



# Introducción a ANGULAR 5

# Historia

- Validaciones con JS
- jQuery
- AJAX
- Librerías MVC (AngularJS, Knockout, etc)
- Web Components
- Angular 2

Angular 1.x



AngularJS

---

Angular 2

Angular 4

Angular 5



Angular

# Qué es Angular

- Framework
  - Librerías: Componentes, routing, Cliente Http, etc
  - Librerías opcionales: Angular Material
  - Herramientas: Angular CLI, VS Code
  - Multiplataforma: Browser, Electron, Ionic
- Lenguajes
  - Javascript, TypeScript, Dart (u otro lenguaje que compile en JS)

# Características

- Cross platform
  - Progressive Web Apps
  - Mobile
  - Desktop
  - S.O.
- Velocidad y Performance
  - Code generation
  - Universal (SPA)
  - Code splitting
- Productividad
  - Templates
  - Angular CLI
  - IDE's
- Full Development Story
  - Testing
  - Animation
  - Accessibility, Int., etc

# Características

- Herramientas y Librerías

- Angular CLI
- Augury
- Angular Material 2
- Etc.

- Cross-Platform Development

- Electron
- Ionic
- Native Script
- Windows (UWP)
- Vue.JS

# En que consiste una App de Angular

- Componentes (*clases TypeScript*)
  - Templates HTML + markup angularizado
  - CSS encapsulado
  - Código: Métodos y propiedades
- Servicios: Código sin representación visual
- Módulos: Agrupa Componentes y Servicios
- Directivas: Atributos de HTML custom, define acciones
- Pipes: Formatos de salida

# Angular CLI <https://cli.angular.io>

- Crea proyectos
- Instala librerías con NPM
- Crea partes de proyectos: componentes, servicios, etc.
- Ejecuta aplicaciones
- Bundling: Webpack
- Unit tests & end-to-end tests
- Lint: Buenas prácticas
- i18N: <https://angular.io/guide/i18n>



# Angular CLI

- `npm install -g @angular/cli` < instalo angular-cli
- `ng new mi-app` < creo un proyecto semilla
- `cd mi-app` < carpeta
- `ng serve --open` < Levanto un servidor web y  
ejecuto la app
- `http://localhost:4200`

# En caso de que de error en la etapa de NPM

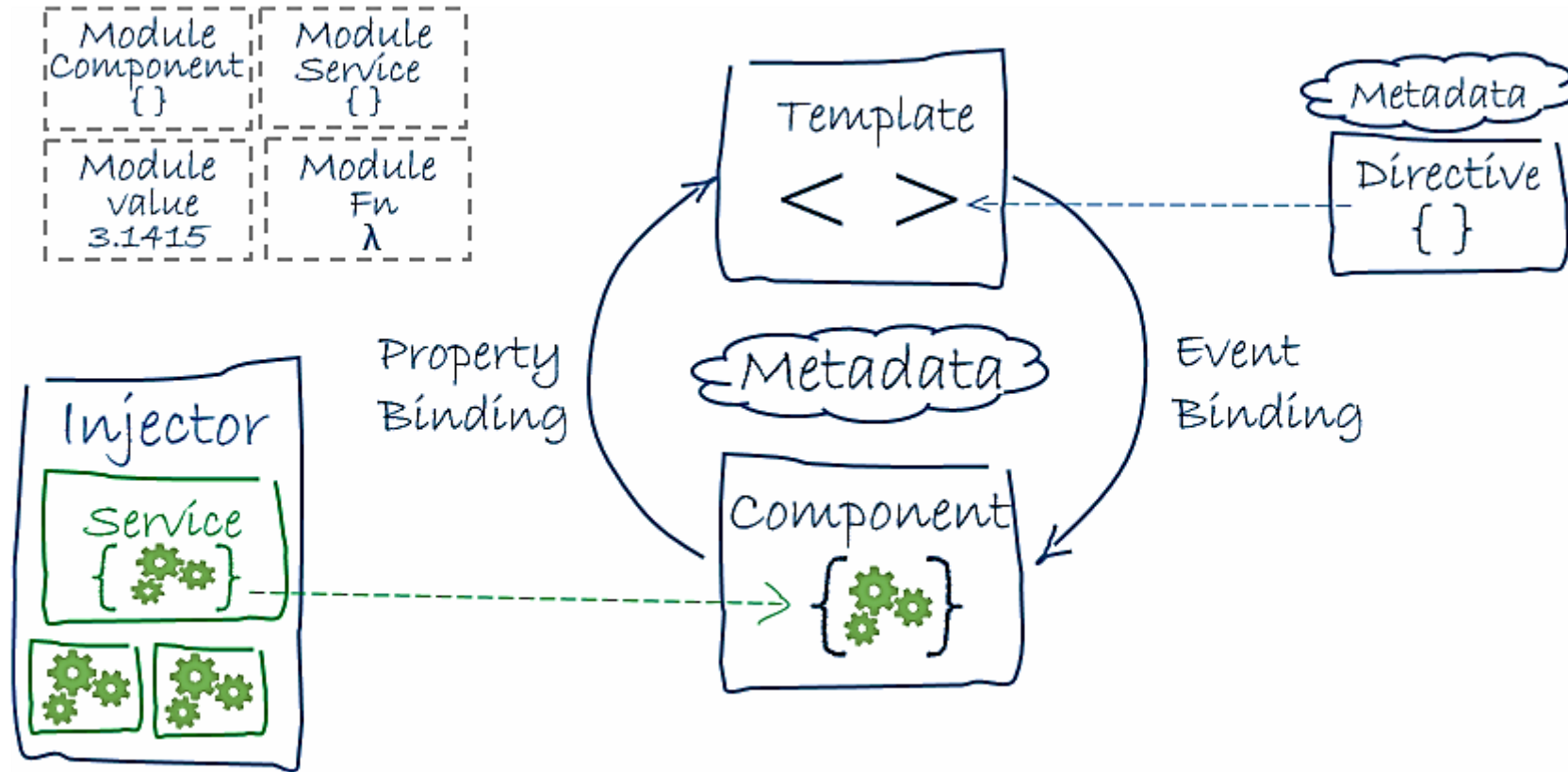
- Cd mi-app
- Npm install
- Npm cache clean --force

# NPM

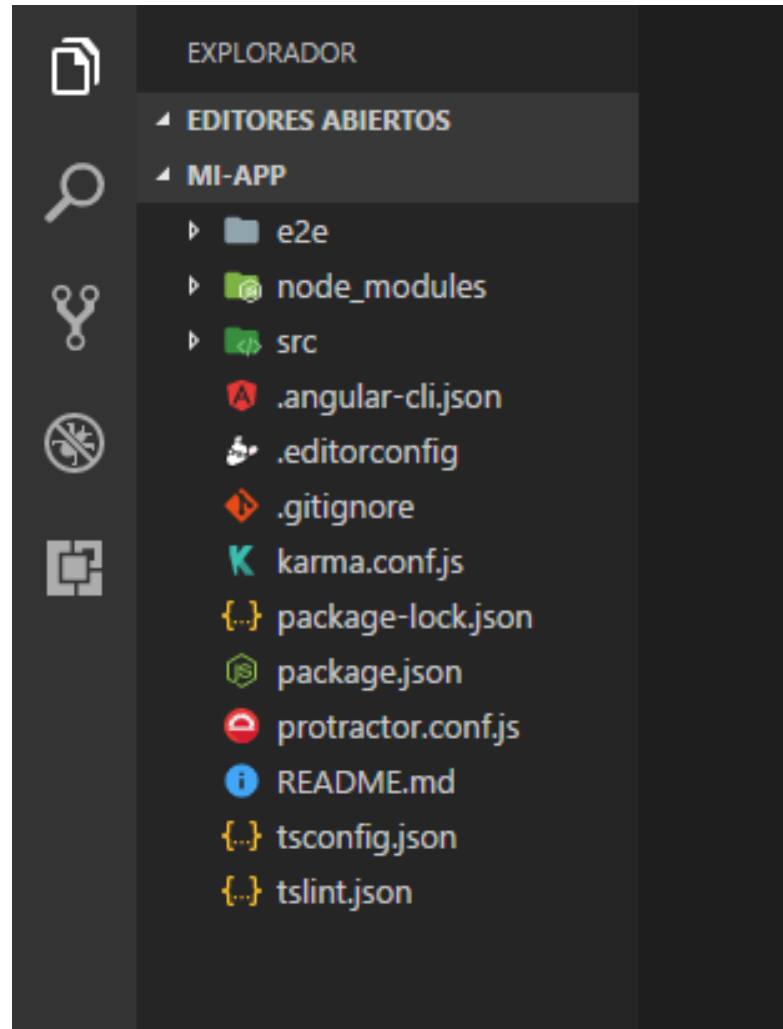
- Node Package Manager
- Instala paquetes de node
- <https://www.npmjs.com/> (Versiones)
- Almacena caché, no siempre baja todo
- Almacena las librerías en la carpeta del proyecto
  - Package.json -> node\_modules -> mantiene versiones
- Con -g almacena a nivel global

