(this.age<18 && this.age>=12){  
      return « I'm a teenager »  
    }else{ return « no » }  
  }  
}
```

Usage :

```
let user1 = new User("John", "Lee", 26)  
let user2 = new User("Peter", "Rock", 31)  
let user3 = new User("Liam", "Tod", 14)  
user2.isAdultOrTeenager();
```

So :

user1, user2, user3 are Objects

---

## TypeScript : Modules

With Typescript we have the possibility to use functions and/or variables from one file in another file. To do that they introduces the concept of Modules.

Simply to make a class User in Typescript behave as a Module we need to export it (file name is user.ts):

```
export class User{ ... }
```

When you export the class User you can import it in another class

```
Import { User } from 'user'
```

```
Class Group{
```

```
  let u1 = new User() }
```

## TypeScript Advantages

TypeScript always point out the compilation errors at the time of development only. Because of this at the run-time the chance of getting errors are very less whereas JavaScript is an interpreted language.

TypeScript has a feature which is strongly-typed or supports static typing. That means Static typing allows for checking type correctness at compile time. This is not available in JavaScript.

## TypeScript Compilation to Javascript

As we said it before Angular is using Typescript as a programming language.

When we run an angular application the typeScript code is compiled to Javascript code that is understandable and interpreted by the browser.



Angular CLI includes the typescript compiler (tsc), this is why we don't need any additional installs.

To make some tests of compilation outside the Angular CLI we will need to install the tsc in the global environment, for that we need to execute the command :

```
npm install -g typescript
```

After that let's create a typescript file called test.ts, writing some code...

To compile this file let's execute this command and see the result :

```
tsc test
```

---

## Angular Setup

---

Introduction :

Angular project has by default a lots of dependencies that they are installed once we create a new project.

These dependencies are defined in the packages.json file.

These dependencies they are also small projects that are offering some features useful for our project and as a project they have their own dependencies

All the dependencies are saved into the folder called node\_modules

The node\_modules folder can be deleted at any time and it can be setup again by executing the command : npm install

---

## Bootstrap integration

---

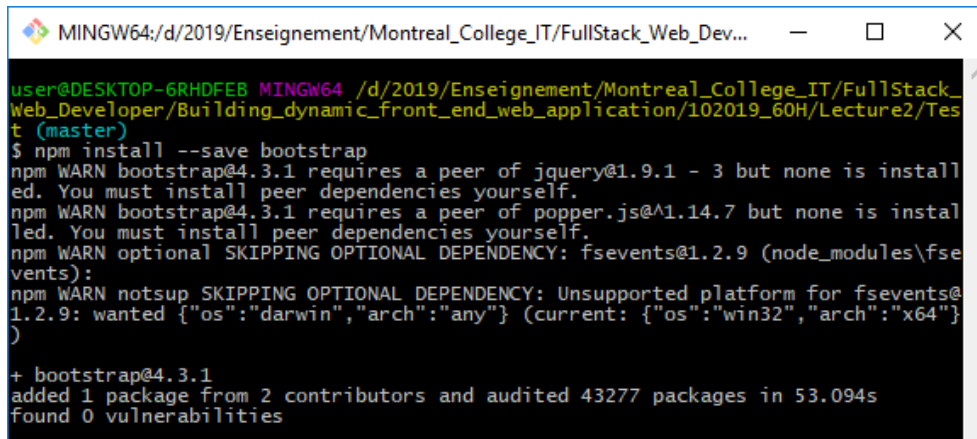
Angular by default does not include Bootstrap framework.

So let's add Bootstrap and see how we add (libraries/ dependencies)

- ✓ Open git bash in the project location
- ✓ Run the command : npm install --save bootstrap

After this you should find bootstrap added automatically in:

- ✓ package.json
- ✓ node\_modules folder
- ✓ package-lock.json

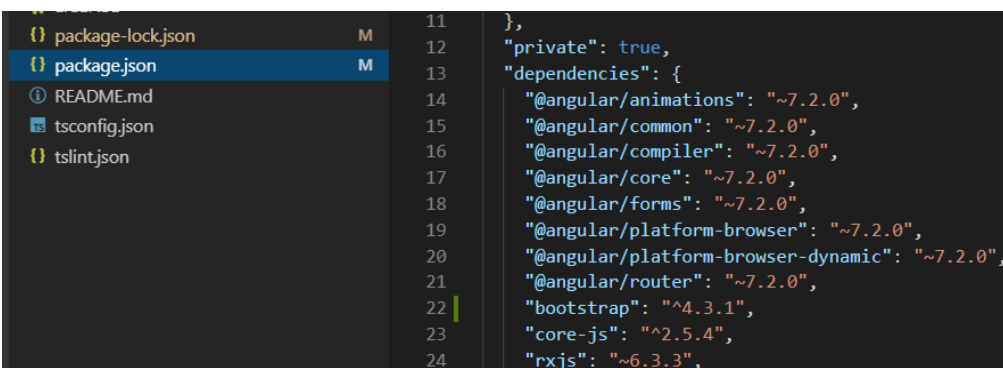


```
user@DESKTOP-6RHD5EB MINGW64 /d/2019/Enseignement/Montreal_College_IT/FullStack_Web_Development/Building_dynamic_frontend_web_application/102019_60H/Lecture2/Test (master)
$ npm install --save bootstrap
npm WARN bootstrap@4.3.1 requires a peer of jquery@1.9.1 - 3 but none is installed. You must install peer dependencies yourself.
npm WARN bootstrap@4.3.1 requires a peer of popper.js@^1.14.7 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ bootstrap@4.3.1
added 1 package from 2 contributors and audited 43277 packages in 53.094s
found 0 vulnerabilities
```



```
1609 },
1610 },
1611 "bootstrap": {
1612   "version": "4.3.1",
1613   "resolved": "https://registry.npmjs.org/bootstrap/-/bootstrap-4.3.1.tgz",
1614   "integrity": "sha512-rXqQOM1H1V1A1ZDyPz1u1f1b1h17b07ef+zLLP1W6494pDxcH234pJ8w7c/6R40UW1zA11MpxjvxZg5kmsbag=="
1615 },
1616 "brace-expansion": {
1617   "version": "1.1.11",
1618   "resolved": "https://registry.npmjs.org/brace-expansion/-/brace-expansion-1.1.11.tgz",
1619   "integrity": "sha512-iCuPHDFgrHX7H2vEI5hhkzcheCgsITB5Dye6BzToVxHuV4vMOfUVVk+yeNglfw6UtXKS853k2+G6NCiWM1kB0A=="
1620 }
```



```
11 },
12 "private": true,
13 "dependencies": {
14   "@angular/animations": "~7.2.0",
15   "@angular/common": "~7.2.0",
16   "@angular/compiler": "~7.2.0",
17   "@angular/core": "~7.2.0",
18   "@angular/forms": "~7.2.0",
19   "@angular/platform-browser": "~7.2.0",
20   "@angular/platform-browser-dynamic": "~7.2.0",
21   "@angular/router": "~7.2.0",
22   "bootstrap": "^4.3.1",
23   "core-js": "^2.5.4",
24   "rxjs": "~6.3.3",
```

- > body-parser
- > bonjour
- > bootstrap
- > brace-expansion
- > braces

## Bootstrap usage :

### 1- Using Bootstrap with normal HTML :

We include the bootstrap.min.css file in to the html file

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<link rel="stylesheet" href="css/bootstrap.min.css">
<title>Test</title>
</head>
```

