

Lab Introducción Angular - ArgentinaPrograma

Autor: Mg. Ing. Hernán Borré

Precondición:

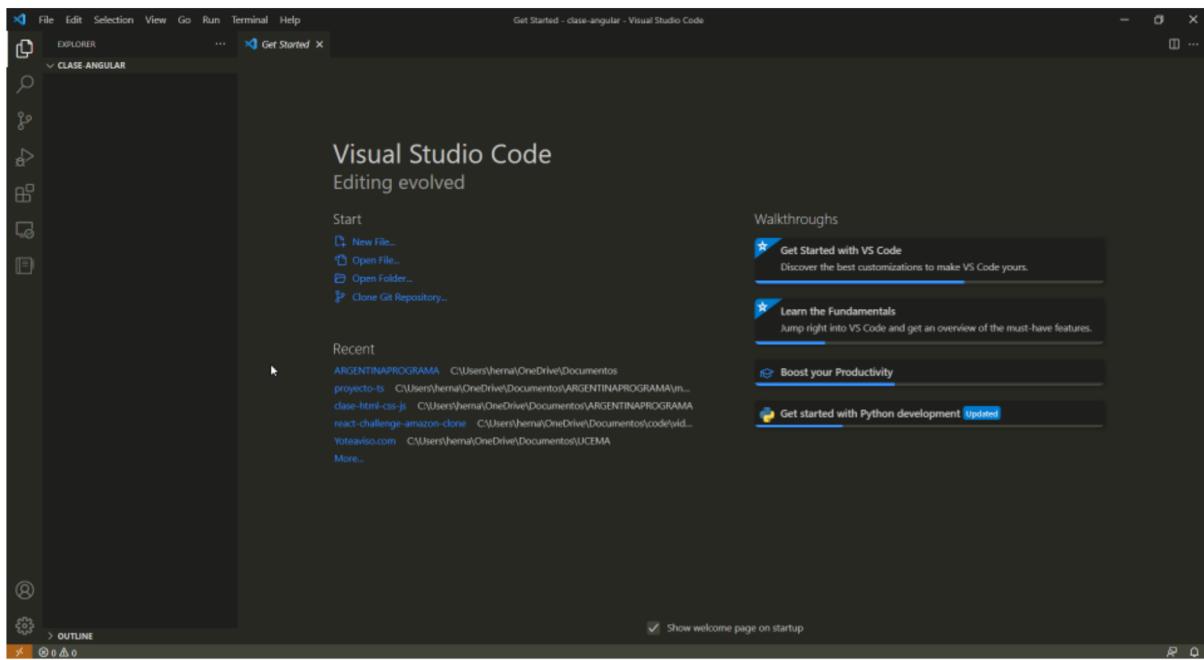
Instalar nodejs y Visual Studio Code

Página Oficial de Angular (todo en inglés solamente): <https://angular.io/>

Ejemplo completo subido en <https://github.com/hernanborre/mc-angular-tutorial>

Creamos una carpeta nueva llamada “clase-angular”

- Creamos una carpeta nueva donde queramos: (yo la voy a llamar “*clase-angular*”)
- Abrimos el VS Code y arrastramos y tiramos esta carpeta nueva en el VS Code (*deberías ver algo así*)

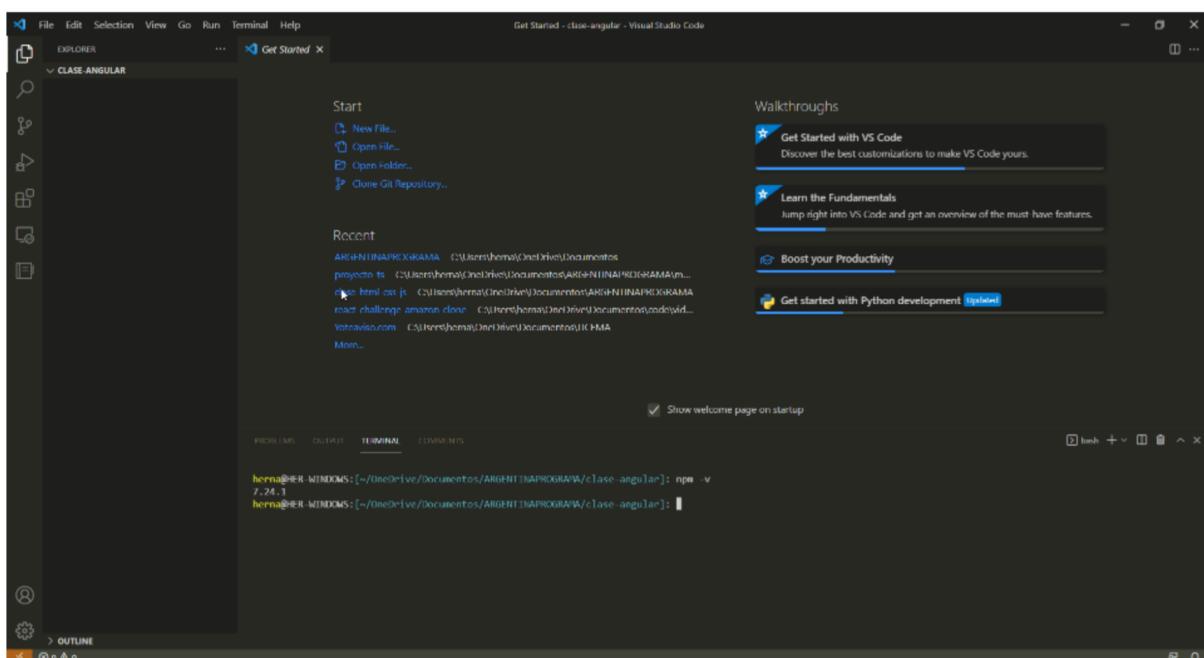


Abrimos una nueva Terminal y ejecutamos:

```
npm -v
```

Si esto nos devuelve un número de versión en vez de un error, podemos continuar (sino fijate que puede estar fallando)