# Práctica 1 - Mi primera App con angular-cli

# En que consiste una App de Angular

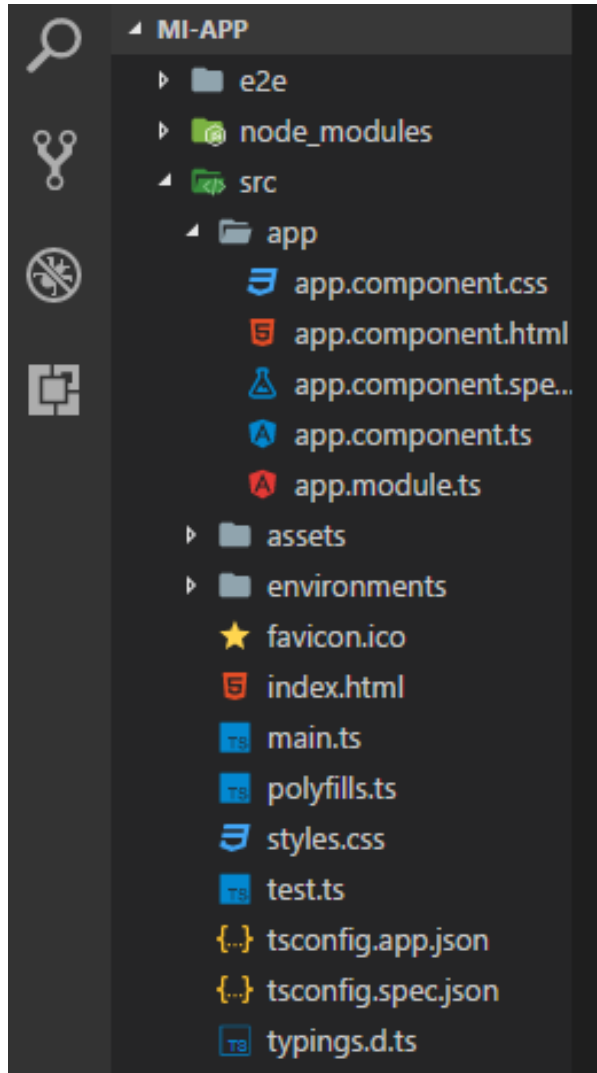


# Estructura del proyecto



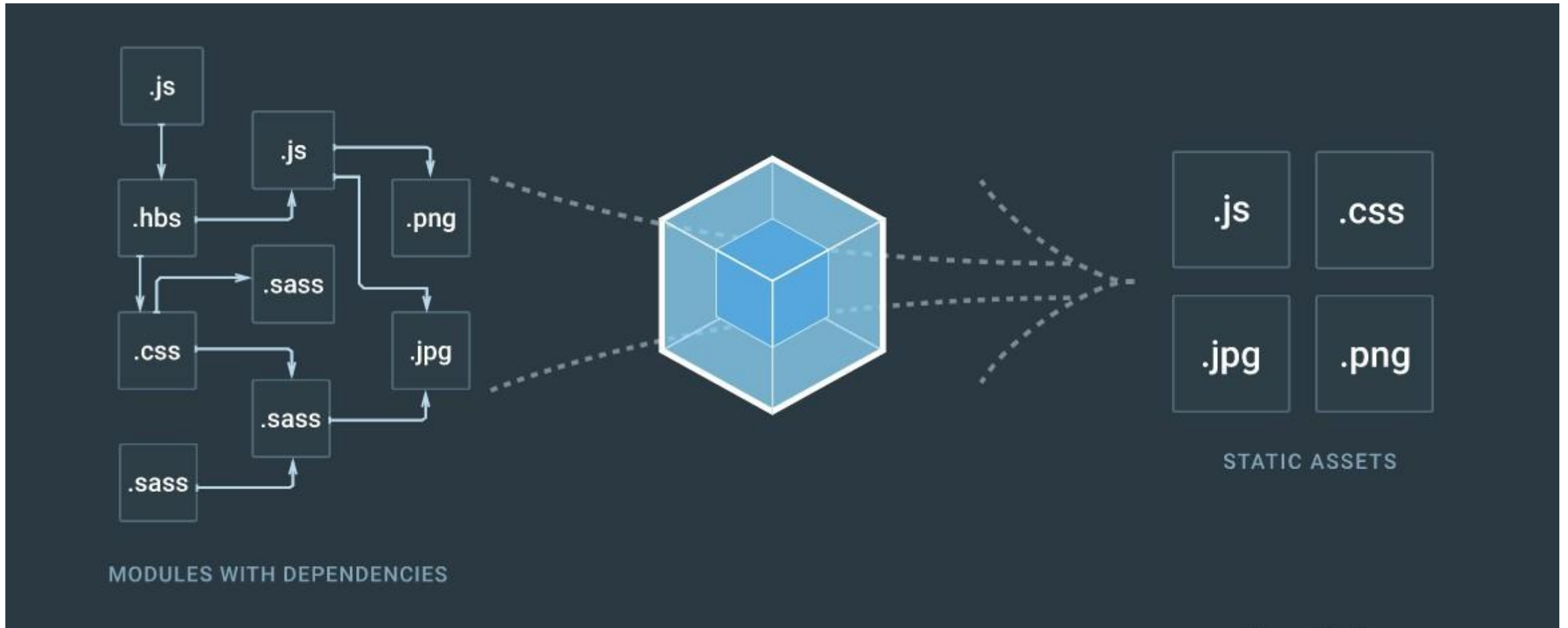
- Carpetas
  - e2e: end to end test
  - node\_modules: librerías
  - src: nuestra aplicación
- Archivos de configuración
  - package.json: configuración npm
  - karma y protractor: testing
  - tsconfig: Type Script
  - tslint: lint

# Estructura del proyecto: src



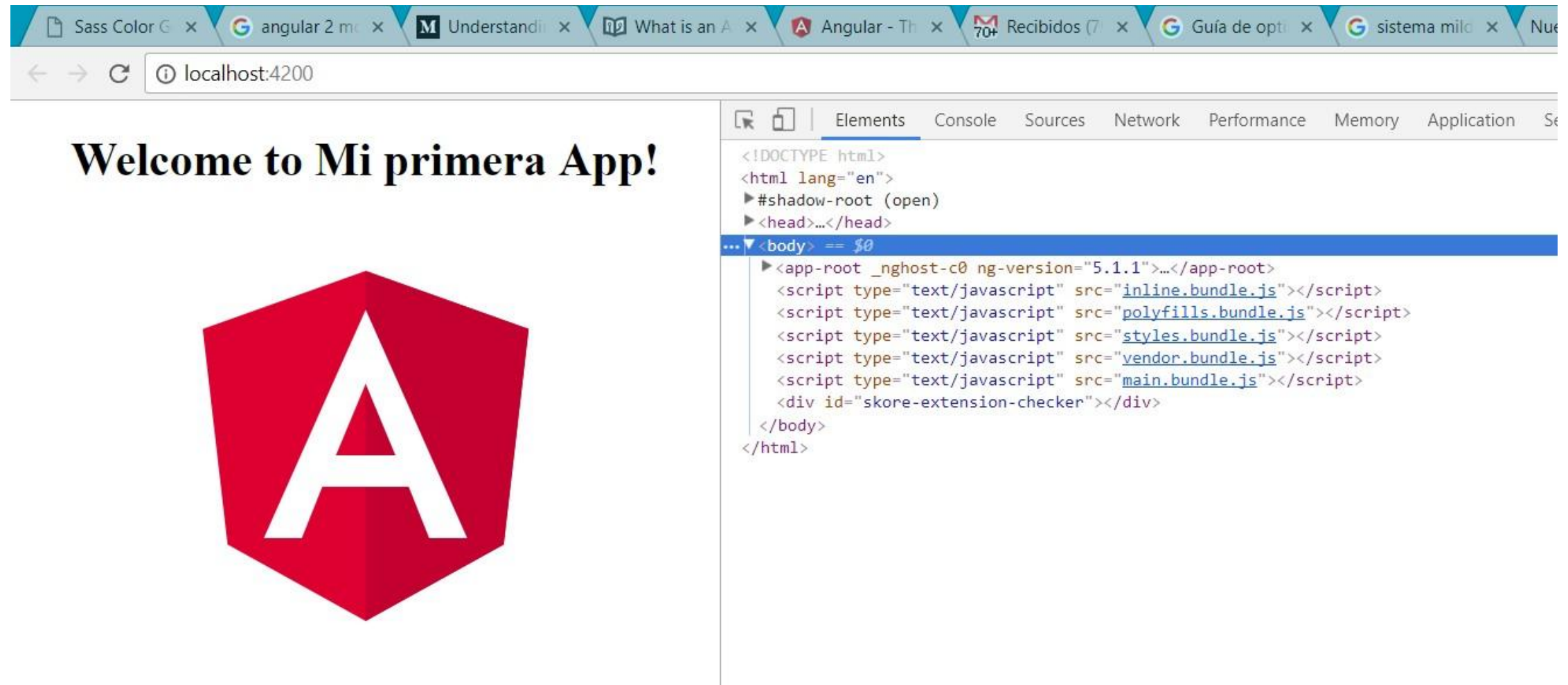
- app: Código fuente
- assets: css, js, íconos, fuentes, etc.
- Index.html: la única página html de nuestra aplicación
- styles.css: hoja de estilo de nuestra app
- test.ts
- main.ts: bootstarp de la app

# Angular CLI y Webpack






# Angular-cli y webpack



The screenshot displays a web browser window at `localhost:4200`. The main content area shows the text "Welcome to Mi primera App!" in a large, bold, black serif font. Below the text is the Angular logo, a red hexagon with a white letter 'A' in the center. To the right of the browser window, the Chrome DevTools 'Elements' panel is open, showing the DOM tree. The root element is `<html lang="en">`, which contains a `<body>` element. The `<body>` element contains an `<app-root _ngghost-c0 ng-version="5.1.1">` element. Inside the `<app-root>` element, there are five `<script>` tags for `inline.bundle.js`, `polyfills.bundle.js`, `styles.bundle.js`, `vendor.bundle.js`, and `main.bundle.js`, followed by a `<div id="skore-extension-checker">` element. The browser's address bar shows several open tabs, including "Sass Color G", "angular 2 m...", "M Understandi...", "What is an A...", "Angular - Th...", "Recibidos (7...", "Guía de opti...", "sistema milo...", and "Nue...".

localhost:4200

## Welcome to Mi primera App!



```
<!DOCTYPE html>
<html lang="en">
  <#shadow-root (open)>
    <head>...</head>
    <body> == %0
      <app-root _ngghost-c0 ng-version="5.1.1">...</app-root>
        <script type="text/javascript" src="inline.bundle.js"></script>
        <script type="text/javascript" src="polyfills.bundle.js"></script>
        <script type="text/javascript" src="styles.bundle.js"></script>
        <script type="text/javascript" src="vendor.bundle.js"></script>
        <script type="text/javascript" src="main.bundle.js"></script>
        <div id="skore-extension-checker"></div>
      </body>
    </html>
```