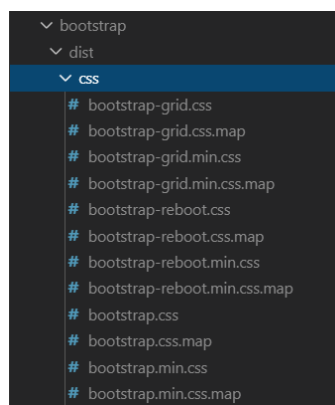
### 2- Using Bootstrap with Angular app

Let's look for the path of the bootstrap.min.css :

node\_modules/bootstrap/dist/css/bootstrap.min.css

All the styles we include them in the angular.json file in the styles array

Means : we need to add path of bootstrap.min.js in Style :[array] seprating with comma

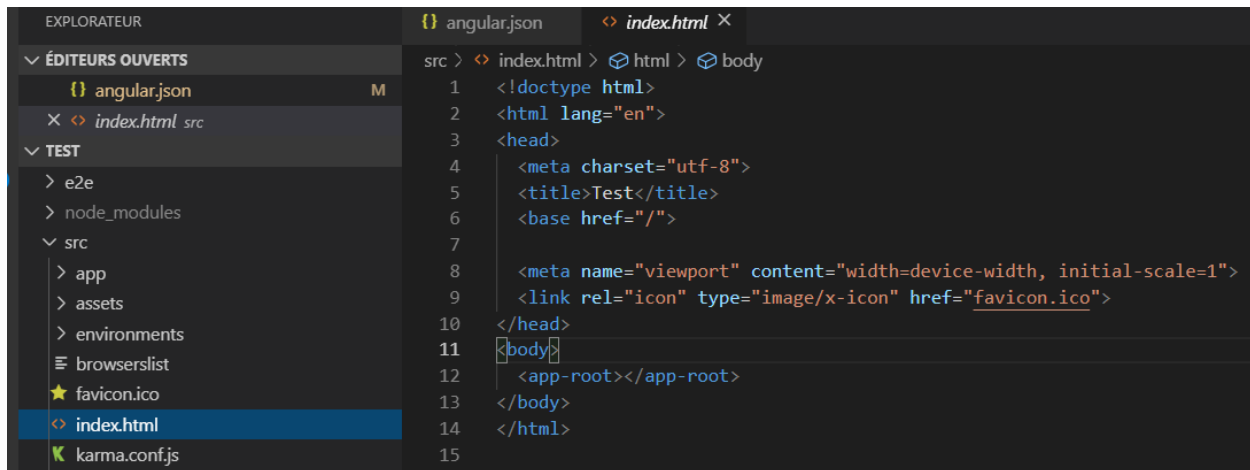


## Angular app Startup & loading ?

Let's understand how actually Angular application is starting and running.

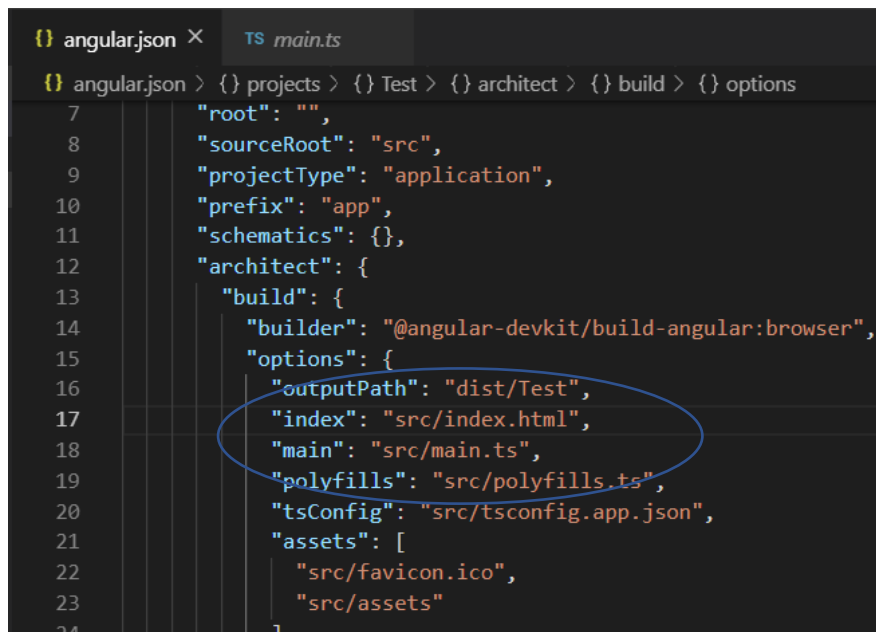
As usual the the entry point of the web app is the index.html , When the compilation is happening angular.json will dictate many rules Including what files to include in the

Index.html on application bootstrapping ---> So when the application is starting the main.js which is the compilation of the main.ts will be added in the index.html and we can see that if we inspect the code.



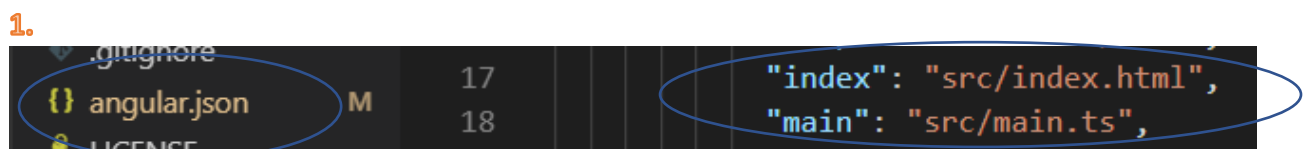
```
EXPLOREUR
└─ Éditeurs ouverts
   └─ angular.json
      └─ index.html src
         └─ TEST
            └─ e2e
               └─ node_modules
                  └─ src
                     └─ app
                        └─ assets
                           └─ environments
                              └─ browserslist
                                 └─ favicon.ico
                                    └─ index.html
                                       └─ karma.conf.js

angular.json
src > index.html > html > body
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Test</title>
6   <base href="/">
7
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
15
```



```
angular.json
angular.json > {} projects > {} Test > {} architect > {} build > {} options
7 "root": "",
8 "sourceRoot": "src",
9 "projectType": "application",
10 "prefix": "app",
11 "schematics": {},
12 "architect": {
13   "build": {
14     "builder": "@angular-devkit/build-angular:browser",
15     "options": {
16       "outputPath": "dist/Test",
17       "index": "src/index.html",
18       "main": "src/main.ts",
19       "polyfills": "src/polyfills.ts",
20       "tsConfig": "src/tsconfig.app.json",
21       "assets": [
22         "src/favicon.ico",
23         "src/assets"
24       ]
25     }
26   }
27 }
```

Angular app Startup & loading :



```
1.
angular.json
angular.json M
17 "index": "src/index.html",
18 "main": "src/main.ts",
```

2.