(ejemplo de `npm` andando bien en tu Terminal)



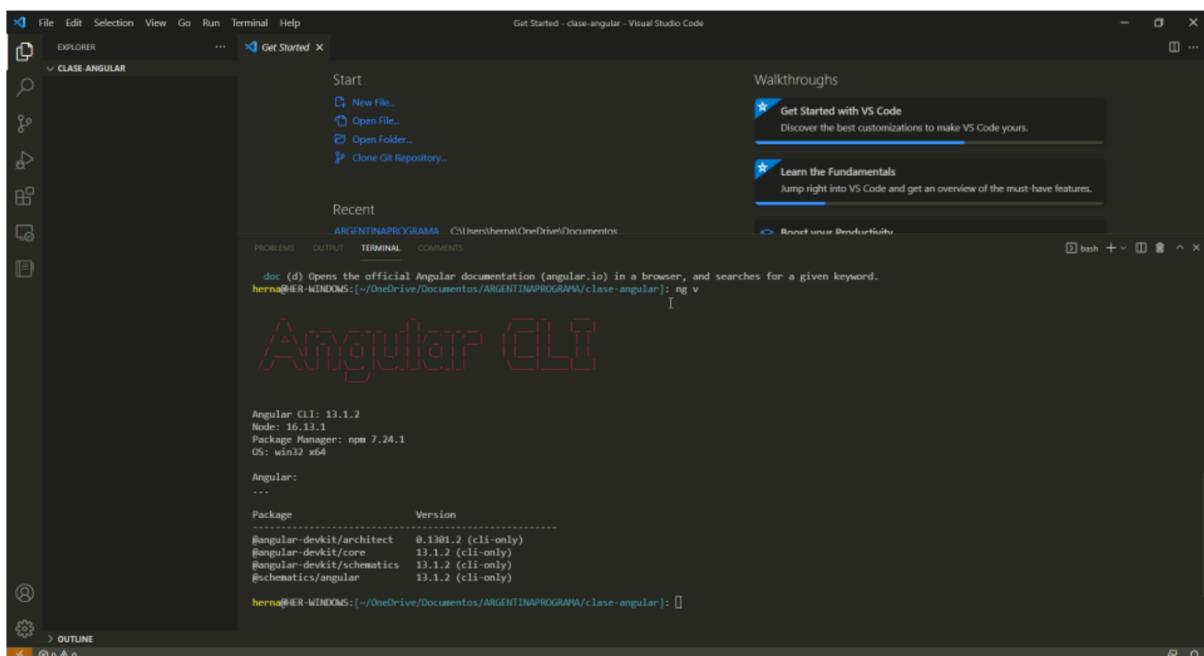
- Ahora si, vamos a instalar **Angular**:

```
npm install -g @angular/cli
```

Luego para asegurarnos que Angular se instaló bien, ejecutamos

```
ng v
```

(debería aparecer algo así)



A screenshot of the Visual Studio Code interface. The title bar says "Get Started - clase-angular - Visual Studio Code". The left sidebar shows a folder named "CLASE ANGULAR". The main area has a large title "Angular CLI" and displays the following text:

```
Angular CLI: 13.1.2
Node: 16.13.1
Package Manager: npm 7.24.1
OS: win32 x64

Angular:
...
Package           Version
@angular-devkit/architect    0.1301.2 (cli-only)
@angular-devkit/core         13.1.2 (cli-only)
@angular-devkit/schematics   13.1.2 (cli-only)
@schematics/angular          13.1.2 (cli-only)
```

- Ahora tenemos que hacer dos cosas, una es crear un nuevo proyecto de angular y la otra es hacer `cd` y meternos dentro de la carpeta recién creada (esto es MUY importante sino no te van a andar los siguientes comandos). Entonces:

```
ng new mi-primer-a-app
```

Le damo `y` a la opción del router y elegimos solo `css` en la segunda pregunta y listo!

```

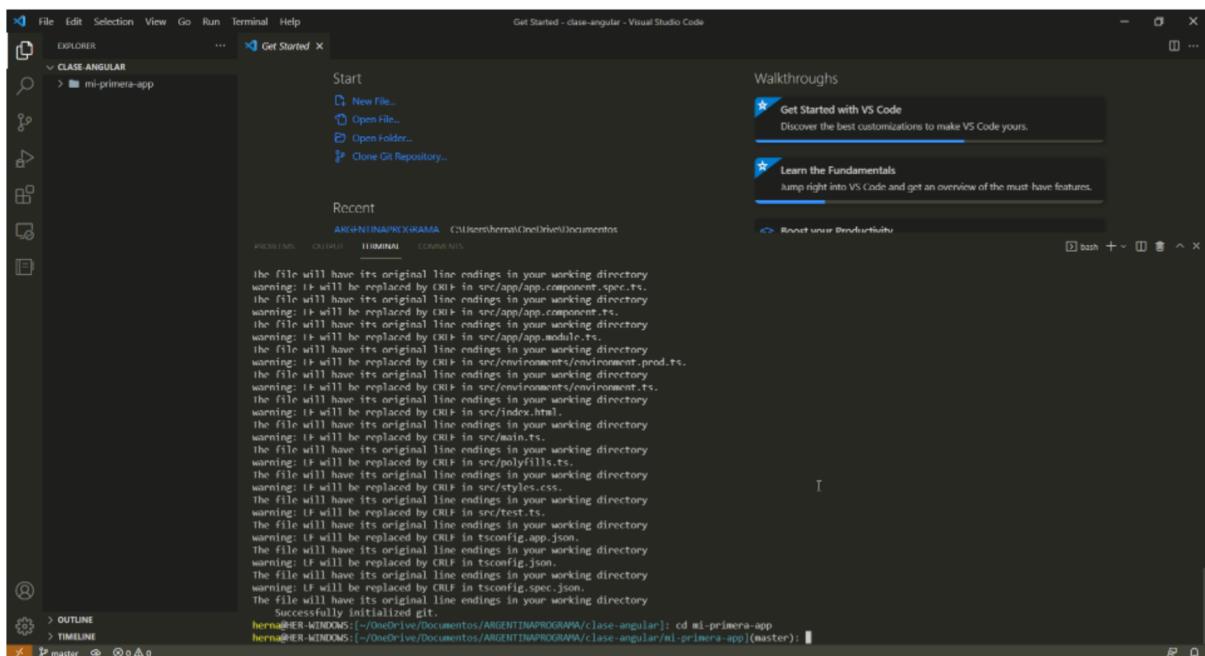
Angular CLI: 13.1.2
herma@HER-WINDOWS:~/OneDrive/Documentos/ARGENTINAPROGRAMA/clase-angular]: ng new mi-primer-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE mi-primer-app/angular.json (3087 bytes)
CREATE mi-primer-app/package.json (1078 bytes)
CREATE mi-primer-app/README.md (10958 bytes)