# Angular bootstrapping

main.ts

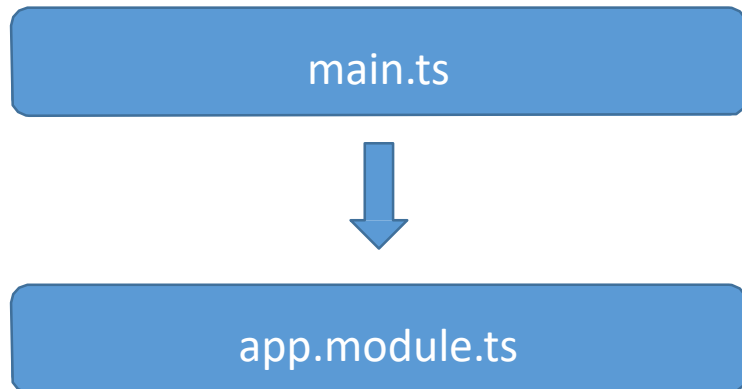
```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

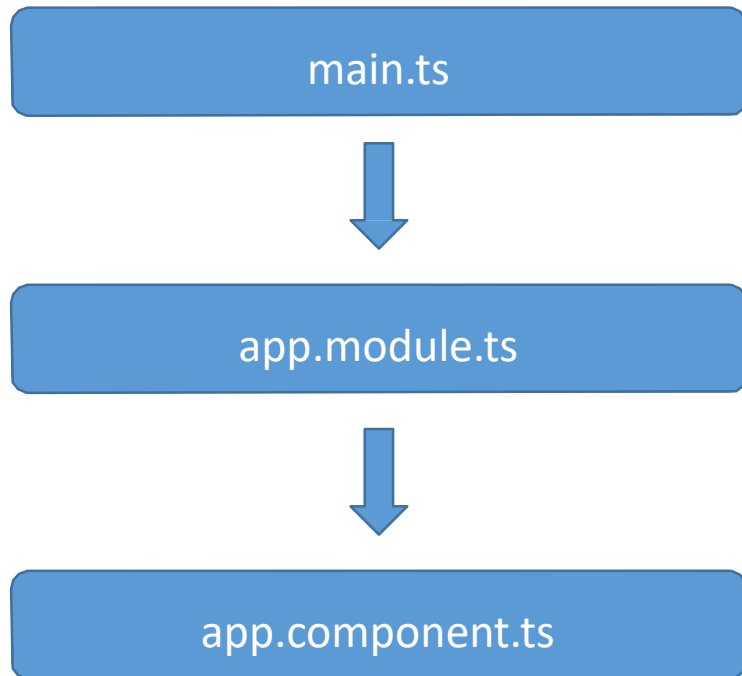
# Angular bootstrapping



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

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

# Angular bootstrapping

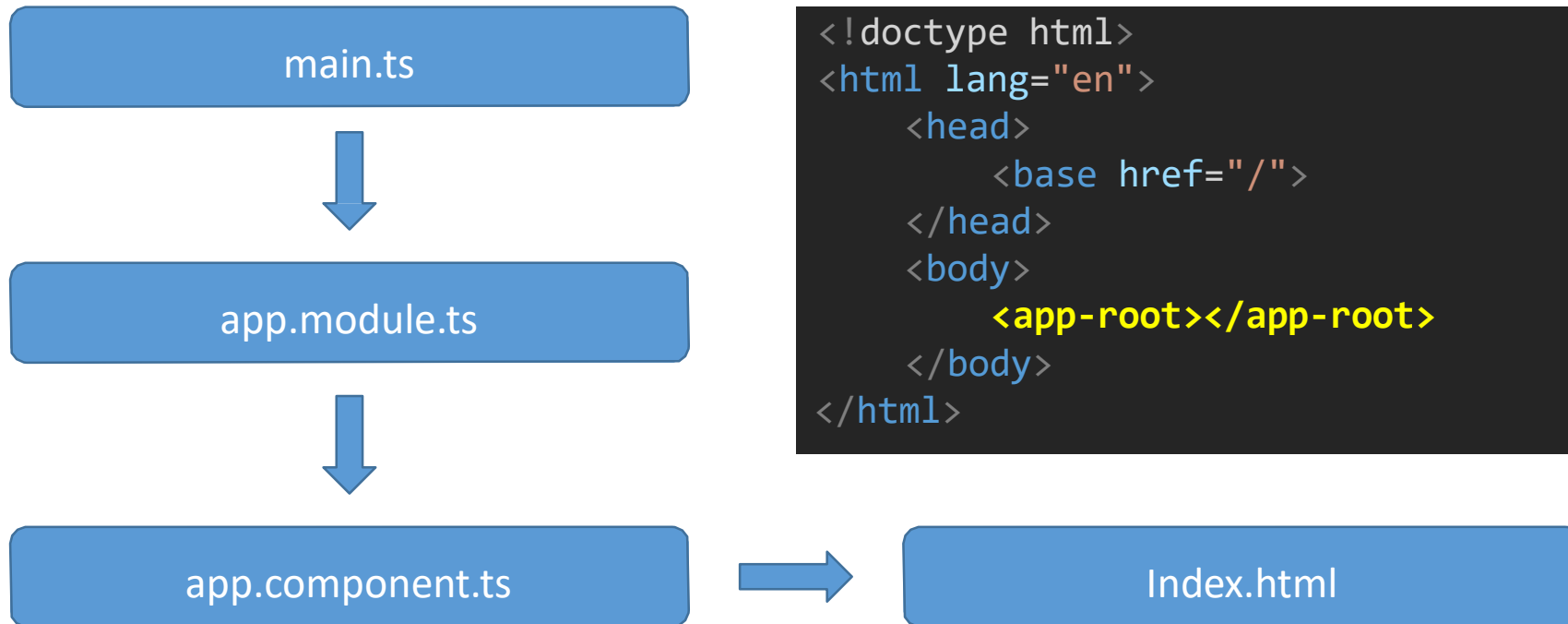


```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Mi primera App';
}
```

Desarrollaremos el tema componentes más adelante

# Angular bootstrapping



```
<!doctype html>
<html lang="en">
  <head>
    <base href="/">
  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

# Componente

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

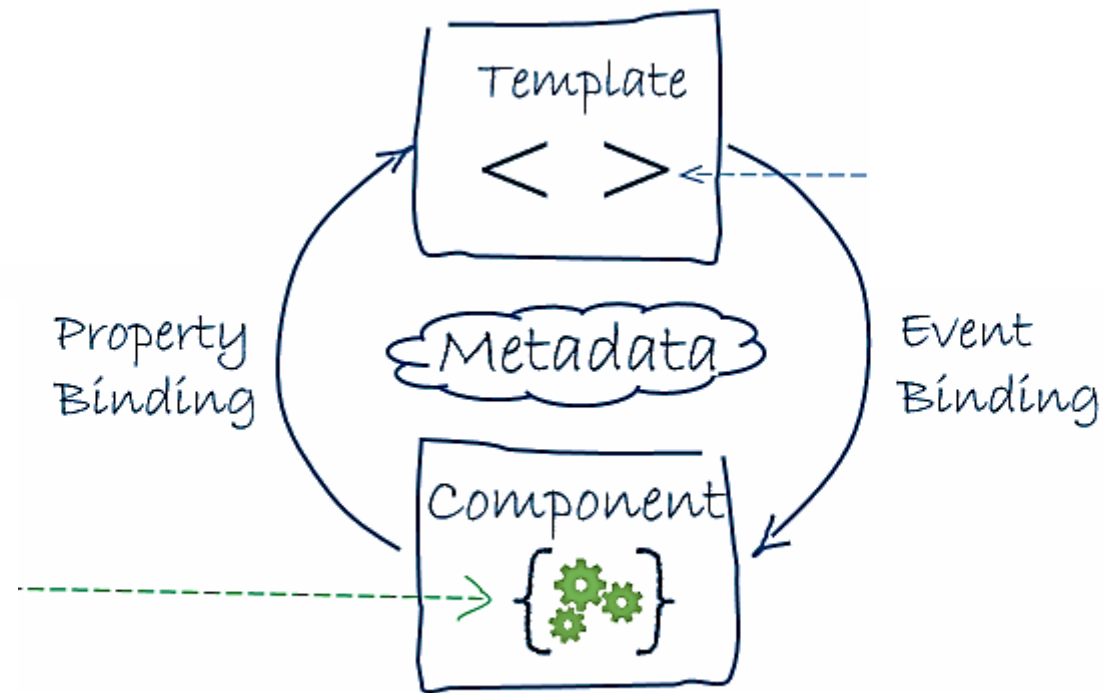
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Mi primera App';
}
```

# Introducción a TypeScript

```
export class AppComponent {  
  alumno: Alumno = {id: 1, nombre: 'Juan', apellido: 'Perez', sexo: 1, perfil: 0,  
    activo: true};  
  
  title = 'app';  
  
  SexoDescripcion(): string {  
    return Alumno.sexos[this.alumno.sexo];  
  }  
  
  PerfilDescripcion(): string {  
    return Alumno.perfiles[this.alumno.perfil];  
  }  
}
```

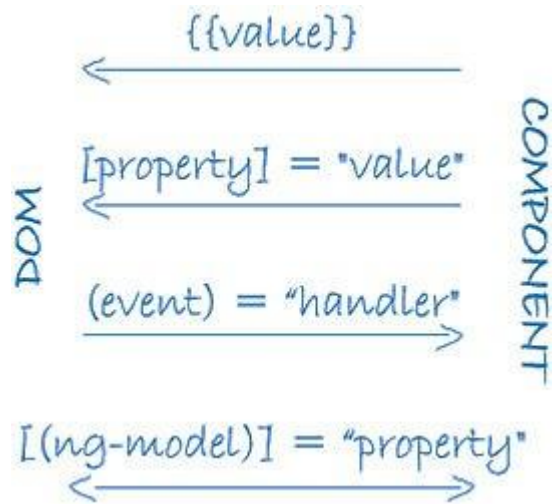
```
export class Alumno {  
  
  static perfiles = ['Desarrollador', '  
    'Operaciones', 'Power User'];  
  static sexos = ['Mujer', 'Hombre', '0'  
  
  id: number;  
  nombre: string;  
  apellido: string;  
  sexo: number;  
  perfil: number;  
  activo: boolean;  
  
}
```

# Binding de datos





# Binding: Data flow



- One way
  - Desde el origen de los datos al template o vista: interpolación
- One way
  - Desde el template o vista al componente: eventos
- Two way

# One way: Template expressions - Interpolation

```
<h3>
```

```
  {{title}}
```

```
  
```

```
</h3>
```

```
<!-- "The sum of 1 + 1 is 2" -->
```

```
<p>The sum of 1 + 1 is {{1 + 1}}</p>
```

```
<p>The sum of 1 + 1 is {{getSum(1,1)}}</p>
```

title, heroImageUrl y 1 + 1 son expresiones que Ng evalúa y convierte a un string

# Template expresions

- Sintaxis no permitida:
  - asignaciones: `=` , `+=`, `++`, etc.
  - `new`
  - Encadenamiento de expresiones: `;` o `,`
  - Operaciones bitwise: `|` y `&`
- Tampoco se puede
  - Llamadas locales: `windows`, `document`
  - `console.log` , `math.max` , etc
- Se permite referencia a
  - Funciones y propiedades del componente
  - Variables dentro del template (lo vemos más adelante)

# Template expressions: recomendaciones

- No debería tener efectos visibles colaterales
  - one way data binding
- Ejecución rápida
  - Se evalúan más frecuentemente de lo que uno imagina – keypress, click, etc.
- Simplicidad
  - Propiedades, funciones simples, quizás un ! pero no más que eso
- Idempotence
  - Retorna siempre el mismo valor a menos que uno de los valores subyacentes cambie

# Practica #2

- Definiremos una clase Alumno
- Crearemos una instancia en nuestro componente
- Lo mostraremos utilizando One Way Data Binding

# One Way Data Binding: Eventos

```
<button (click)="onSave()">Save</button>
```

Todos los eventos que expone el DOM pueden ser atrapados

# Two way data binding

```
<input [value]="currentHero.firstName"  
      (keyup)="ChangeName($event.target.value)">
```

Sintaxis `[( )]` : Una banana dentro de una caja

```
<input [(ngModel)]="currentHero.firstName">
```

# Built-in directives

- Attributes Directive
  - ngClass
  - ngStyle
- Structural Directive
  - ngIf
  - ngSwitch
  - ngFor



# ngStyle

```
<div [style.font-size]="isSpecial ? '20px' : '16px'" >
```

This div is x-large.