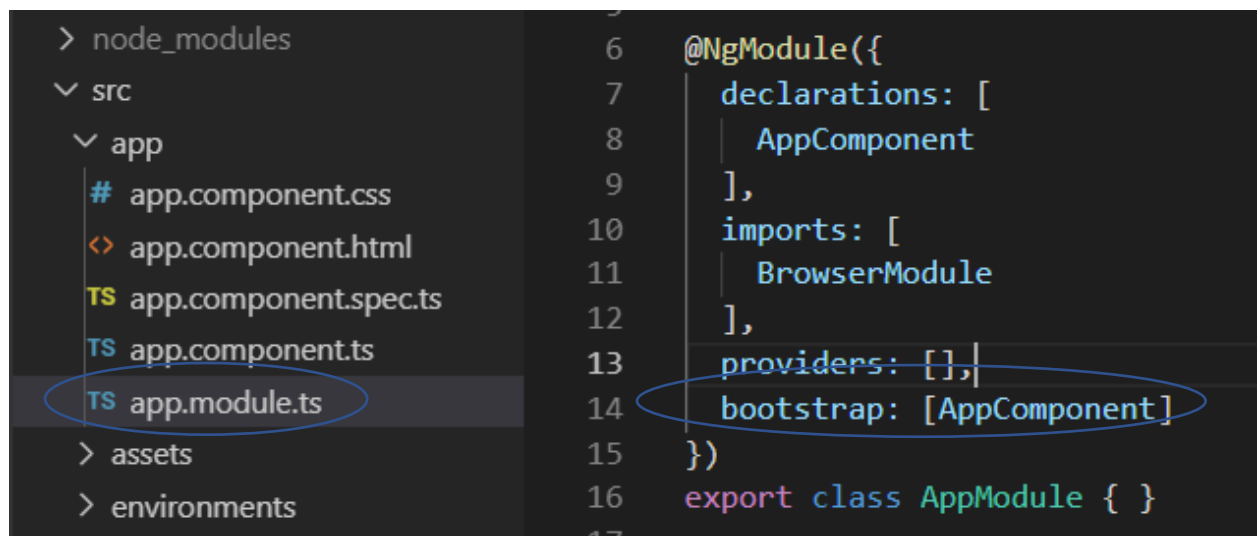


The screenshot shows the VS Code interface. On the left, the file explorer displays a project structure with files like `environments`, `browserslist`, `favicon.ico`, `index.html`, `karma.conf.js`, and `main.ts`. The `main.ts` file is selected and highlighted with a blue circle. On the right, the code editor shows the following TypeScript code:

```
10  
11 platformBrowserDynamic().bootstrapModule(AppModule)  
12   .catch(err => console.error(err));  
13
```

The `bootstrapModule` method call is circled in blue.

3.

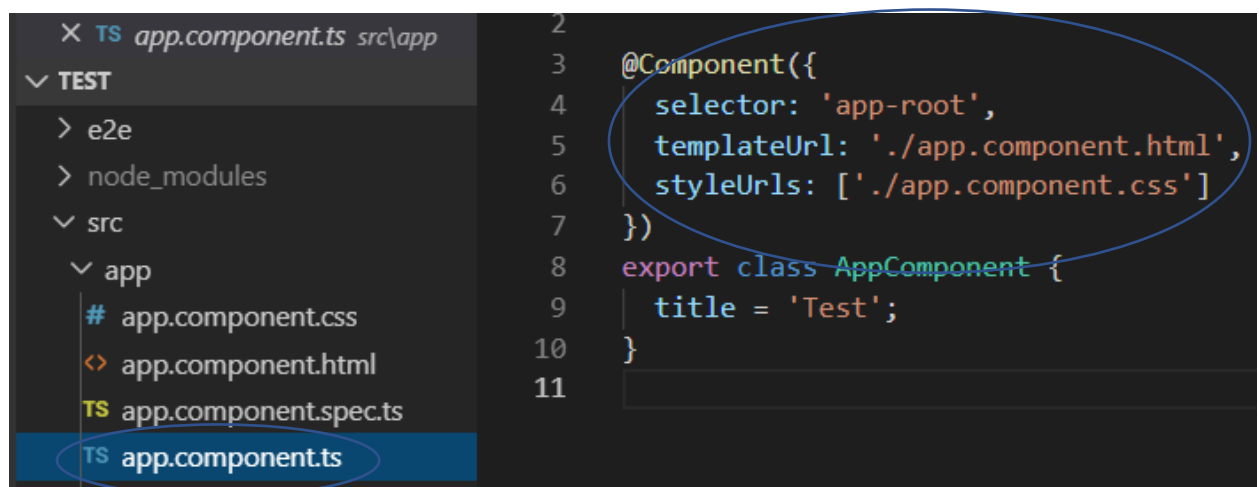


The screenshot shows the VS Code interface. On the left, the file explorer displays a project structure with folders like `node_modules`, `src`, and `assets`. The `src` folder is expanded, showing subfolders like `app` and `environments`. The `app` folder is expanded, showing files like `app.component.css`, `app.component.html`, `app.component.spec.ts`, and `app.module.ts`. The `app.module.ts` file is selected and highlighted with a blue circle. On the right, the code editor shows the following TypeScript code:

```
6 @NgModule({  
7   declarations: [  
8     AppComponent  
9   ],  
10  imports: [  
11    BrowserModule  
12  ],  
13  providers: [],  
14  bootstrap: [AppComponent]  
15 })  
16 export class AppModule { }  
17
```

The `bootstrap: [AppComponent]` property is circled in blue.

4.



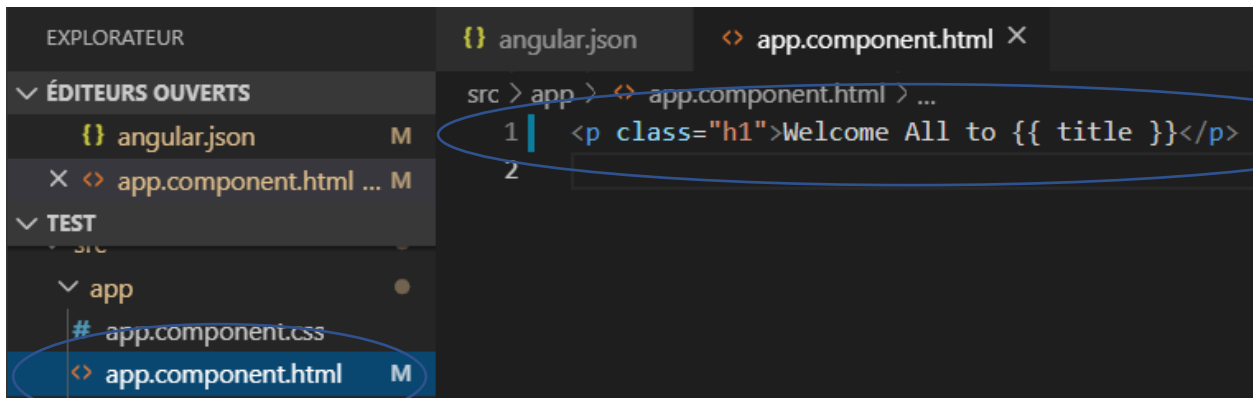
The screenshot shows the VS Code interface. On the left, the file explorer displays a project structure with folders like `node_modules`, `src`, and `assets`. The `src` folder is expanded, showing subfolders like `app` and `environments`. The `app` folder is expanded, showing files like `app.component.css`, `app.component.html`, `app.component.spec.ts`, and `app.component.ts`. The `app.component.ts` file is selected and highlighted with a blue circle. On the right, the code editor shows the following TypeScript code:

```
2  
3 @Component({  
4   selector: 'app-root',  
5   templateUrl: './app.component.html',  
6   styleUrls: ['./app.component.css']  
7 })  
8 export class AppComponent {  
9   title = 'Test';  
10 }  
11
```

The `@Component` decorator and its properties are circled in blue.

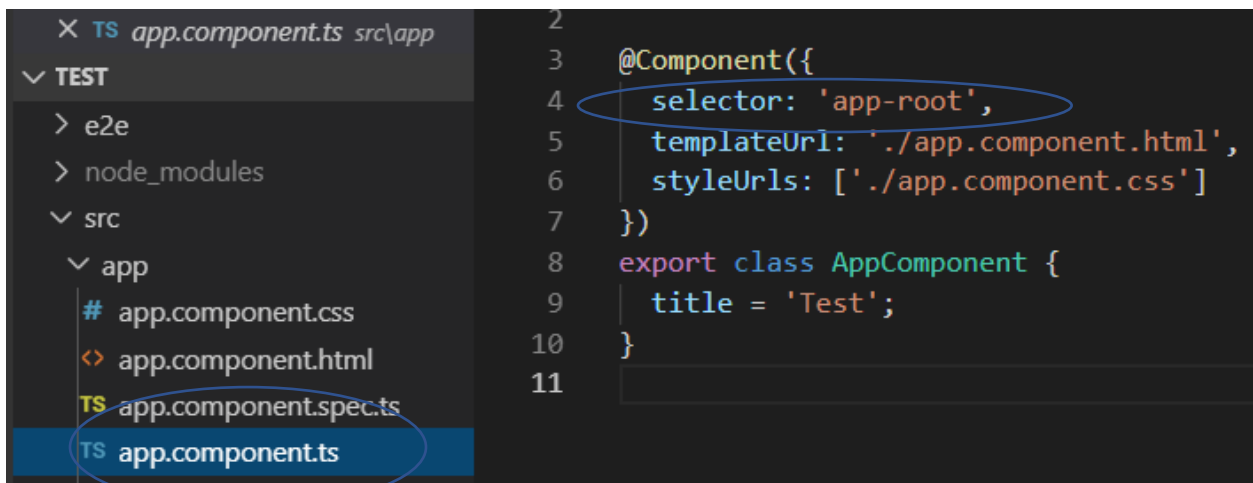


5.