```

(esperamos un rato que se cree todo, puede tardar y luego ejecutarmos)

`cd mi-primer-app`

(deberíamos tener algo así)

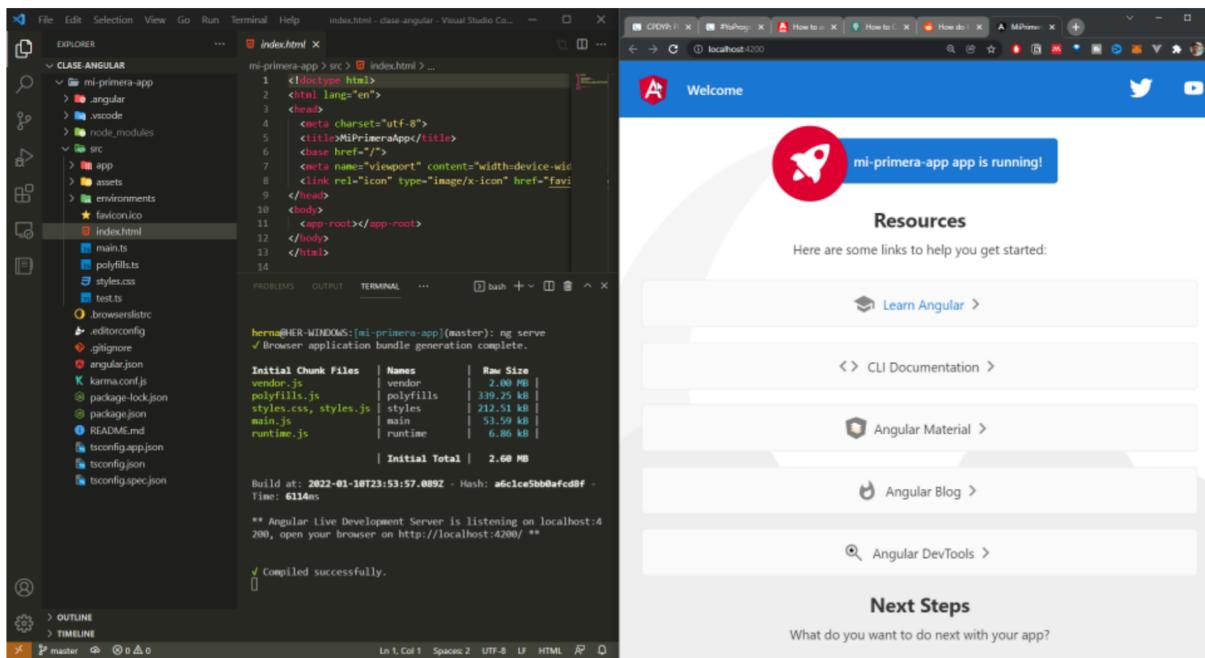


En la carpeta creada en el Explorer de VS Code, si abrís la nueva carpeta deberías ver varios archivos generados por angular!

Lo importante está dentro de la carpeta `src`

- Ahora ejecutemos la app a ver si está andando (puede tardar un poco, no te desesperes)

`ng serve`



Deberías ver algo similar a lo que hay acá arriba, además abrí en un browser de Chrome, la url que me indica Angular (<http://localhost:4200>)

- **Vamos a agregar Bootstrap 5, así ya nos queda listo para usar**

De dónde sacar Bootstrap 5?

Una de las opciones (la más rápida para practicar por ahora, es copiar y pegar los links a los CDN de Bootstrap 5, con lo cual, vamos a la página oficial y copiamos tanto los links a css como a los de js

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

(acordate que esto va pegado dentro de la etiqueta <head> de tu html principal)

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztQTwFspd3yD65VohpuuCOMLASjC" crossorigin="anonymous">
```

Esto va antes de cerrar la etiqueta </body> de tu html principal

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous"></script>
```

Listo! ya tenemos Bootstrap 5 en nuestro proyecto Angular integrado!

Listo! ahora podemos crear nuestro primer **componente**

Todo comienza en `app.module.ts` y esto nos dice que todo apunta a que su primer y único componente por ahora es el “AppComponent” así que vamos a ver qué tiene!

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CLASE-ANGULAR". The "src/app" folder contains "app-routing.module.ts", "app.component.html", "app.component.spec.ts", and "app.module.ts".
- Code Editor:** The active file is "app.module.ts", which contains the following code:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```
- Terminal:** Shows the build output:

```
Build at: 2022-01-11T00:05:30.328Z - Hash: a6c1ce5bb0afcd8f - Time: 80ms
✓ Compiled successfully.
```

Todo componente podría tener tres archivos principales (fijate más sobre esto en la página oficial de buenas prácticas y convenciones de Angular)

`app.component.ts`

`app.component.html`

`app.component.css`

Como te podrías imaginar, los de html y css son la vista (View) de nuestro componente llamado app en este caso! Y el código javascript (typescript) con la lógica y algunas cosas más, lo vamos a estar escribiendo en el archivo `.ts`

De la imagen anterior podemos inferir en 3 partes importantes que hacen a un componente en Angular y que se encuentran dentro del decorador `@Component()`:

1. `selector`, le dice a Angular que cree e inserte una instancia de este componente siempre que encuentre la etiqueta en el html. Por ejemplo, si el HTML de una aplicación contiene `<app-root></app-root>`, entonces Angular inserta una instancia de la vista HeaderComponent entre esas etiquetas.
2. `templateUrl`, la dirección relativa al template HTML del componente. Alternativamente, se puede escribir código html.
3. `stylesUrl`, la dirección relativa al archivo CSS del componente.