```
</div>
```

```
<div [ngStyle]="setStyles()">
```

This div is italic, normal weight, and extra large (24px).

```
</div>
```

```
setStyles() {  
  let styles = {  
    // CSS property names  
    'font-style': this.canSave ? 'italic' : 'normal', // italic  
    'font-weight': !this.isUnchanged ? 'bold' : 'normal', // normal  
    'font-size': this.isSpecial ? '24px' : '8px', // 24px  
  };  
  return styles;  
}
```

# ngClass

<!-- toggle the "special" class on/off with a property -->

<div [class.special]="isSpecial">The class binding is special</div>

isSpecial retorna un valor Boolean

<div [ngClass]="setClasses()">This div is saveable and special</div>

```
setClasses() {  
  let classes = {  
    saveable: this.canSave, // true  
    modified: !this.isUnchanged, // false  
    special: this.isSpecial, // true  
  };  
  return classes;  
}
```

# ngIf

```
<div *ngIf="currentHero">Hello, {{currentHero.firstName}}</div>
```

Exclusión del DOM tree

No olvidarse del \*

# ngSwitch

```
<span [ngSwitch]="toeChoice">  
  <span *ngSwitchCase="'Eenie'">Eenie</span>  
  <span *ngSwitchCase="'Meanie'">Meanie</span>  
  <span *ngSwitchCase="'Miney'">Miney</span>  
  <span *ngSwitchCase="'Moe'">Moe</span>  
  <span *ngSwitchDefault>other</span>  
</span>
```

# ngFor

```
<ul>
```

```
  <li *ngFor="let hero of heroes; let i=index">
```

```
    {{i + 1}} - {{hero}}
```

```
  </li>
```

```
</ul>
```

```
export class AppComponent {  
  heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];  
}
```

# Introducción a Angular Material 2



# Componentes de Angular Material 2

- autocomplete
- button
- button-toggle
- cards
- checkbox
- chips
- data-table
- datepicker
- dialog
- expansion-panel
- grid-list
- icon
- input
- list
- menu
- paginator
- progress-bar
- progress-spinner
- radio
- ripples
- select
- sidenav
- slide-toggle
- slider
- snackbar / toast
- sort-header
- stepper
- tabs
- textarea
- toolbar
- tooltip

# Práctica #3

- Uso de Angular Material 2
- Eventos
- Uso de directivas ngIf / ngFor / ngClass



# Componente padre – componente hijos

```
<mat-toolbar> ... </mat-toolbar>
```

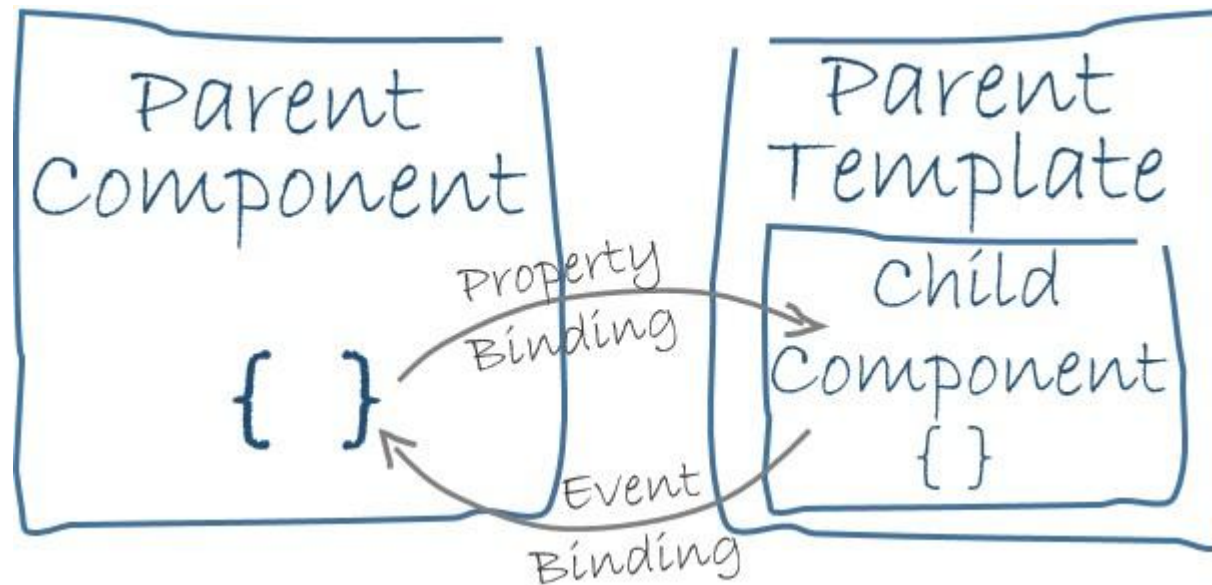
```
<div class="content">
```

```
  <alumnos-lista> </alumnos-lista>
```

```
  <alumno-edit> </alumno-edit>
```

```
</div>
```

# Data Binding: Parent – Child components



# Property Binding @Input – en el padre

```
<mat-toolbar> ... </mat-toolbar>
```

```
<div class="content">
```

```
  <alumnos-lista [alumnos] = "alumnos" > </alumnos-lista>
```

```
  <alumno-edit [alumno] = "alumnoSeleccionado">
```

```
  </alumno-edit>
```

```
</div>
```

# Property Binding @Input – en el hijo

```
import { Component, Input } from '@angular/core';  
....  
export AlumnosListaComponent {  
    @Input() alumnos: Alumnos[];  
}
```

# Componente hijo: capturar cuando se modifica una propiedad desde el padre

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'name-child',
  template: '<h3>{{name}}</h3>'
})
export class NameChildComponent {
  private _name = '';
  @Input()
  set name(name: string) {
    this._name = (name && name.trim()) || '<no name set>';
  }
  get name(): string { return this._name; }
}
```

# Eventos del hijo al padre – En el hijo

```
import { Component, EventEmitter, Input, Output } from '@angular/core';  
...
```

En el componente

```
Seleccionar(id: number) {  
    @Output() onSeleccion= new EventEmitter<number>();  
....  
    Seleccion(id: number) {  
        this.onSeleccion.emit(id);  
    }  
}
```

# Eventos del hijo al padre – En el padre

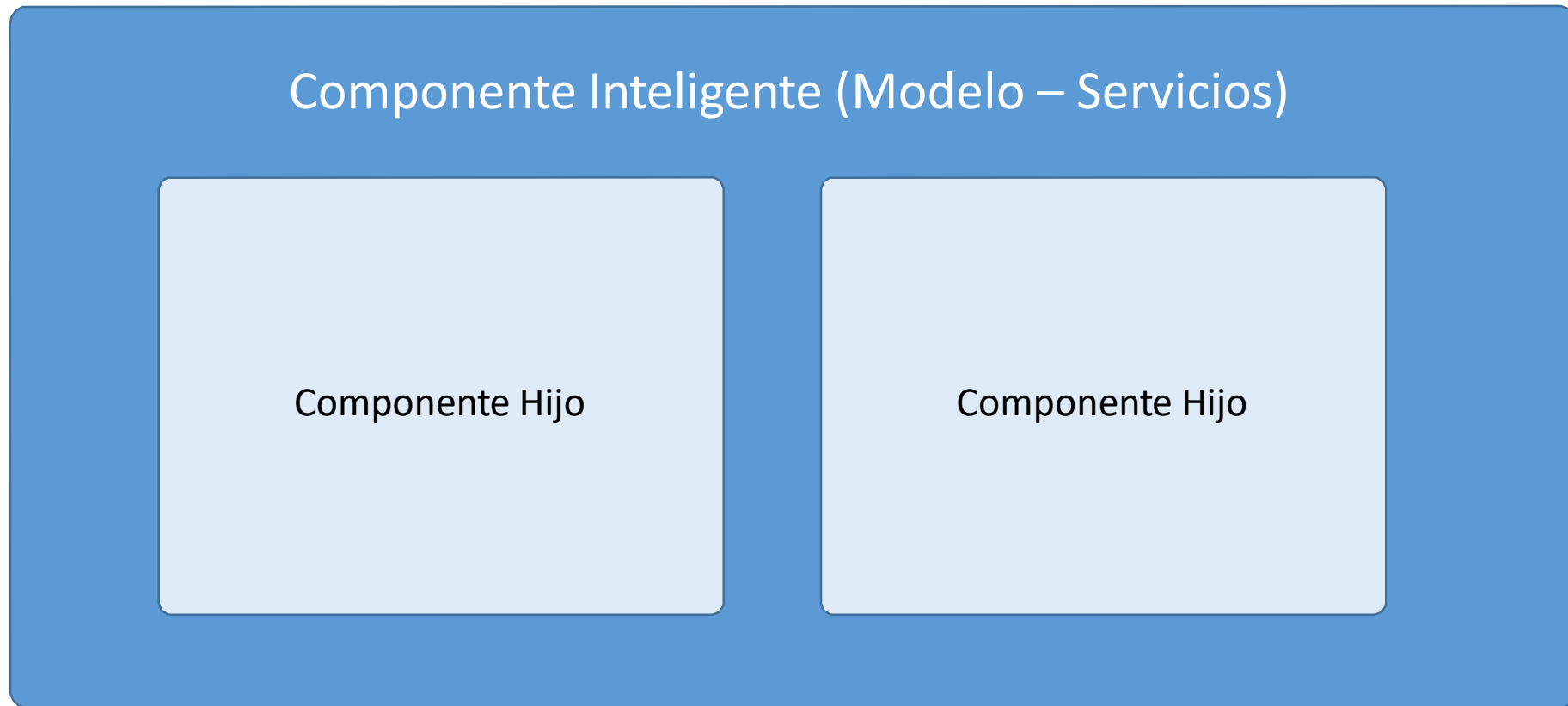
```
<h1>Gestión de Alumnos</h1>
<alumnos-lista [alumnos] = "alumnos"
                    (onSeleccion)="Seleccionar($event)">
</alumnos-lista>
<alumno-edit [alumno] = "alumnoSeleccionado"> </alumno-edit>
```