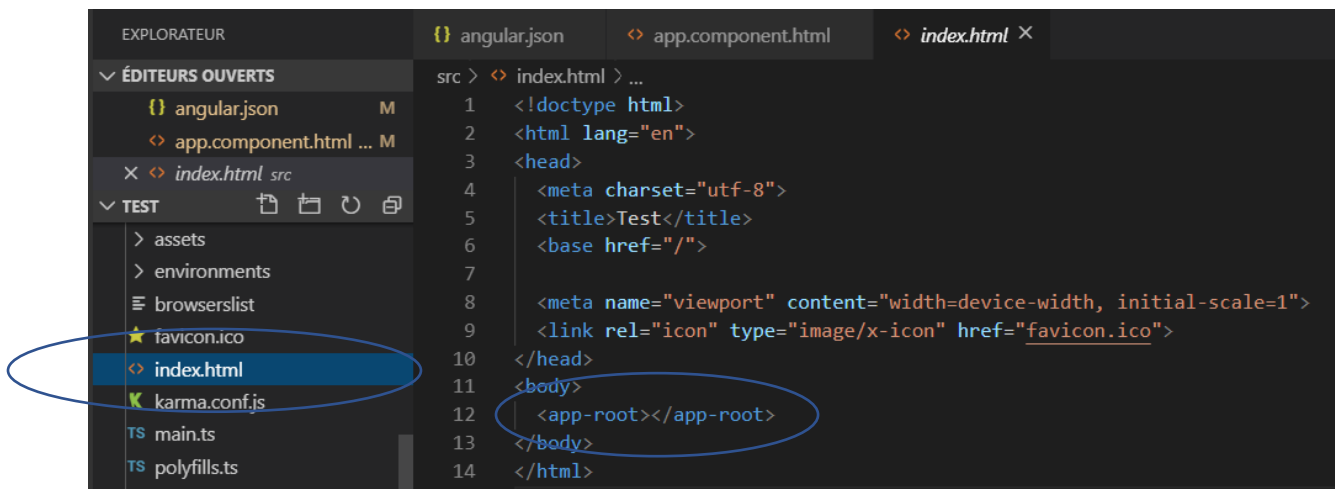


Angular app Startup & loading

A :



B:

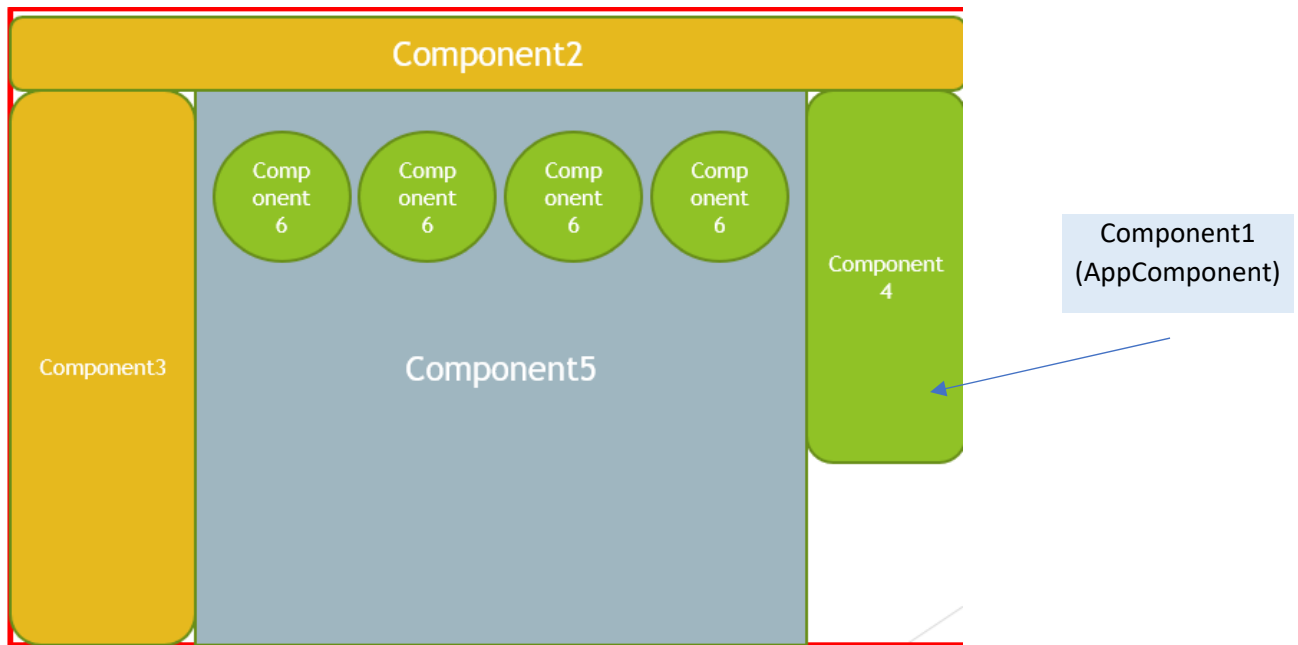


---

## Angular Component

---

Components are a key feature in angular. Our angular application will be composed by many components.



A component is composed by four files (we will consider X as a component name) :

- ✓ x.component.html (mandatory)

The .html file is the HTML template of the component having the code of the specific part, ex : (sidebar, navbar, aside,...)

- ✓ x.component.ts (mandatory)

The .ts file is a typescript class holding the business logic of the component

- ✓ x.component.css (optional)

The .css file is the styling of the HTML part if needed

- ✓ x.component.spec.ts (optional)

The .spec.ts file is the typescript class used for writing the unit test cases...

## Components :

Let's understand the minimum required syntax of a component class named navbar just an example :

navbar.component.ts: (when you open it you will see ) :

```
import { Component } from '@angular/core';
```

```
@Component({
```

```
  selector: 'app-navbar',
```

```
  templateUrl: './navbar.component.html',
```

```
  styleUrls: ['./navbar.component.css']
```

```
})
```

```
export class NavbarComponent {
```

```
  //put your code here
```

```
}
```

### Decorator

We use decorators to give additional information about the component

This is the selector (Tag) that will be referring to the component in other component's template

We define the link between the component (.ts) and the template (.html) and the style (.css)

---

## Creat a Components

---

How to create a component?

Usually all the components should be under the folder app.