En pocas palabras, el selector es el nombre con el cual luego invocamos al componente en el html como se puede observar en el **index.html** en este caso!:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CLASE ANGULAR". The "src" folder contains "app" which has "app-routing.module.ts", "app.component.css", "app.component.html", "app.component.spec.ts", and "app.component.ts". Other files in "src" include "assets", "environments", "favicon.ico", "index.html", "main.ts", "polyfills.ts", "styles.css", "test.ts", and "tsconfig.json". Global configuration files like ".editorconfig", ".gitignore", and "karma.conf.js" are also visible.
- Code Editor:** The "app.component.ts" file is open, showing the following code:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mi primera app';
}
```
- Terminal:** Shows the output of a build process:

```
5 unchanged chunks
Build at: 2022-01-11T00:05:30.320Z - Hash: a6c1ce5bb8afcd8f - Time: 80ms
✓ Compiled successfully.
```
- Status Bar:** Shows "In 11, Col 1" and "TypeScript".

La manera más fácil y rápida de crear un componente es través de la consola de comandos, ayudados por angular cli!

```
ng generate component <nombre-del-componente> [opciones]
```

Entonces vamos a ejecutar en nuestra Terminal, estas dos líneas por separado

Una va a crear el componente personas y el otro el componente persona (ya te estarás imaginando qué va a hacer cada uno!)

```
ng generate component componentes/personas --skip-tests=true
```

```
ng generate component componentes/persona --skip-tests=true
```

También vamos a crear una carpeta nueva dentro de `src` que vamos a llamar `models` y adentro vamos a crear un nuevo archivo llamado `Persona.ts`

En persona, vamos a definir la clase Persona y va a ser muy importante para poder usar desde la vista y empezar a separar la Vista del Model (Model / View / Controller) También deberíamos hacer un Servicio Persona para tener así el Controller pero por el momento lo vamos a dejar para más adelante

(debería quedar algo así)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CLASE-ANGULAR". It includes the root folder "mi-primer-app" containing ".angular", ".vscode", "node_modules", "src", and "environments". Inside "src" are "app" and "models" folders. "models" contains "Persona.ts". Other files in "app" include "app-routing.module.ts", "app.component.css", "app.component.html", "app.component.spec.ts", "app.component.ts", "app.module.ts", "index.html", "main.ts", "polyfills.ts", "style.css", and "test.ts".
- Terminal:** Shows the command line output of two "ng generate component" commands:

```
herma@HER-WINDOWS:[mi-primer-app](master): ng generate component componentes/personas --skip-tests=true
CREATE src/app/componentes/personas/personas.component.html (23 bytes)
CREATE src/app/componentes/personas/personas.component.ts (283 bytes)
CREATE src/app/componentes/personas/personas.component.css (0 bytes)
UPDATE src/app/app.module.ts (495 bytes)

herma@HER-WINDOWS:[mi-primer-app](master): ng generate component componentes/persona --skip-tests=true
CREATE src/app/componentes/persona/persona.component.html (22 bytes)
CREATE src/app/componentes/persona/persona.component.ts (279 bytes)
CREATE src/app/componentes/persona/persona.component.css (0 bytes)
UPDATE src/app/app.module.ts (593 bytes)
```

Entonces dentro de `Persona.ts` vamos a crear su clase!

```
export class Persona {  
  
    nombre:string  
    apellido: string
```

```

edad: number

constructor(nombre:string = "" , apellido:string = "", edad:number = 0 ){
    this.nombre = nombre
    this.apellido = apellido
    this.edad = edad
}

cumplirAnios(){
    this.edad++
}
}

```

```

File Edit Selection View Go Run Terminal Help
EXPLORER Personas - class-angular - Visual Studio Code
mi-primer-app > src > app > models > Personas > Persona
1 export class Persona {
2
3     nombre: string
4     apellido: string
5     edad: number
6
7     constructor(nombre:string = "", apellido:string = "", edad:number = 0 ){
8         this.nombre = nombre
9         this.apellido = apellido
10        this.edad = edad
11    }
12
13    cumplirAnios(){
14        this.edad++
15    }
16

```

PROBLEMS OUTPUT TERMINAL COMMENTS

component.css (0 bytes)
UPDATE src/app/app.module.ts (495 bytes)
herma@IER-WINDOWS:[mi-primer-app](master): ng generate component componentes/persona --skip-tests=true
CREATE src/app/componentes/persona/persona.component.html (22 bytes)
CREATE src/app/componentes/persona/persona.component.ts (279 bytes)
CREATE src/app/componentes/persona/persona.component.css (0 bytes)
UPDATE src/app/app.module.ts (593 bytes)
herma@IER-WINDOWS:[mi-primer-app](master):

In 1. Col 1 (315 selected) Spaces: 4 UTF-8 CRLF {} TypeScript

Borramos absolutamente todo del

`app.component.html`

y vamos a escribir nuestra etiqueta del nuevo componente creado de Personas!

En nuestro `personas.component.html` vamos a escribir el siguiente código para mostrar una lista de personas usando la directiva de angular `*ngFor`

```

<div class="personas">
    <h3> Esta es nuestra lista de personas!</h3>

    <div *ngFor="let persona of personas">
        <h4>Nueva Persona</h4>
        <p>Nombre: {{persona.nombre}}</p>
        <p>Apellido: {{persona.apellido}}</p>

```