En el componente

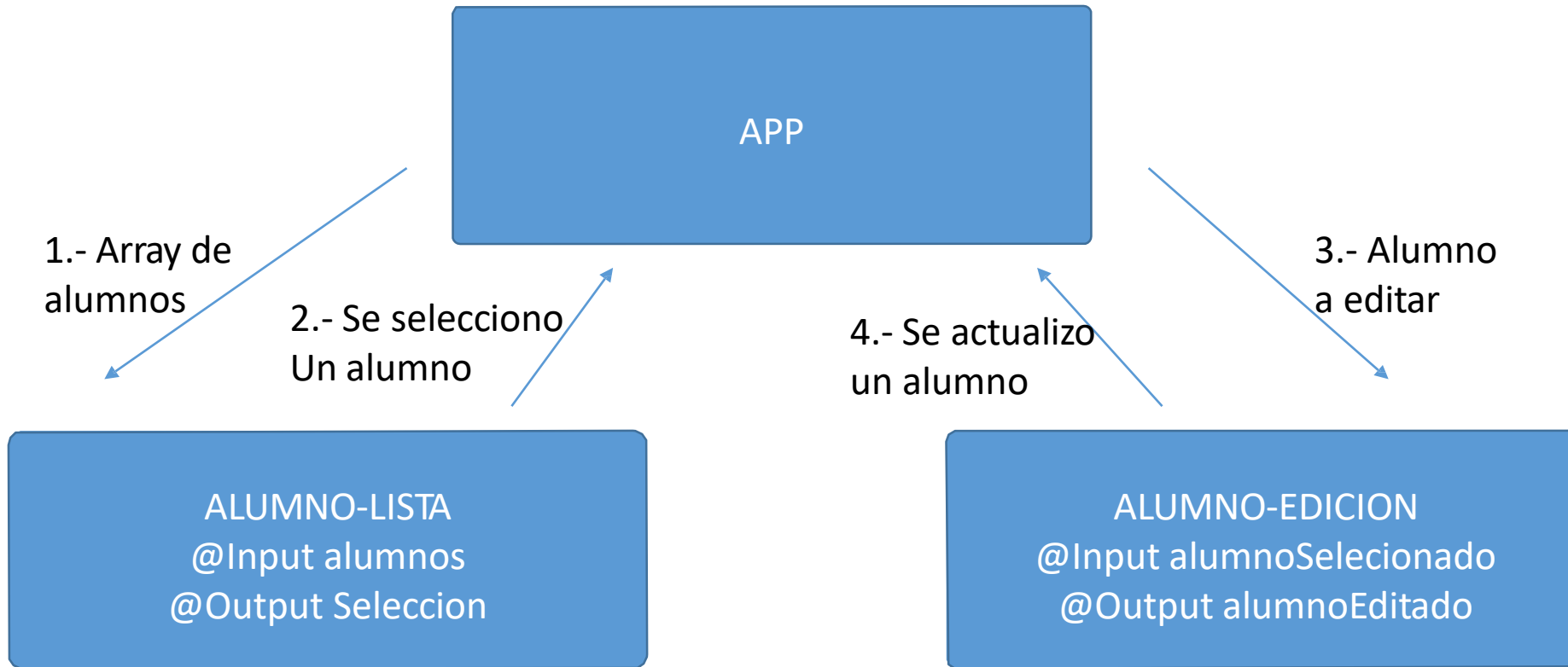
```
Seleccionar(id: number) {

}
```

# Patrón de Diseño – Componente padre Inteligente – Componente hijos simples







# Agregar un componente nuevo

1. Crear los archivos del componente (.html, .ts y .css) dentro de una carpeta
2. Agregar el componente al módulo de la aplicación, dentro del Array declaration
3. Incluir el tag del componente en el html del componente padre

## O con Angular-CLI:

1. `ng generate component alumnos-edit`
2. Incluir el tag del componente en el html del componente padre

¿Cómo hago para acceder directamente a las propiedades y métodos del componente hijo?

- En forma directa no es posible
- @ViewChild()

```
import { Component, ViewChild } from
 '@angular/core';
import { ABMComponent } from './abm.component';

@Component({
})
export class GestorComponent{
  @ViewChild(ABMComponent) ABMComponent:
  ABMComponent;

  constructor() { }

  ngAfterViewInit() {
    this.mostrar();
  }

  mostrar() {
    this.ABMComponent.accion = 'M';
    this.ABMComponent.entidad = 'Alumno';
    this.ABMComponent.modelo = this.alumno;
    this.ABMComponent.Show();
  }
}
```

# Práctica #4

Agregar un componente AlumnosLista

Agregar un componente AlumnoEdicion

Agregar inputs y eventos

# Forms en Angular

- Template Driven: El form está definido en el template (HTML)
- Reactive Forms: estilo de programación "reactive". Creamos desde programación objetos que se vinculan a los controles del template.
- Dynamic Forms: Utilizando los Reactive Forms es posible generar forms dinámicos a partir de información de metada.
- Lo que veremos en este curso son los Template Driven Forms.

# Template Driven Forms

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

A nivel de módulo

Esto incluye la funcionalidad de ngModel y ngForm

# Form, ngForm y variables locales

```
<div>
  <h2>Agregar alumno</h2>
  <form #f="ngForm" (ngSubmit)="salvar(f.value, f.valid)"
    novalidate>
    <!-- we will place our fields here -->
    <button type="submit">Submit</button>
  </form>
</div>
```

#f es un objeto ngForm que contiene los valores y estados del formulario (f.value, f.valid)  
Puede o no estar asociado al modelo

# Asociación form con modelo de datos ngModel, [ngModel] y [(ngModel)]

`<input type="text" name="nombre" ngModel>`

- Vincula la propiedad nombre al valor del input, unidireccional

`<input type="text" name="nombre" [ngModel]="user.nombre">`

- Vincula la propiedad user.nombre al input, unidireccional

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

- Vincula la propiedad user.nombre al input, bidireccional



# Input radio

```
<div class="radio" *ngFor="let s of sexos;  
                                let i = index" >  
  <label>  
    <input type="radio" name="sexo" [value]="i"  
      [(ngModel)]="alumnoEdit.sexo">  
      {{s}}  
  </label>  
</div>
```

# Input select

```
<div class="form-group">
  <label for="nombre">Perfil</label>
  <select class="form-control" name="perfil"
    placeholder="perfil"
    [(ngModel)]="alumnoEdit.perfil">
    <option *ngFor="let p of perfiles; let j = index"
      [value]="j">{{p}}</option>
  </select>
</div>
```

# Checkbox

```
<div class="checkbox">  
  <label>  
    <input type="checkbox" name="activo"  
      [(ngModel)]="alumnoEdit.activo">  
    Activo  
  </label>  
</div>
```

# Submit

```
<div>
  <h2>Agregar alumno</h2>
  <form #f="ngForm" (ngSubmit)="salvar(f.value, f.valid)"
    novalidate>
    <!-- we will place our fields here -->
    <button type="submit">Submit</button>
  </form>
</div>
```

- f.value: Objeto con todos los valores de los controles ngModel'eados
- f.valid: boolean que indica si el model es válido.... Válido? Validaciones?

# Validaciones - submit

```
<form novalidate (ngSubmit)="onSubmit(f)"  
#f="ngForm">  
  ...  
  <button type="submit"  
    [disabled]="f.invalid">Aceptar</button>  
</form>
```

# Validaciones - controles

- Atributos de HTML5
  - required
  - maxlength
  - minlength
  - min y max
  - disabled
  - pattern (regexp)
    - `pattern="^[a-zA-Z0-9_+.]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-]+$"`

# Validaciones - controles

```
<input name="nombre" ngModel required #nombre="ngModel">  
<div *ngIf="nombre.hasError('required')" class="error">  
    El nombre es obligatorio  
</div>
```

```
<input name="nombre" ngModel required #nombre="ngModel">  
<div *ngIf="nombre.invalid" class="error">  
    El nombre es obligatorio  
</div>
```

# Validaciones - controles

- Propiedades de variables ngModel (y clases CSS)
  - valid
  - invalid
  - pristine
  - dirty
  - touched
  - untouched



# It's a Material World - Angular Material Form Field

- `import {MatFormFieldModule} from '@angular/material/form-field';`

`<mat-form-field>`

`<input matInput placeholder="Input">`

`</mat-form-field>`



# Form Field

- Input
  - TextArea
  - Select
- 
- Color
  - floatLabel
  - hideRequiredMarker

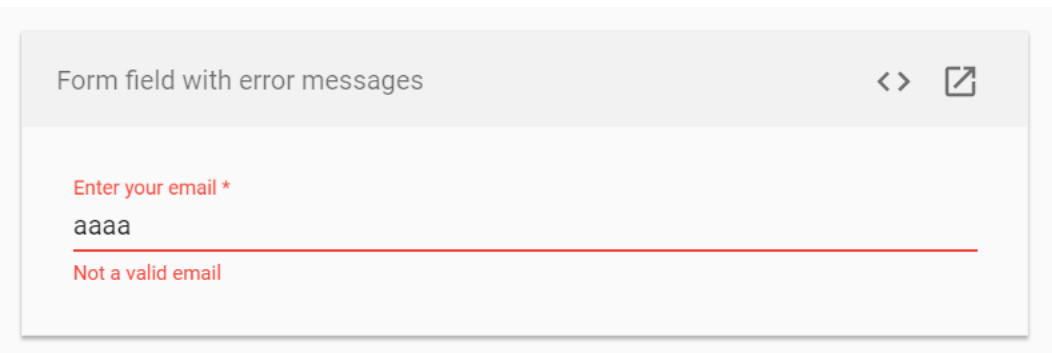
# Form Field - Hint

```
<mat-form-field hintLabel="Max 10 characters">  
  <input matInput #input maxlength="10"  
    placeholder="Enter some input">  
  <mat-hint align="end">{{input.value?.length || 0}}/10</mat-hint>  
</mat-form-field>
```



# Form Field - Error Message

```
<mat-form-field>  
  <input matInput placeholder="Enter your email"  
    [formControl]="email" required>  
  <mat-error *ngIf="email.invalid">  
    {{getErrorMessage()}}  
  </mat-error>  
</mat-form-field>
```



Form field with error messages

Enter your email \*

aaaa

Not a valid email


# Form Field - Prefijos & Sufijos

```
<mat-form-field>
```

```
<input matInput placeholder="Enter your password"  
[type]="hide ? 'password' : 'text'">
```

```
<mat-icon matSuffix (click)="hide = !hide">  
  {{hide ? 'visibility' : 'visibility_off'}}  
</mat-icon>
```

```
</mat-form-field>
```