1. We can create a component Manually:

- ✓ Let's create a folder called sidebar under the folder app
- ✓ Under the folder sidebar we will create the three files :
  - sidebar.component.html
  - sidebar.component.ts
  - sidebar.component.css
- ✓ And let's put the necessary code in there

2. We can create a component Automatically using the cmd:

- ✓ ng generate component navbar    OR
- ✓ ng g c navbar

---

## How to use Components

---

We need to do two steps:

1. Add the component in the related Module (in our case we have only one module “AppModule” but in angular application we can have multiple modules)

```
4 import { AppComponent } from './app.component';
5 import { SidebarComponent } from './sidebar/sidebar.component';
6
7 @NgModule({
8   declarations: [
9     AppComponent
10    SidebarComponent
11  ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule {}
```

\* in this way when the application is running the AppModule will load the SidebarComponent

2. Add the component selector (Tag) in the HTML Template of the desired component

\* In this case we will add it in the app.component.html this custom tag which let us load the component there

```
{ angular.json | TS app.module.ts | <> app.component.html X
src > app > <> app.component.html > ...
1 | <app-sidebar></app-sidebar>
2 |
```

---

## Component Selector

---

In a component Selector can be one of the three ways :

1. Element :
  - Selector: ‘app-navbar’
  - how to Use ➔ <app-navbar></app-navbar>

## 2. Class :

- selector: 'app-navbar'
- how to Use ➔ <div class='app-navbar'></div>

## 3. Attribute :

- selector: '[app-navbar]'
- how to Use ➔ <div app-navbar></div>

\* you have to define the selector in ts files

---

## Modules

---

Let's understand the minimum required syntax of a module class : (inside the app.module.ts) :

```
import { BrowserModule } from '@angular/platform-browser';
```

```
import { NgModule } from '@angular/core';
```

```
import { AppComponent } from './app.component';
```

```
import { Content1Component } from './content1/content1.component';
```

```
import { Content2Component } from './content2/content2.component';
```

```
import { Content3Component } from './content3/content3.component';
```

```
@NgModule({
```

```
  declarations:[
```

```
    AppComponent,
```

```
    NavbarComponent,
```

```
    Content1Component, Content2Component, Content3Component],
```

```
  imports:[
```

```
    BrowserModule
```

```
  ],
```

```
  providers:[],
```

```
  bootstrap:[AppComponent],
```

```
})
```

```
export class AppModule { }
```

### Decorator

We use decorators to give additional information about the Module

We define all the components that we want them to be loaded with the App and used in this section of declarations, this declaration is an array inside the decorator

In order to be able to use other (except NgModule) modules we need to import them in this section of imports

## Workspace npm dependencies

<a href="#">@angular/animations</a>	Angular's animations library makes it easy to define and apply animation effects such as page and list transitions. For more information, see <a href="https://angular.io/guide/animations">https://angular.io/guide/animations</a>
<a href="#">@angular/common</a>	The commonly-needed services, pipes, and directives provided by the Angular team. For more information, see the <a href="https://angular.io/api/common">https://angular.io/api/common</a> .
<a href="#">@angular/compiler</a>	Angular's template compiler. It understands templates and can convert them to code that makes the application run and render. Typically you don't interact with the compiler directly; rather, you use it indirectly via platform-browser-dynamic when JIT compiling in the browser.
<a href="#">@angular/core</a>	Critical runtime parts of the framework that are needed by every application. Includes all metadata decorators, <a href="#">Component</a> , <a href="#">Directive</a> , dependency injection, and the component lifecycle hooks.
<a href="#">@angular/forms</a>	Support for both <a href="#">template-driven</a> and <a href="#">reactive forms</a> . For information about choosing the best forms approach for your app, see <a href="#">Introduction to forms</a> .
<a href="#">@angular/platform-browser</a>	Everything DOM and browser related, especially the pieces that help render into the DOM.
<a href="#">@angular/platform-browser-dynamic</a>	Includes <a href="#">providers</a> and methods to compile and run the app on the client using the JIT compiler.
<a href="#">@angular/router</a>	The router module navigates among your app pages when the browser URL changes. For more information, see <a href="#">Routing and Navigation</a> .

<https://angular.io/guide/npm-packages> (for the full details)

\* JIT compiler: A Just-In-Time (JIT) **compiler** is a feature of the run-time interpreter, that instead of interpreting bytecode every time a method is invoked, will compile the bytecode into the machine code instructions of the running machine, and then invoke this object code instead.

---

## Angular –Routing

---

Routes :

Angular Router is a powerful JavaScript router built and maintained by the Angular core team that can be installed from the `@angular/router` package.

Routers provides a complete routing library with the possibility to have multiple router outlets, different path matching strategies, easy access to route parameters and route guards to protect components from unauthorized access.

The Angular router is a core part of the Angular platform. It enables developers to build Single Page Applications with multiple views and allow navigation between these views.

How To use:

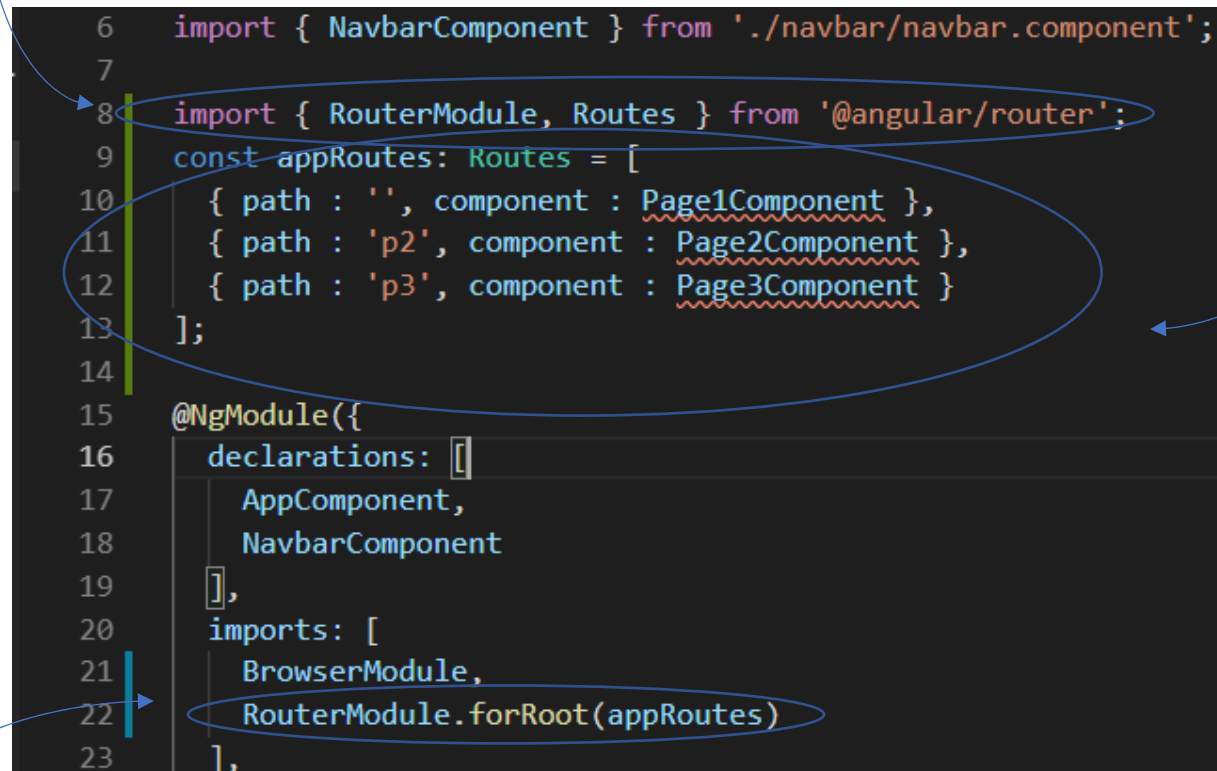
First thing we need to update our app.module.ts :

- 1- Import the new predefined classes :

```
import { RouterModule, Routes } from '@angular/router';
```

- 2- Configuration: Create the array of objects that is matching each component with it's path

```
const appRoutes: Routes = [ path : '' ,
```

A screenshot of a code editor showing the app.module.ts file. The code is as follows:

```
6 import { NavbarComponent } from './navbar/navbar.component';
7
8 import { RouterModule, Routes } from '@angular/router';
9 const appRoutes: Routes = [
10   { path : '', component : Page1Component },
11   { path : 'p2', component : Page2Component },
12   { path : 'p3', component : Page3Component }
13 ];
14
15 @NgModule({
16   declarations: [
17     AppComponent,
18     NavbarComponent
19   ],
20   imports: [
21     BrowserModule,
22     RouterModule.forRoot(appRoutes)
23   ],
```

Blue circles and arrows highlight specific parts of the code. One circle around line 8 points to the import statement for RouterModule and Routes. Another circle around lines 9-13 points to the appRoutes array definition. A third circle around line 22 points to the RouterModule.forRoot(appRoutes) call in the imports array. Arrows also point from the list items above to these specific code sections.