```

        <p>Edad: {{persona.edad}}</p>
    </div>
</div>

```

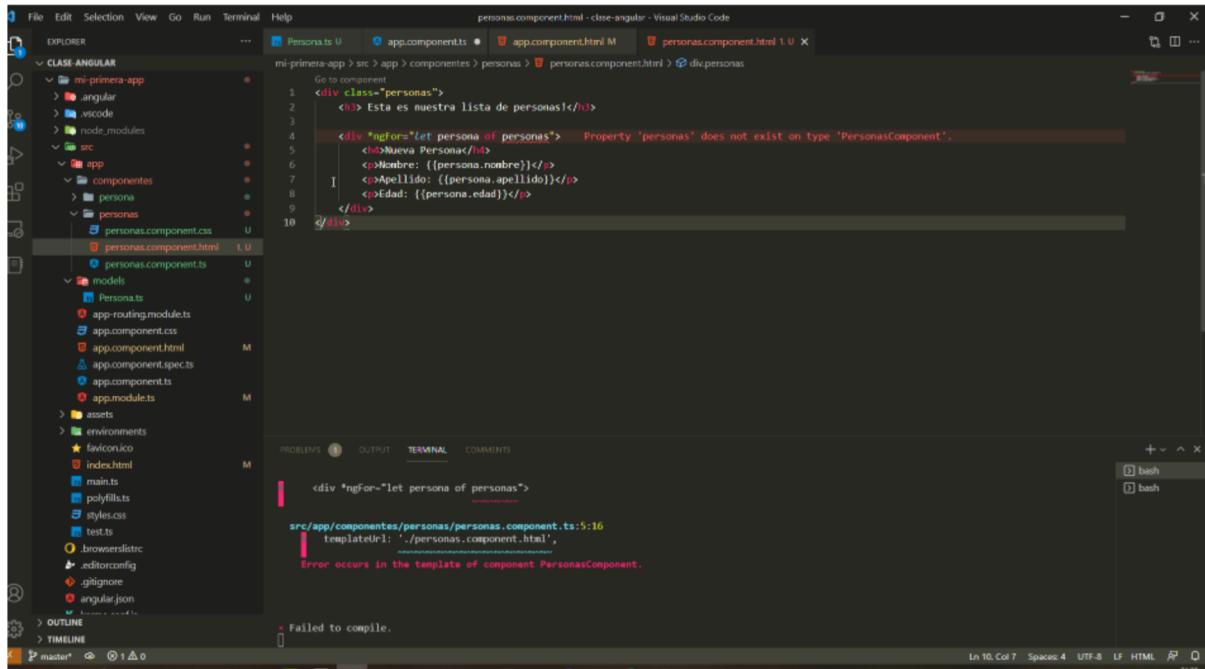
También en el `app.component.html` vamos a borrar todo y dejar solo el siguiente código (llamado al componente de personas y su selector definido

```

<div class="container">
    <app-personas></app-personas>
</div>

```

Todavía esto no va a andar porque necesitamos definir en el archivo de lógica del componente de dónde está saliendo esa lista o array de `personas` que le estamos pidiendo iterar en en el html, no?



Entonces en el `personas.component.ts` vamos a importarlo y definirlo, quedando así

```

import { Component, OnInit } from '@angular/core';
import { Persona } from 'src/app/models/Persona';

```

```

@Component({
  selector: 'app-personas',
  templateUrl: './personas.component.html',
  styleUrls: ['./personas.component.css']
})
export class PersonasComponent implements OnInit {

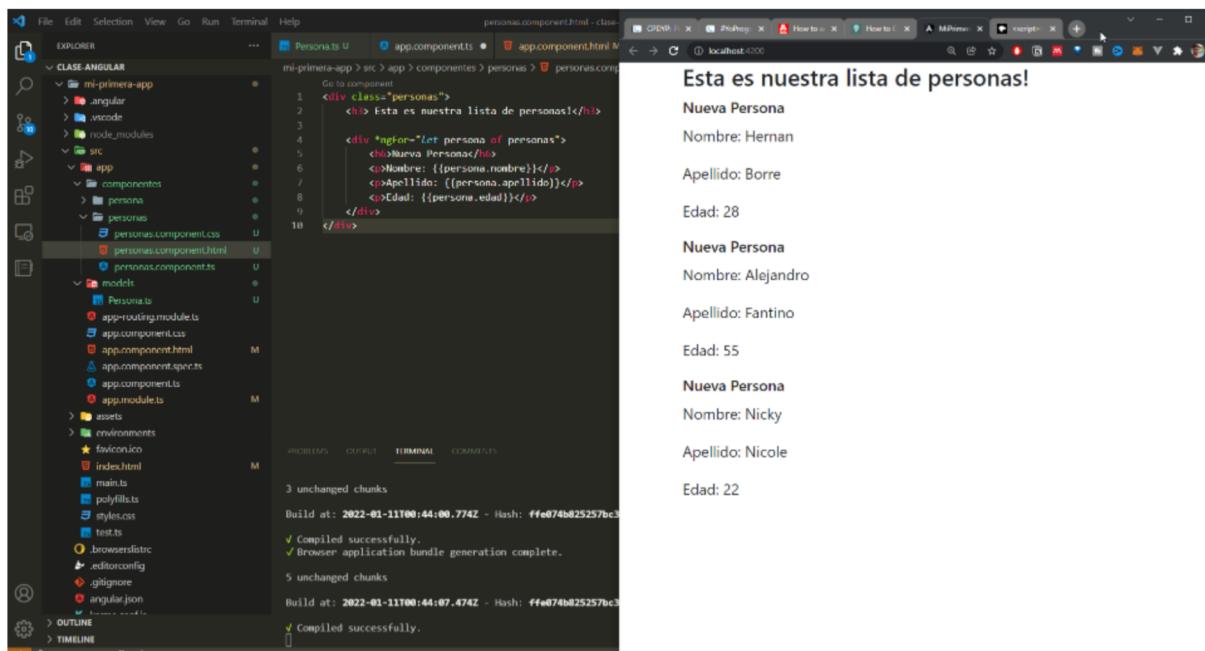
  personas: Persona[] = []