Form field with prefix & suffix

Enter your password  
sasdas

\$ Amount .00

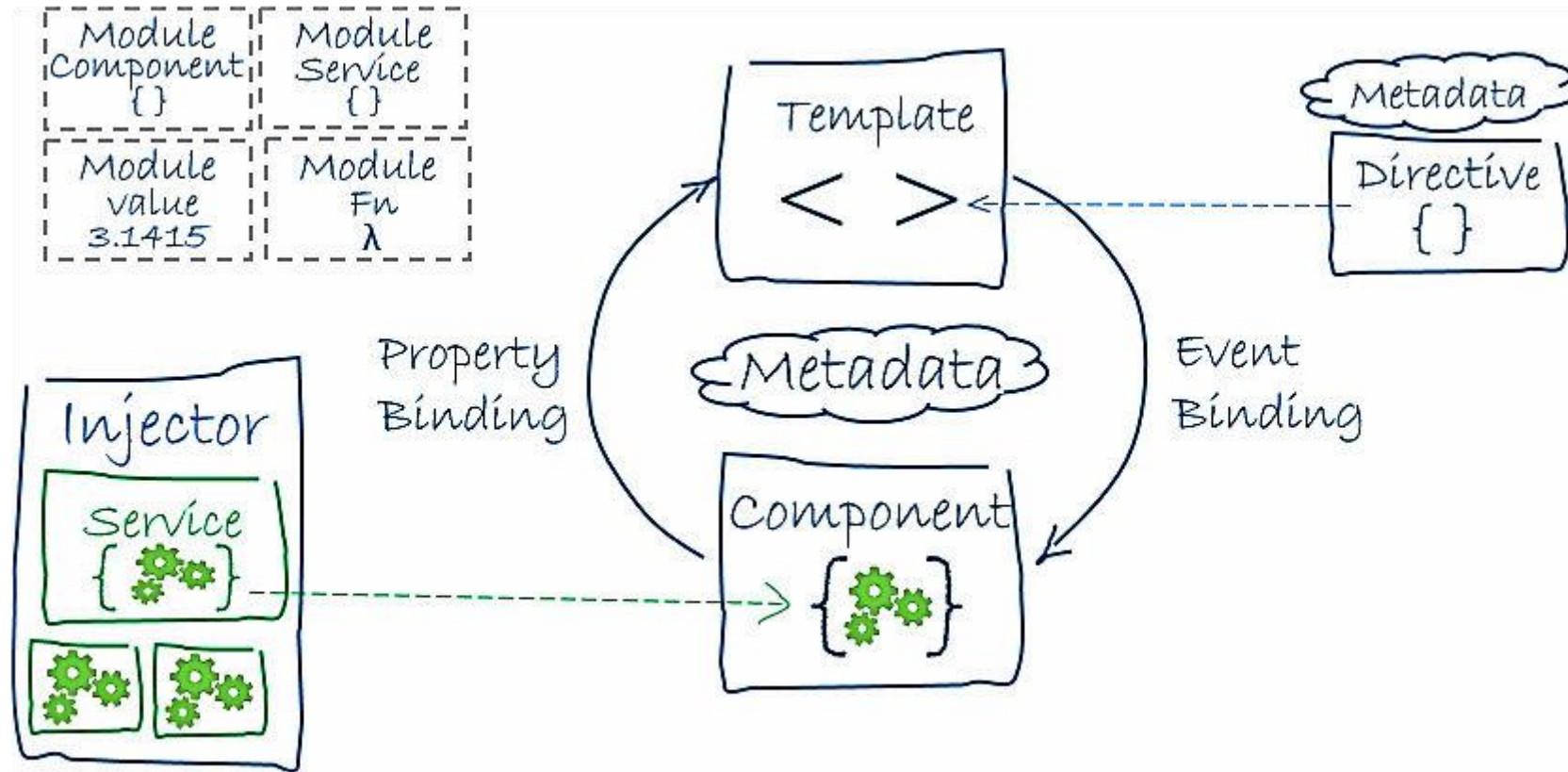
# Material Form Inputs

- Input
- Autocomplete
- Checkbox
- Datepicker
- Radio Button
- Select
- Slider
- Slide Toogle

# Práctica #5

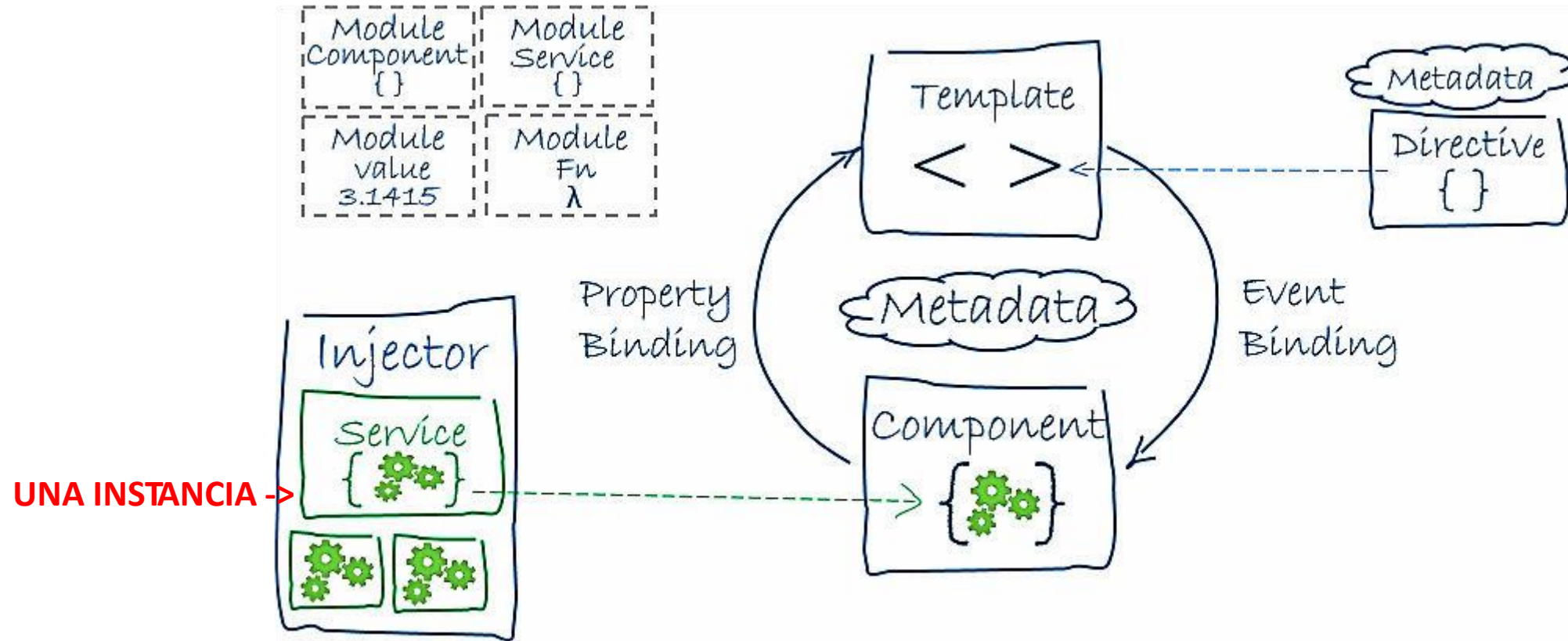
- Uso del ngForm y ngModel
- Validaciones
- Uso de Angular Material

# Servicios .... Dependency Injection

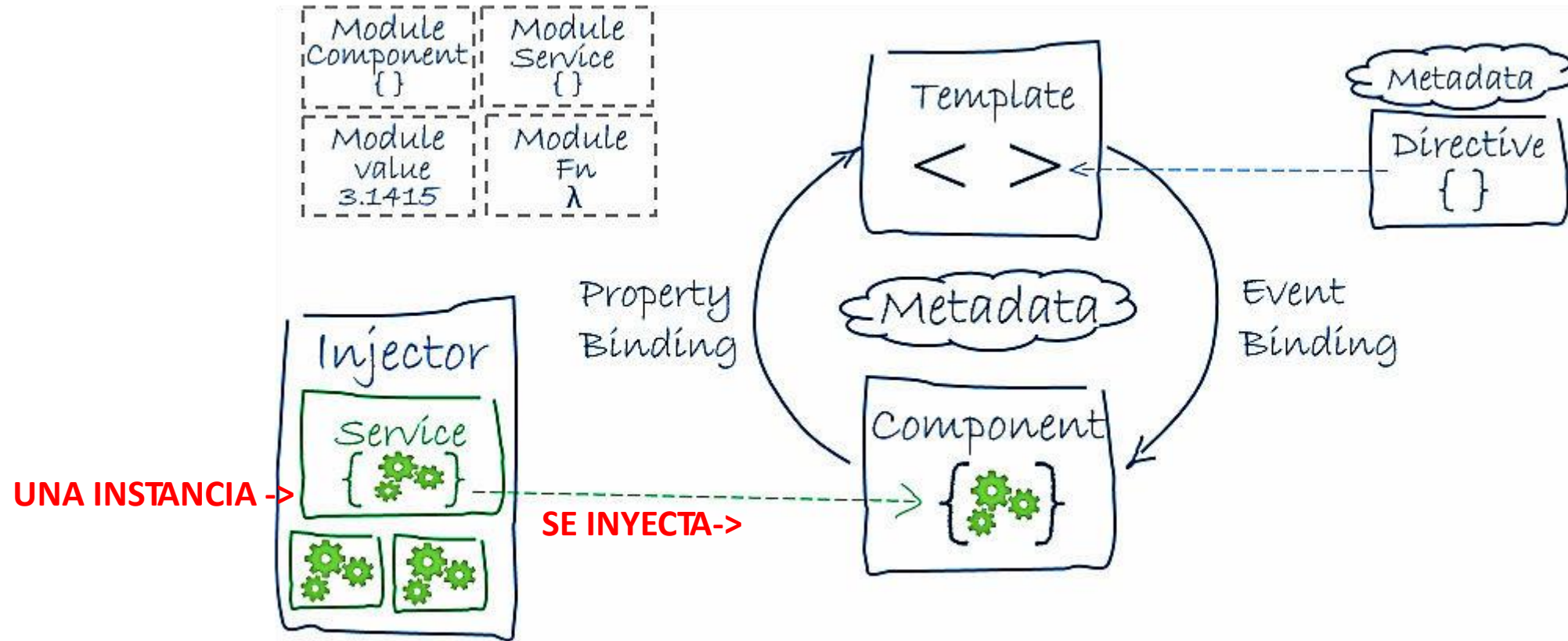




# Servicios .... Dependency Injection



# Servicios .... Dependency Injection



# Servicios en Angular 5

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

@Injectable()
export class AlumnosService {

    propiedades...

    constructor() { }

    funciones() ....