- 3- Load the routes created in the module (add it to import)

```
RouterModule.forRoot(appRoutes)
```

- 4- Then you need to add the routing tag in the HTML code where you want these components will be loaded when the path is matching : **<router-outlet></router-outlet>**

5- After that all what you need to do is to activate the links in for example the <a> tags :

In the <a> tag we no longer use “href” attribute.

We replaced it with “routerLink”

Example : <a routerLink="/">page1</a>

<a routerLink="/p2">page2</a>

<a routerLink="/p3">page3</a>

OR

<a [routerLink]=" '/p2' ">page2</a>

OR

<a [routerLink]="['/p2']">page2</a>

---

We will come back later in details for these alternatives and why we will need them

### Styling the routes

With angular like with bootstrap we can style our active router link. Simply we will update our links for example :

<a routerLink="/" **routerLinkActive="active"**>page1</a>

<a routerLink="/p2" **routerLinkActive="active"**>page2</a>

<a routerLink="/p3" **routerLinkActive="active"**>page3</a>

By adding routerLinkActive=“any Styling class” that class will be active on the precise link when the path is matching

---

## Data binding

---

Data binding in Angular is the synchronization / Communication between the model and the view.

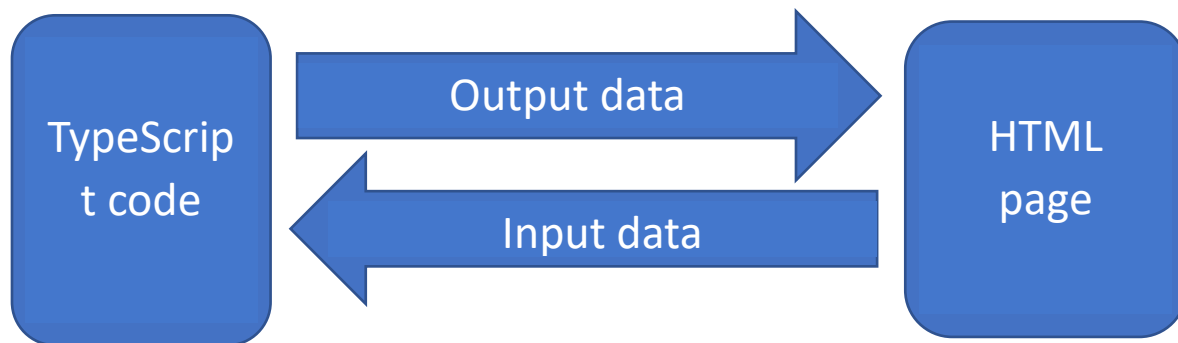
What is the View? : The view is the HTML page (any HTML page of any component is called a view)

What is the Model? : The Model is any type of data available in the Typescript file (any TS file of any component)



## Data Binding: different methods :

- 1) String Interpolation : {{ data }}
  - 2) Property binding : [property] ="data"
  - 3) Event binding : (event) ="expression"
  - 4) Two-way-binding : [(ngModel)] ="data"
- 



---

### 1. String Interpolation :

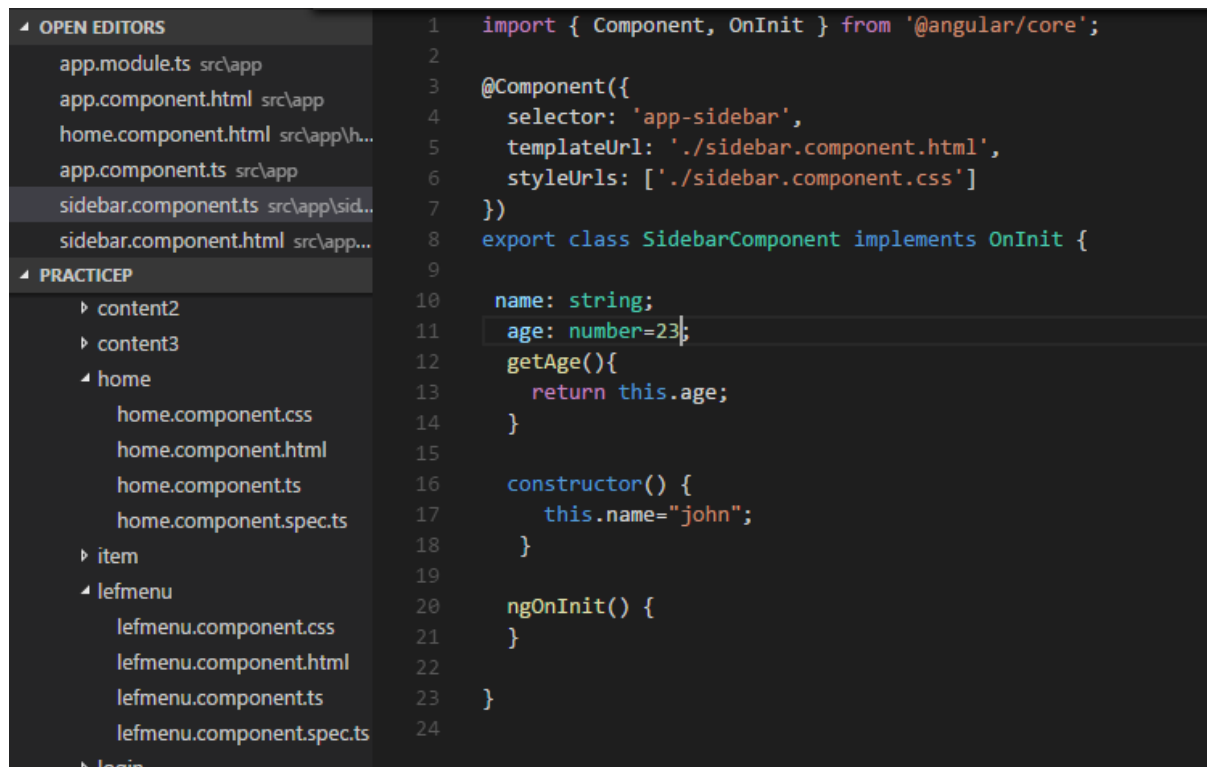
---

We use string interpolation to display in the HTML some dynamic data coming from the Typescript.

Example :

✓ In typescript :

```
export class SomeComponent {  
    name: string = «John»;  
    age: number = 23;  
    getAge(){  
        return this.age;  
    }  
}
```



In HTML :

<p>Your name is {{ name }} and you are {{ age }} years old.</p>

Also we can use functions :

<p>Your name is {{ name }} and you are {{ getAge() }} years old.</p>

---

## 2. Property Binding

---

We use property binding to assign dynamically values to the HTML DOM property.

Example :

In typescript :

```
export class SomeCompoent {
  isActive: boolean = false;
}
```

In HTML :

```
<button class=«btn btn-primary» [disabled]=«!isActive»>Save</button>
```

NOTE :

```
<p>{{ isActive }}</p> ↔ or → <p [innerText]=«isActive»></p>
```

---

### 3. Event Binding

---

We use property binding to assign dynamically values to the HTML DOM property.