  constructor() { }

  ngOnInit(): void {
    let persona1 = new Persona("Hernan", "Borre", 28 )
    this.personas.push(persona1)
    this.personas.push(new Persona("Alejandro", "Fantino", 55 ))
    this.personas.push(new Persona("Nicky", "Nicole", 22 ))
  }
}

```

Genial ahora debería estar andando todo!!! 😊

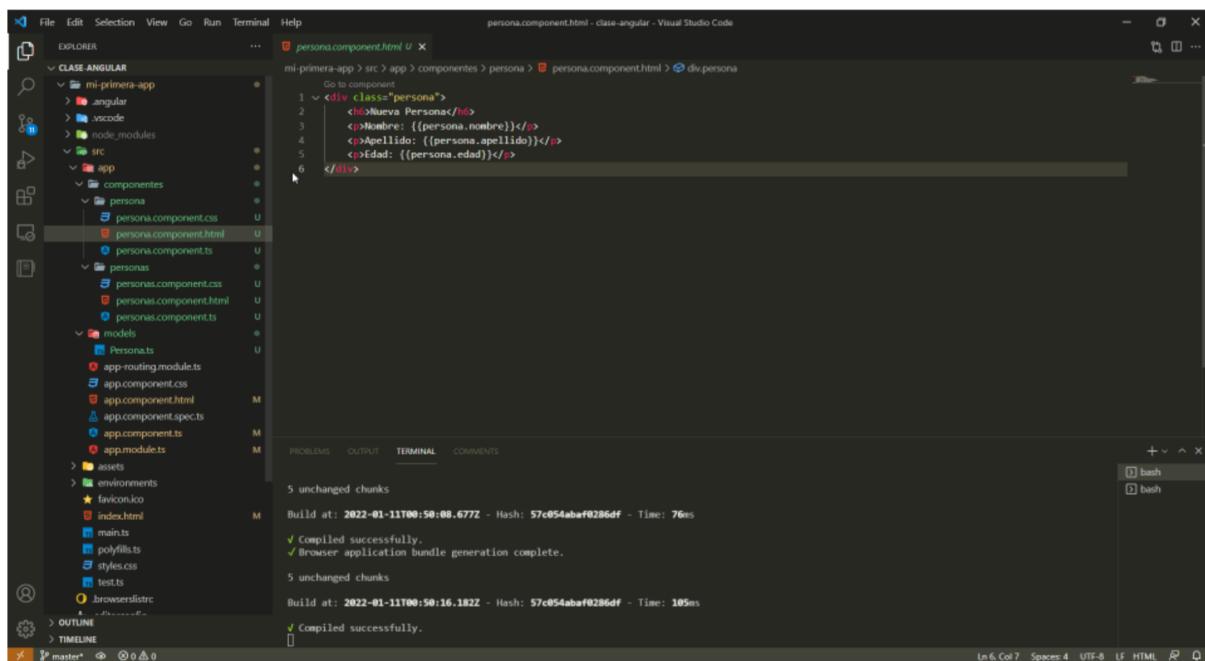


Ahora vamos a componentizar más esto y a poner nuestra funcionalidad de mostrar una persona, dentro de su propio componente en singular `persona.components.ts`

Te animás a hacer solo/a???

Acá te debe estar fallando un truco y nos introduce a otra cosa importantísima de Angular que es cómo pasar información (`props o properties`) de un componente padre a un hijo, eso es super fácil y se hace con el decorador de `@Input` delante de la variable de clase creada en nuestro componente hijo y así cuando llamamos a nuestro selector desde el padre, solo hacemos referencia a ese nombre definido como `Input`, mirá:

```
<div class="persona">
  <h6>Nueva Persona</h6>
  <p>Nombre: {{persona.nombre}}</p>
  <p>Apellido: {{persona.apellido}}</p>
  <p>Edad: {{persona.edad}}</p>
</div>
```



```
import { Component, Input, OnInit } from '@angular/core';
import { Persona } from 'src/app/models/Persona';

@Component({
  selector: 'app-persona',
  templateUrl: './persona.component.html',
```

```

    styleUrls: ['./persona.component.css']
  })
export class PersonaComponent implements OnInit {

  @Input() persona: Persona = new Persona()

  constructor() { }

  ngOnInit(): void {
  }

}

```

The screenshot shows the Visual Studio Code interface with the file `persona.component.ts` open in the editor. The code defines a component named `PersonaComponent` that injects a `Persona` service via the `@Input()` decorator. The component has a selector of `'app-persona'`, a template URL of `'./persona.component.html'`, and a CSS style URL of `'./persona.component.css'`. The code is part of a larger Angular application structure, with other files like `app-routing.module.ts`, `app.component.ts`, and `app.module.ts` visible in the Explorer sidebar.

Finalmente, nuestro componente padre `personas.component.html` va a quedar así (usando los corchetes para indicar que ese elemento será “pasado” a su componente hijo)

```

<div class="personas">
  <h3> Esta es nuestra lista de personas!</h3>

  <div *ngFor="let persona of personas">
    <app-persona [persona]="persona"></app-persona>
  </div>
</div>

```

The screenshot shows the Visual Studio Code interface with the 'personas.component.html' file open. The code displays a list of persons using an ngFor loop and an app-persona component. The terminal shows two successful build logs.