}
```

# Servicios y angular-cli

```
ng generate service alumnos
```

```
ng generate service alumnos --flat
```

```
ng generate service alumnos --module app
```

Actualiza el array de inyectables (providers)

# Servicios en Angular 6

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

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

  propiedades...

  constructor() { }

  funciones() ....
}
```

# Servicios – LO QUE NO HAY QUE HACER

```
import { AlumnosService } from './alumnos.service';
```

```
alumnosService = new AlumnosService(); // ←-----don't do this
```

# Servicios – Como utilizarlos

```
import { AlumnosService } from './alumnos.service';  
.....  
  providers: [AlumnosService]  
.....  
constructor(private alumnosService: AlumnosService)  
{  
}  
  
getAlumnos(): void {  
  this.Alumnos = this.alumnosService.getAlumnos();  
}
```

# OnInit : Inicialización del componente

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

```
export class AppComponent implements OnInit {
```

```
.....
```

```
  ngOnInit(): void {  
    getAlumnos()  
  }
```

```
}
```



# Árbol de Inyección - providers

Módulo

app.component

Alumnos-lista  
.component

Alumno-edicion  
.component

# Práctica #6

Encapsular en un servicio de Alumnos, la maqueta de acceso HTTP implementada por medio de arrays

# Routing en Angular

- Dependiendo de la URL se muestra un componente u otro
  - Localhost:4200/alumnos
  - Localhost:4200/alumno/3
- Base: Define como se van a componer las rutas
  - `<base href="/">`

# Router – Definición de rutas en el módulo

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

```
....
```

```
Imports: [
```

```
  RouterModule.forRoot([
```

```
    { path: '', redirectTo: 'home', pathMatch: 'full' },
```

```
    { path: 'home', component: HomeComponent },
```

```
    { path: 'login', component: LoginComponent },
```

```
    { path: 'dashboard', component: DashboardComponent }
```

```
  ]),
```

```
....
```

# routerLink & router-outlet

```
<h1>{{titulo}}</h1>
```

```
<nav>
```

```
<a routerLink="/dashboard" routerLinkActive="activo">Dashboard</a>
```

```
<a routerLink="/alumnos" routerLinkActive="activo">Alumnos</a>
```

```
</nav>
```

```
<router-outlet> </router-outlet>
```

# Rutas con parámetros

- En la definición de la ruta:

```
imports: [  
  BrowserModule,  
  RouterModule.forRoot(  
    { path: 'alumno/:id' , component: AlumnoEdicionComponent},
```

# Rutas con múltiples parámetros

- En la definición de la ruta:

```
imports: [  
  BrowserModule,  
  RouterModule.forRoot([  
    { path: 'alumno/:operacion/:id' , component: AlumnoEdicionComponent},
```

# Navegación desde código pasando parámetros

```
import { Router } from '@angular/router';
```

```
....
```

```
constructor( private router: Router, private service...) {}
```

```
...
```

```
this.router.navigate(['/alumno', alumno.id]);
```

```
this.router.navigate(['/alumno', 'modificar',alumno.id]);
```



# Resolviendo los parámetros

```
import { Router, ActivatedRoute } from '@angular/router';

constructor(
  private router: Router,
  private activeRoute: ActivatedRoute,
  private alumnosService: AlumnosService) { }

ngOnInit() {
  const id = this.activeRoute.snapshot.paramMap.get('id');
  this.alumno = this.alumnosService.get(id);
}
```

# Práctica #7

- Implementación de menú y rutas
- Cambios en la forma de comunicar los componentes
- Utilización de mat-table

# Observables, una introducción

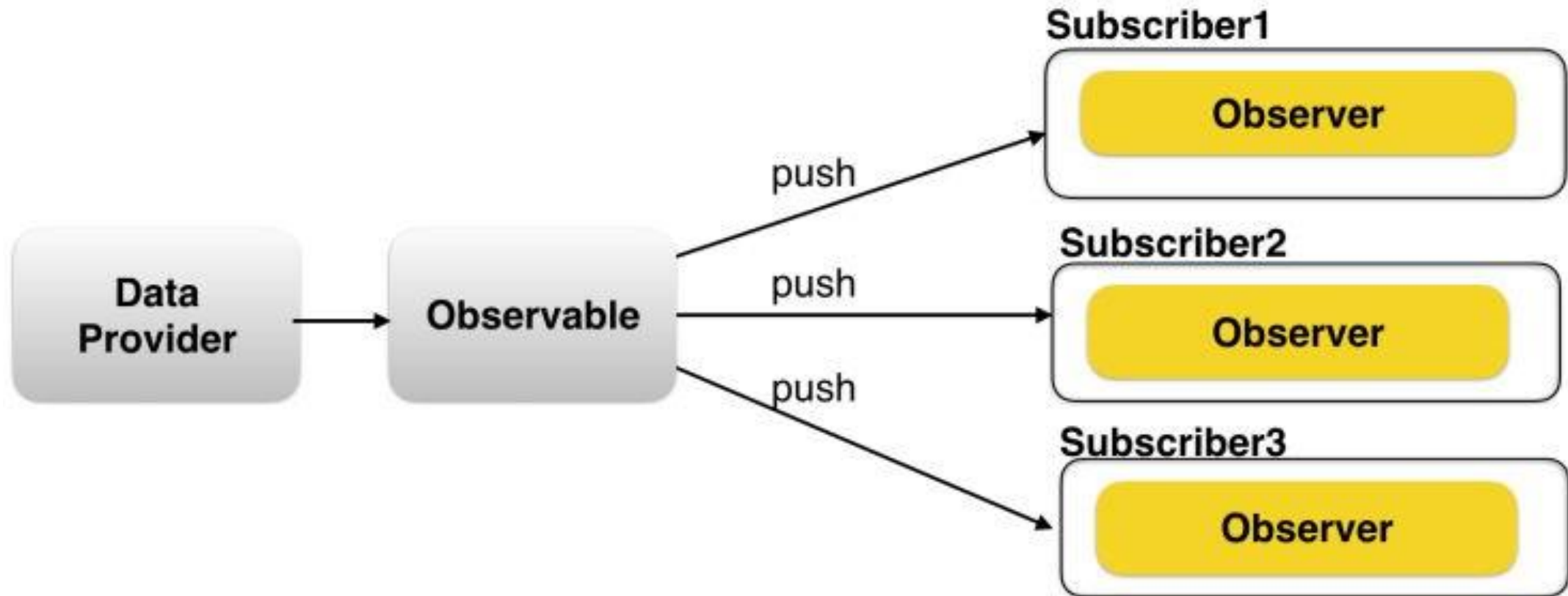
- Programación reactiva
- Parecidos a Promise()
- Pero con ventajas
  - Se pueden cancelar
  - Se usan Data Stream, pueden transformarse

# Reactive Programming

Reactive vs Imperative Programming

```
let a1 = 5;  
let b1 = 1;  
let c1 = a1 + b1;  
a1 = 50;  
b1 = 60;
```

# Patrón de Diseño: Observable



# Patrones de Diseño: Pull vs Push

<b>Pull</b>	<b>Push</b>
Arrays, Generators, Iterables	Promises Observables
synchronous	asynchronous

# Patrón de Diseño: Observable

- Observable -> of()
- Observador -> 3 funciones
  - value => console.log(value),
  - err => console.error(err),
  - () => console.log('Streaming is over')
- El Observador se subscribe al Observable (se lee alreves)

```
of(1, 2, 3)
  .subscribe(
    value => console.log(value),
    err => console.error(err),
    () => console.log('Streaming is over')
  );
```

# RxJS

- Desarrollada por Microsoft
- Implementada para muchos lenguajes
- Implementa Observables
- Muy utilizada dentro de Angular





# Asynchronous Data Stream

- Asynchronous: Va a suceder en algún momento del futuro
- Data: Información cruda
- Streams: Los valores se generan a lo largo del tiempo

# Reactive Programming

- Es programar con asynchronous data streams
- Es tomar un stream y combinarlo, transformarlo, crearlo, filtrarlo, etc hasta obtener el resultado esperado

# Ejemplos

Stream

1

2

3

Array

[

1

,

2

,

3

]

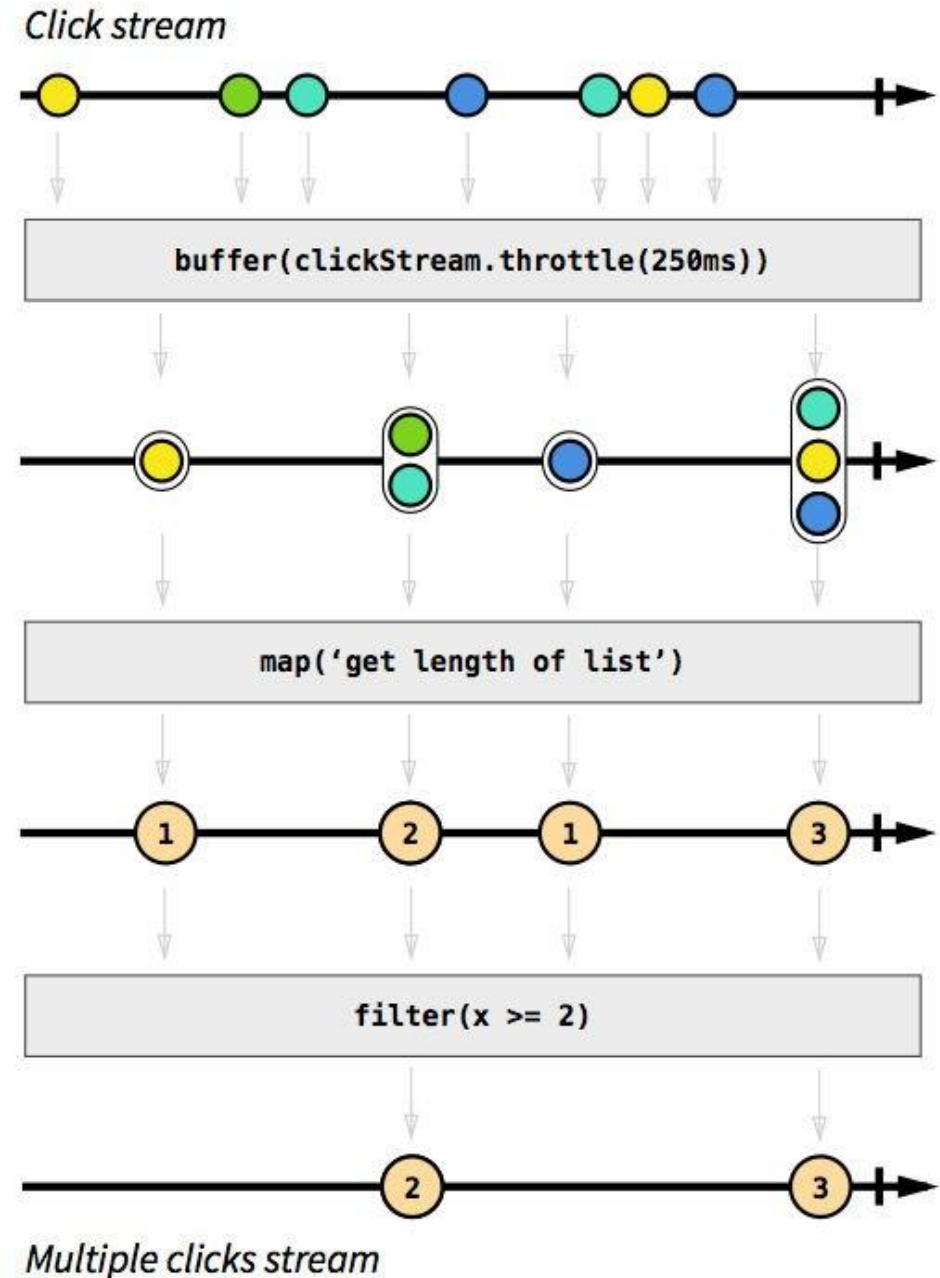
Clicks

1

2

3

# Ejemplo: Implementación de MultiClick



# Pero vamos de a poco....Arrays

Extras en ES5

.forEach()