Example :

In typescript :

```
export class SomeCompoent {  
  status : string = ""  
  someFunction(){  
    this.status = «button clicked»  
  }  
}
```

In HTML :

```
<button class=«btn btn-primary» (click)=«someFunction()»>Click</button>  
<p>{{ status }}</p>
```

---

Example2

In typescript :

```
export class SomeCompoent {  
  writtenText : string = ""  
  onWrite(event: Event){  
    this.writtenText =  
    (<HTMLInputElement>event.target).value;  
  }  
}
```

In HTML :

```
<input type=«text» class=«form-control» (input)=«onWrite($event)»>
<p>{{ writtenText }}</p>
```

---

#### 4. two-way-binding

---

For the two-way-binding we need to enable « ngModel » directive, for this reason we need to add FormsModule to the imports array in the AppModule. From @angular/forms.

Example

In typescript :

```
export class SomeComponent {
  writtenText : string = ""
}
```

In HTML :

```
<input type=«text» class=«form-control» [(ngModel)]=«writtenText»>
<p>{{ writtenText }}</p>
```

---

## Angular Framework -Directives

---

Directives:

Directives are the most important things in angular since Components are Directives.

At the core, a directive is a function that executes whenever the Angular compiler finds it in the DOM. Angular directives are used to extend the power of the HTML by giving it new syntax

In another word Directives are instructions / rules in the DOM

We have 3 categories of the Directives:

- ✓ Components
- ✓ Attribute Directives
- ✓ Structural Directives

### Directives : Components

As we saw earlier, components are just directives with templates. Under the hood, they use the directive API and give us a cleaner way to define them.

It is mainly used to specify the HTML templates. It is the most commonly-used directive in an Angular project. It is decorated with the `@component` decorator.

The component directive is used to specify the template/HTML for the Dom Layout.

The other two directive types don't have templates. Instead, they're specifically tailored to DOM manipulation.

### Directives : Attribute Directives

Attribute directives manipulate the DOM by changing its behavior and appearance.

We use attribute directives to apply conditional style to elements, show or hide elements or dynamically change the behavior of a component according to a changing property.

We have 2 Attribute Directives :

#### ❖ ngStyle

NgStyle directive is similar to one of the data binding techniques called style binding in Angular. The main point is NgStyle is used to set multiple inline styles for the HTML element.

#### ❖ ngClass

It is similar to NgStyle attribute but here it is using the class attribute of the HTML element to apply the style.

### **Directives : Attribute Directives – ngStyle**

Example :

```
<p [ngStyle]=«{'background-color' : 'red', 'color' : 'white'}»>Some Text</p>
```

OR

```
<p [ngStyle]=«{backgroundColor : 'red', 'color' : 'white'}»>Some Text</p>
```

OR

```
<p [ngStyle]=«{backgroundColor : bgColor, 'color' : txColor}»>Some Text</p>
```

AND

In TypeScript :

```
export class SomeComponent {  
    bgColor = 'blue';  
    txColor = 'white';}
```

---

### **Directives : Attribute Directives – ngClass**

Example :

In CSS :

```
.bgB{  
    background-color = 'blue';}  
  
.txC{  
    color = 'white';}
```

In HTML :

```
<p [ngClass]=«{bgB : isOK, txC : txColor === 'white'}»>Some Text</p>
```

In TypeScript :

```
export class SomeComponent {
```

```
isOK = true;
txColor = 'white'; }
```

## Directives : Structural Directives

These are specifically tailored to create and destroy DOM elements.

The structural Angular directives are much less DOM friendly, as they add or completely remove elements from the DOM. So, when using these, we have to be extra careful, since we're actually changing the HTML structure

We have 3 Structural Directives :

### \*ngIf

\*ngIf It is used to create or remove a part of the DOM tree depending on a condition.

### \*ngFor

\*ngFor It is used to customize data display. It is mainly used for displaying a list of items using repetitive loops.

### \*ngSwitch

\*ngSwitch It is like the JavaScript switch. It can display one element from among several possible elements, based on a switch condition. Angular puts only the selected element into the DOM.

## Directives : Structural Directives - \*ngIf

In HTML :

```
<p *ngIf=«isClicked»>Display some text</p>
```

OR

```
<ng-template [ngIf]=«isClicked»>
```

```
<p>Display some text</p>
```

```
</ng-template>
```

In TypeScript :

```
export class SomeComponent {
    isClicked = false; }
```

## Directives : Structural Directives - \*ngIf/else

In HTML :

```
<p *ngIf=«isClicked; else notClicked»>Display some text</p>
```

OR

```
<ng-template *ngIf=«isClicked; else notClicked»>
```

```
<p>Display some text</p>
```

```
</ng-template>
```

AND

```
<ng-template #notClicked>
```

```
<p>Another text as alternative</p>
```

```
</ng-template>
```

In TypeScript :

```
export class SomeComponent {  
    isClicked = false; }
```

## Directives : Structural Directives - \*ngFor

In HTML :

```
<div *ngFor=«let item of [1,2,3,4]>
```

```
{{ item }}
```

```
</div>
```

OR

```
<div *ngFor=«let item of items>
```

```
{{ item }} is of index {{ i }}
```

```
</div>
```

In TypeScript :

```
export class SomeComponent {  
    items = ['item1', 'item2', 'item3'] }
```



## **Directives : Structural Directives - \*ngFor with index**

In HTML :

```
<div *ngFor=«let item of items; let i= index»>
  {{ item }} is of index {{ i }}
</div>
```

In TypeScript :

```
export class SomeComponent {
    items = ['item1', 'item2', 'item3'] }
```

## **Directives : Structural Directives – ngSwitch**

In HTML :

```
<div ngSwitch=«value»>
  <p *ngSwitchCase=«5»> choice 1</p>
  <p *ngSwitchCase=«10»> choice 2</p>
  <p *ngSwitchCase=«15»> choice 3</p>
  <p ngSwitchDefault> default choice</p>
</div>
```

In TypeScript :

```
export class SomeComponent {
    value = 10 }
```

## **Directives : Binding custom properties**

In HTML :

```
<app-item [value]=«'hello'»></app-item>
```

In TypeScript :

```
Import { Input } from '@angular/core'
export class SomeComponent {
  @Input() value : string;
}
```

## Example 2

In HTML :

```
<app-item *ngFor=«item of items» [value]=«item»></app-item>
```

In TypeScript :

```
Import { Input } from '@angular/core'  
export class SomeComponent {  
  @Input() value : {title : string, price : number, description : string}; }
```

---

### Directives : Binding custom properties / Alias

In HTML :

```
<app-item *ngFor=«item of items» [itm]=«item»></app-item>
```

In TypeScript :

```
                                Import { Input } from '@angular/core'  
export class SomeComponent {  
                                @Input(itm) value : {title : string, price : number, description :  
string};  
}
```

NOTE : when you give an alias to an Attribute you cannot call it with it's original name anymore.

---

---

### Angular Framework - Routing – Deep Dive

---

Routers : Routing programatically :

In HTML :

```
<button class=«btn btn-primary» (click)=«goTo()»>Submit</button>
```

In TypeScript :

```
Import { Router } from '@angular/router'  
export class SomeComponent {
```

```

constructor(private router: Router){}

goTo(){
//some code

this.router.navigate(['/somePath']);      }      }

```

### Routers : Children routers (Nested routers)

With Angular it's possible to nest Components (Load components inside another components...)