```

<div class="personas">
  <h3> Esta es nuestra lista de personas!</h3>

  <div *ngFor="let persona of personas">
    <app-persona [persona]="persona"></app-persona>
  </div>
</div>

```

Cómo venis? Ya falta poco!!!

Ahora vamos a intentar hacer lo que se conoce como *two-way binding* y para esto vamos a usar un input de html y una cosa especial de angular que es combinar los corchetes y los paréntesis, entonces!

En `persona.component.html`, agregamos el input

```
<input type="text" [(ngModel)]="persona.nombre">
```

(quedando así)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it displays the project structure for "mi-primer-app". The "CLASE ANGULAR" folder contains "src" and "app". The "app" folder has "components" and "models". "components" contains "personas" which has "personas.component.css", "personas.component.html", and "personas.component.ts". "models" contains "Personas.ts". Other files like "app-routing.module.ts", "app.component.css", etc., are also listed.
- Code Editor:** The main area shows the content of "personas.component.html".

```
<div class="persona">
  <h2>Nueva Persona</h2>
  <input type="text" [(ngModel)]="persona.nombre">
  <p>Nombre: {{persona.nombre}}</p>
  <p>Apellido: {{persona.apellido}}</p>
  <p>Edad: {{persona.edad}}</p>
</div>
```
- Terminal:** At the bottom right, there are two terminal tabs labeled "bash" and "bash".
- Status Bar:** At the bottom, it shows "Ln 7, Col 7" and "10/01/2020".

Y en el `app.module.ts` necesitamos agregar dentro de los imports el `FormsModule` así:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "CLASE ANGULAR". The "src/app" folder contains "components" and "models". "components" contains "personas" which has "person.component.ts", "person.component.html", and "person.component.css". "models" contains "person.ts". "app-routing.module.ts", "app.component.css", "app.component.html", "app.component.spec.ts", and "app.component.ts" are also listed.
- Code Editor (Center):** Displays the content of `app.module.ts`. The code defines the `AppModule` with declarations, imports, providers, and bootstrap.
- Terminal (Bottom):** Shows the build output:
 - 2 unchanged chunks
 - Build at: 2022-01-11T01:10:59.628Z - Hash: 2a87ac52566dd736 - Time: 298ms
 - ✓ Compiled successfully.
 - ✓ Browser application bundle generation complete.
 - 5 unchanged chunks
 - Build at: 2022-01-11T01:10:59.953Z - Hash: 2a87ac52566dd736 - Time: 118ms
 - ✓ Compiled successfully.
- Status Bar (Bottom):** Shows the current file is "master", the commit hash, and other system information like "In 20, Col 1 (16 selected) Spaces 2 UTF-8 LF () TypeScript R Q".

Ahora si! Deberíamos poder modificar y ver el cambio en vivo de los nombres de nuestras

personas con el input!!

The screenshot shows a web application running on localhost:4200. The title bar says "localhost:4200". The main content area has a heading "Esta es nuestra lista de personas!". Below it, there's a section for "Nueva Persona" with a form input containing "Nuevo nombre en vivo!!" and the message "Nombre: Nuevo nombre en vivo!!". There are also fields for "Apellido: Borre" and "Edad: 28". Another "Nueva Persona" section follows, with an input field containing "Alejandro" and the message "Nombre: Alejandro". It includes fields for "Apellido: Fantino" and "Edad: 55". A third "Nueva Persona" section shows an input field with "Nicky" and the message "Nombre: Nicky". It includes fields for "Apellido: Nicole" and "Edad: 22".

Para capturar eventos de los elementos html, los ponemos entre paréntesis y de la manera que angular nos indique en su documentación oficial.

Por ejemplo en el evento `onClick` se captura con `(click)`

Creemos un botón para cada persona que cuando se clique, llame a una función que a su vez llamará al método `cumplirAnios()` de la clase `persona`, es decir del objeto `persona` que estamos modificando. Y su edad se actualizará en vivo (reactividad)

- Agregamos en `persona.component.html`

```
<button class="btn btn-primary mb-3" (click)="festejo(persona)">Cumplir Años</button>
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "mi-primer-app". The "src/app/componentes/persona" folder is selected, displaying files like persona.component.css, persona.component.html, persona.component.ts, personas.component.css, personas.component.html, personas.component.ts, Persona.ts, app.routing.module.ts, app.component.css, app.component.html, app.component.spec.ts, app.component.ts, and app.module.ts.
- Editor:** The "persona.component.html" file is open, showing the following code:

```
<div class="persona">
  <h3>Nueva Persona</h3>
  <input type="text" [(ngModel)]="persona.nombre">
  <p>Nombre: {{persona.nombre}}</p>
  <p>Apellido: {{persona.apellido}}</p>
  <p>Edad: {{persona.edad}}</p>
  <button class="btn btn-primary mb-3" (click)="festeo(persona)">Cumplir Años</button>
</div>
```
- Terminal:** The terminal shows the build process and success message:

```
3 unchanged chunks

Build at: 2022-01-11T01:25:39.109Z - Hash: 360c87fd2a03ca2b - Time: 108ms

✓ Compiled successfully.
✓ Browser application bundle generation complete.

5 unchanged chunks

Build at: 2022-01-11T01:25:43.856Z - Hash: 360c87fd2a03ca2b - Time: 127ms

✓ Compiled successfully.
```

Y en `persona.component.ts` el método que estamos llamando desde el html (festejo). Vale aclarar que le pasamos la persona como parámetro así, nuestra aplicación sabe exactamente a qué persona estamos queriendo modificar su atributo (estado).

```
festejo(persona:Persona) {  
    persona.cumplirAnios()  
}
```

```

1 import { Component, Input, OnInit } from '@angular/core';
2 import { Persona } from 'src/app/models/Persona';
3
4 @Component({
5   selector: 'app-persona',
6   templateUrl: './persona.component.html',
7   styleUrls: ['./persona.component.css']
8 })
9 export class PersonaComponent implements OnInit {
10
11   @Input() persona: Persona = new Persona();
12
13   constructor() {}
14
15   ngOnInit(): void {
16
17     festear(persona: Persona) {
18       persona.cumpleAnios();
19     }
20   }
21
22 }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74

```

PROBLEMS OUTPUT TERMINAL COMMENTS

5 unchanged chunks

Build at: 2022-01-11T01:31:23.217Z - Hash: 360c07fd2a03ca2b - Time: 117ms

✓ Compiled successfully.