All these nested components are handled by the Nested routing, Example :

```

24  const appRoutes: Routes = [
25    { path : '', component : LoginComponent },
26    { path : 'registration', component : RegistrationComponent },
27    { path : 'home', component : HomeComponent, children : [
28      { path : '', component : ContentComponent },
29      { path : 'sell_path', component : SellComponent },
30      { path : 'order_link', component : OrdersComponent },
31      { path : 'favorites', component : FavoritesComponent },
32      { path : 'details/:id', component : DetailsComponent },
33      { path : 'add-product', component : AddProductComponent },
34    ] },
35  ];

```

By adding the children all the children components will be loaded in the parent component, of course for this we need to do another step.

```

9      <router-outlet></router-outlet>

```

We need to add a router-outlet tag in the parent component, in our case here in the HomeComponent

### Routers : Routers with parameters

With routes also we can pass parameters in the path dynamically (let's follow these steps):

In the routing paths we define parameters as a variable with ':' as a prefix, like we can see in the picture below : :n, :description, :price

```

29    { path : 'details/name/:n/desc/:description/p/:price', component : DetailsComponent },

```

Then, in the routerLink, you should use the property binding of [routerLink], that way you can pass multiple parameters

```
.description }}</p>  
" [disabled]="btnStatus" [routerLink]="['/home/details/name', laptop.name, 'desc', laptop.description, 'p', laptop.price]" >{{ btnText }}</button>
```

The [routerLink] property binding will take an array of parameters ( a mix between strings and variables) that if you assemble them you will get the full path defined in the app.module.ts.

- :n => laptop.name
- :description => laptop.description
- :price => laptop.price

---

## Routers : Routers with parameters

Last thing we need to get all these parameters / information in the destination components:

In our case, in the details.component.ts we need to do the necessary things in order to access these informations.

```
1  import { Component, OnInit } from '@angular/core';  
2  import { ActivatedRoute } from '@angular/router';  
3  
4  @Component({  
5    selector: 'app-details',  
6    templateUrl: './details.component.html',  
7    styleUrls: ['./details.component.css']  
8  })  
9  export class DetailsComponent implements OnInit {  
10    name: string;  
11    description: string;  
12    price: string;  
13    constructor(private route: ActivatedRoute) { }  
14  
15    ngOnInit() {  
16      this.name = this.route.snapshot.params['n'];  
17      this.description = this.route.snapshot.params['description'];  
18      this.price = this.route.snapshot.params['price'];  
19    }  
20  
21  }
```

---

## Angular Framework -Local References

---

Definition :

Instead of two-way binding, we can easily fetch a value of any input through **local references in Angular**.

Local references can be fetched directly from the component template and into the component typescript class.

A local reference can be placed on any HTML element by adding #anyname.

### Create Local Reference :

Example :

to create a local reference :

```
<div #mydiv></div>
```

```
<input type="text" #inputref />
```

→ #mydiv and #inputref are local references for the div and the input

---

### Read Local Reference

Example :

to read a local reference we can do it in two ways :

1) Send the local reference as a method parameter :

In .html

```
<form id="login_form" (submit)="loginUser(email)">
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control"
      id="exampleInputEmail1" aria-describedby="emailHelp"
      placeholder="Enter email" required #email>
    <small id="emailHelp" class="form-text text-warning" #sm>
  </div>
```

In .ts

```
loginUser(e){  
  console.log(this.smm.nativeElement.innerText);  
  console.log(this.password.nativeElement.value);  
  let wrongCredentials = true;  
  let accounts = JSON.parse(localStorage["accounts"]);  
  for(let account of accounts){  
    if(e.value == account.email &&
```

## Read Local Reference

Example :

to read a local reference we can do it in two ways :

- 1) Use the @ViewChild decorator :

In .html

```
<div class="form-group">  
  <label for="exampleInputPassword1">Password</label>  
  <input type="password" class="form-control"  
    #password  
    id="exampleInputPassword1" placeholder="Password" required>  
</div>
```

► In .ts

```
1 import { Component, OnInit, ViewChild, ElementRef } from '@angular/core';  
2 import { Router } from '@angular/router';  
3
```

```
21 @ViewChild('password') password: ElementRef;  
22 @ViewChild('smm') smm: ElementRef;  
23  
24 fColor: string = "#ff0000";  
25 isOK = false;  
26 fSize = 23;  
27 constructor(private router: Router) {  
28  
29 }  
30  
31 ngOnInit() {  
32 }  
33  
34 loginUser(e){  
35   console.log(this.password.nativeElement.value);  
36
```

---

## Template Driven Forms (TD Forms)

---

### TD Forms

Template Driven form or (TD forms) are a way of managing and manipulating form based on the Local References.

In order to use and apply TD forms in our project, we need to follow some steps :

1) In app.module.ts

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

},
imports: [
  BrowserModule,
  RouterModule.forRoot(appRoutes),
  FormsModule,
  ReactiveFormsModule
],
```

NOTE : without this step, of course the TD forms will not work.

---

### TD Forms

2) In x.component.html

A) Prepare your form tag with a local reference and an ngSubmit event to fire a method

The local reference needs to be mapped with the “ngForm” in order to make angular that this form is an angular form (a special one)

B) For each form input add a name and ngModel attribute, the ngModel binding will let the form knows that the information needs to be send with the data.

3) In x.component.ts

A) Import the NgForm form @angular/forms library

B) use the @ViewChild decorator in order to synchronize the #f form in the .html with the loginForm property in the .ts

```
import { NgForm } from '@angular/forms';
```

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  connectedUser = {
    first_name: '',
    last_name: '',
    email: '',
    password: '',
    country: '',
    state: '',
    city: '',
    zip: ''
  }
  @ViewChild('f') loginForm : NgForm;
```

---

## TD Forms

4) In x.component.ts

A) By clicking on the Submit button the ngSubmit event will be fired and the loginUser() method will be executed.

Within this method we can get all the information submitted with the form like the email and the password

Each information is accessible by it's name defined in the html part, but not as a direct information, it's stored under the "form\_property.value.name" :

This.loginForm.value.email



```

loginUser(){
  console.log(this.loginForm);

  let wrongCredentials = true;
  let accounts = JSON.parse(localStorage["accounts"]);
  for(let account of accounts){
    if(this.loginForm.value.email == account.email &&
      this.loginForm.value.password == account.password ){
      wrongCredentials = false;
      this.router.navigate(['/home']);
    }
  }
}

```

TD Forms :

5) In x.component.html

With TD forms we can add some form validations differently from the usual way.

1) we give some rules attributes to our form fields like : required, email...

Required : means that the field needs to be mandatory, not empty

Email : means that the field needs to be having the structure of an email address.

By giving #f="ngForm" to the form tag as we said we will tell angular that this is not an ordinary form but a special one. Special one means will have additional attributes and informations that we can use like :

valid / invalid : these are attributes that only one at a time of them will be assigned to the form tag to indicate if the form is valid or not with respect of the validation rules of all the fields.

2) By giving a form field a local reference example : #em="ngModel", means that the field also is no longer a normal field but is an angular field that will have many more additional informations like :

valid / invalid : these are attributes that only one at a time of them will be assigned to the field tag to indicate if the field value is valid or not with respect of the validation rules.

Touched / untouched : these are attributes that only one at a time of them will be assigned to the field tag to indicate if the field has been touched by the user or not.

3) we can use these attributes in order to make some styling based on some conditions.

```

<form id="login_form" (ngSubmit)="loginUser()" #f="ngForm">
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control"
      name="email"
      ngModel
      #em="ngModel"
      placeholder="Enter email" required email>
    <small class="form-text text-warning" *ngIf="!em.valid && em.touched">Yuo need to enter a valid Email address</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control"
      name="password"
      ngModel
      id="exampleInputPassword1" placeholder="Password" required>
  </div>
  <button type="submit" class="btn btn-warning" [disabled]="!f.valid">Submit</button>
</form>

```