✓ Browser application bundle generation complete.

5 unchanged chunks

Build at: 2022-01-11T01:31:32.088Z - Hash: 360c07fd2a03ca2b - Time: 97ms

✓ Compiled successfully.

In 21, Col 1 Spaces: 2 UTF-8 LF () TypeScript

Bueno vamos por el Service! Lo vamos a crear dentro de la carpeta de services con la ayuda de angular, vamos a la Terminal de nuevo y ejecutamos:

```
ng generate service services/persona
```

Dentro de la nueva carpeta, services, en el archivo `persona.service.ts` debería quedar algo así:

```

import { Injectable } from '@angular/core';
import { Persona } from '../models/Persona';

@Injectable({
  providedIn: 'root'
})
export class PersonaService {

  personas: Persona[] = []
  constructor() {}

  getAllPersonas(): Persona[] {
    let persona1 = new Persona("Hernan", "Borre", 28 )
    this.personas.push(persona1)
    this.personas.push(new Persona("Alejandro", "Fantino", 55 ))
    this.personas.push(new Persona("Nicky", "Nicole", 22 ))

    return this.personas
  }
}

```

```
// solo un ejemplo de los servicios que se pueden ofrecer
removePersona(persona:Persona):void {
    // TODO implement logic to remove a Persona
}
}
```

Entonces `personas.component.ts` va a quedar así:

```
import { Component, OnInit } from '@angular/core';
import { Persona } from 'src/app/models/Persona';
import { PersonaService } from 'src/app/services/persona.service';

@Component({
  selector: 'app-personas',
  templateUrl: './personas.component.html',
  styleUrls: ['./personas.component.css']
})
export class PersonasComponent implements OnInit {

  personas: Persona[] = []

  constructor(private personaService:PersonaService) { }

  ngOnInit(): void {
    this.getAllPersonas()
    // let persona1 = new Persona("Hernan", "Borre", 28 )
    // this.personas.push(persona1)
    // this.personas.push(new Persona("Alejandro", "Fantino", 55 ))
    // this.personas.push(new Persona("Nicky", "Nicole", 22 ))
  }

  getAllPersonas():void {
    this.personas = this.personaService.getAllPersonas()
  }
}
```

'Bueno nos queda el decorador de `@output` para pasar eventos y datos desde nuestros componentes hijos a nuestros padres

Vamos a borrar una persona

Esto involucra cuatro pasos, agregar el botón de Eliminar en el componente hijo, agregar el `@output` en el mismo componente hijo y luego hacer la definición del mismo para que emita el evento hacia los componentes padres.

Por último debemos capturar el evento desde el componente padre (**personas**) en este tutorial y así finalmente, definir el evento capturado y hacer lo necesario con él desde el padre - en este caso será eliminar de la lista de todas las persona, la seleccionada para ser eliminada

Veamos como queda todo el código, en orden de implementación:

`persona.component.html`

```
<div class="persona">
  <h6>Nueva Persona</h6>
  <input type="text" [(ngModel)]="persona.nombre">
  <p>Nombre: {{persona.nombre}}</p>
  <p>Apellido: {{persona.apellido}}</p>
  <p>Edad: {{persona.edad}}</p>
  <button class="btn btn-danger mb-3" (click)="borrarPersona(persona)">Eliminar!</button>
  <button class="btn btn-primary mb-3" (click)="festejo(persona)">Cumplir Años</button>
</div>
```

`persona.component.ts`

```
import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
import { Persona } from 'src/app/models/Persona';

@Component({
  selector: 'app-persona',
  templateUrl: './persona.component.html',
  styleUrls: ['./persona.component.css']
})
export class PersonaComponent implements OnInit {

  @Input() persona: Persona = new Persona()
  @Output() deletePersona:EventEmitter<Persona> = new EventEmitter()

  constructor() { }

  ngOnInit(): void {
  }

  festejo(persona:Persona) {
    persona.cumplirAnios()
  }

  borrarPersona(personaParaBorrar:Persona) {
    this.deletePersona.emit(personaParaBorrar)
  }
}
```

```
    }  
}
```

personas.component.html

```
<div class="personas">  
  <h3> Esta es nuestra lista de personas!</h3>  
  
  <div *ngFor="let persona of personas">  
    <app-persona [persona]="persona"  
      (deletePersona)="borrarPersonaDeLista($event)"></app-persona>  
  </div>  
</div>
```

personas.component.ts

```
import { Component, OnInit } from '@angular/core';  
import { Persona } from 'src/app/models/Persona';  
import { PersonaService } from 'src/app/services/persona.service';  
  
@Component({  
  selector: 'app-personas',  
  templateUrl: './personas.component.html',  
  styleUrls: ['./personas.component.css']  
})  
export class PersonasComponent implements OnInit {  
  
  personas: Persona[] = []  
  
  constructor(private personaService: PersonaService) { }  
  
  ngOnInit(): void {  
    this.getAllPersonas()  
    // let persona1 = new Persona("Hernan", "Borre", 28 )  
    // this.personas.push(persona1)  
    // this.personas.push(new Persona("Alejandro", "Fantino", 55 ))  
    // this.personas.push(new Persona("Nicky", "Nicole", 22 ))  
  }  
  
  getAllPersonas():void {  
    this.personas = this.personaService.getAllPersonas()  
  }  
  
  borrarPersonaDeLista(personaParaBorrar: Persona) {  
    // con este truco borramos a la persona pero en realidad deberíamos  
    // llamar al service  
    this.personas = this.personas.filter(p => p.nombre !== personaParaBorrar.nombre)  
  }  
}
```

Ahora si, deberíamos poder borrar personas desde nuestra lista en la página que hemos creado! 😊