.map()

.filter()

.reduce()

Composición

# .forEach

```
var team = [  
  { name: "Igor Minar", commits: 259 },  
  { name: "Jeff Cross", commits: 105 },  
  { name: "Brian Ford", commits: 143 }  
];  
  
for(var i=0, ii=team.length; i<ii; i+=1){  
  console.log(team[i].name);  
}  
  
team.forEach( member => console.log(member.name) );
```

# .map()

```
var team = [  
    { name: "Igor Minar", commits: 259 },  
    { name: "Jeff Cross", commits: 105 },  
    { name: "Brian Ford", commits: 143 }  
];  
  
var newTeam = [];  
for(var i=0, ii=team.length; i<ii; i+=1){  
    newTeam.push({ name: team[i].name });  
}  
  
var onlyNames = team.map(  
    member => { name: member.name }  
);
```

.filter()

```
var team = [  
  { name: "Igor Minar", commits: 259 },  
  { name: "Jeff Cross", commits: 105 },  
  { name: "Brian Ford", commits: 143 }  
];  
  
var onlyOver120Commits = [];  
for(var i=0, ii=team.length; i<ii; i+=1){  
  if (team[i].commits>120) {  
    onlyOver120Commits.push(team[i]);  
  }  
}  
  
var onlyOver120Commits = team.filter(  
  member => member.commits>120  
);
```



.reduce()

```
var team = [  
  { name: "Igor Minar", commits: 259 },  
  { name: "Jeff Cross", commits: 105 },  
  { name: "Brian Ford", commits: 143 }  
];  
var total = 0; // initial value  
for(var i=0, ii=team.length; i<ii; i+=1){  
  total = total + team[i].commits;  
}  
var total = team.reduce(  
  (total, member) => total + member.commits  
, 0); // initial value  
  
// 507
```

# Composición

```
var over120Commits = x => x.commits>120;  
var memberName = x => x.name;  
var toUpperCase = x => x.toUpperCase();  
var log = x => console.log(x);
```

```
team  
  .filter(over120Commits)  
  .map(memberName)  
  .map(toUpperCase)  
  .forEach(log);
```

```
"IGOR MINAR"
```

```
"BRIAN FORD"
```

# RxJS - Terminología básica

- Observable:
  - stream de datos que pushea valores a lo largo del tiempo
- Observer:
  - Consumidor de un stream de datos observable
- Subscriber:
  - Conecta un Observable con un Observer
- Operator :
  - Función que transforma los valores de los stream de datos

# RxJS en Angular

Asynchronous processing

Http

Forms: controls, validation

Component events

EventEmitter

# Observable

```
//Observable    constructor

let obs$ = new Observable(observer => {
  try{
    //pushing values
    observer.next(1);
    observer.next(2);
    observer.next(3);

    //complete stream
    observer.complete();
  }
  catch(e) {
    //error handling
    observer.error(e);
  }
});
```

# Funciones auxiliares de Observables

```
//Observable creation helpers

Observable.of(1); // 1|
Observable.of(1,2,3).delay(100); // ---1---2---3|

Observable.from(promise);
Observable.fromArray([1,2,3]); // ---1---2---3|

Observable.fromEvent(inputDOMElement, 'keyup');
```

# Subscribe

```
Observable.subscribe(  
  /* next */ x => console.log(x),  
  /* error */ x => console.log('#'),  
  /* complete */ () => console.log('|')  
);
```

```
Observable.subscribe({  
  next: x=>console.log(x),  
  error: x =>console.log('#'),  
  complete: () => console.log('|')  
});
```

# Unsubscribe

```
var subscriber = Observable.subscribe(  
  twit => feed.push(twit),  
  error=> console.log(error),  
  () => console.log('done')  
);  
  
subscriber.unsubscribe();
```



# Operators

```
// simple operators
```

```
map(), filter(), reduce(), scan(), first(), last(), single(),  
elementAt(), toArray(), isEmpty(), take(), skip(), startWith()
```

```
// merging and joining
```

```
merge(), mergeMap(flatMap), concat(), concatMap(), switch(),  
switchMap(), zip()
```

```
// splitting and grouping
```

```
groupBy(), window(), partition()
```

```
// buffering
```

```
buffer(), throttle(), debounce(), sample()
```

¿Porqué observables?

Flexible: sync or async

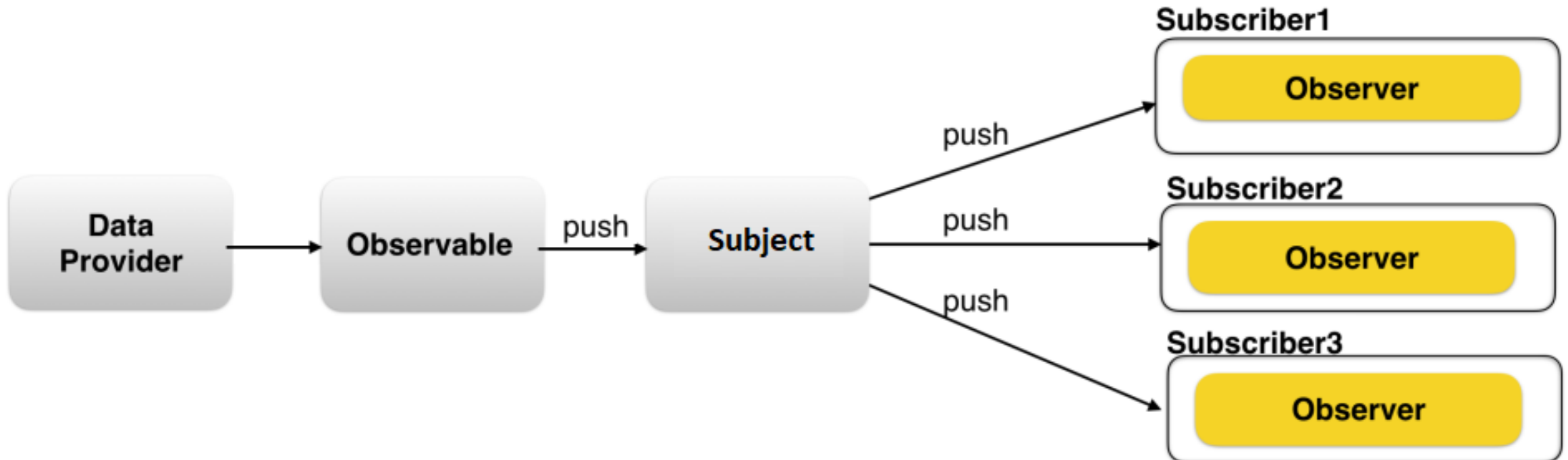
Powerful operators

Less code

# Subject

- Un Observable diferente
- Es un observable y un observador al mismo tiempo (proxy)
- Emite el mismo valor a todos los suscriptores (multicast)

# Observable Data Service



# Práctica #8

- Introducción a RxJS
- Implementación de un Observable Data Service

# HttpClient

```
import {HttpClientModule} from '@angular/common/http';
```

Funcionalidad armada sobre XMLHttpRequest

# GET

```
results: string[];

// Inject HttpClient into your component or service.
constructor(private http: HttpClient) {}

ngOnInit(): void {
  // Make the HTTP request:
  this.http.get('/api/items').subscribe(data => {
    // Read the result field from the JSON response.
    this.results = data['results'];
  });
}
```

GET  
de un tipo  
de datos

```
interface ItemsResponse {  
    results: string[];  
}  
  
this.results: ItemsResponse;  
  
http.get<ItemsResponse>('/api/items').subscribe  
(data => {  
    // data is now an instance of type  
    ItemsResponse,  
    // so you can do this:  
    this.results = data.results;  
});
```



# GET de toda la respuesta (no sólo el body)

```
let MyJsonData: any;
http
  .get<MyJsonData>('/data.json', {observe: 'response'})
  .subscribe(resp => {
    // Here, resp is of type HttpResponse<MyJsonData>.
    // You can inspect its headers:
    console.log(resp.headers.get('X-Custom-Header'));
    // And access the body directly, which is typed as MyJsonData
    // as requested.
    console.log(resp.body.someField);
  });
}
```

# HTTP Errors

```
http
.get<ItemsResponse>('/api/items')
  .subscribe(
    // Successful responses call the first callback.
    data => {...},
    // Errors will call this callback instead:
    err => {
      console.log('Something went wrong!');
    }
  );
```

# HTTP Errors

```
http
  .get<ItemsResponse>('/api/items')
    .subscribe(
      data => {...},
      (err: HttpResponse) => {
        if (err.error instanceof Error) {
          // A client-side or network error occurred. Handle it accordingly.
          console.log('An error occurred:', err.error.message);
        } else {
          // The backend returned an unsuccessful response code.
          // The response body may contain clues as to what went wrong,
          console.log(`Backend returned code ${err.status}, body was:
            ${err.error}`);
        }
      }
    );
```

# HTTP retry

```
import 'rxjs/add/operator/retry';

http
  .get<ItemsResponse>('/api/items')
  // Retry this request up to 3 times.
  .retry(3)
  // Any errors after the 3rd retry will fall through to the app.
  .subscribe(...);
```

# GET datos no-json

```
http
.get('/textfile.txt', {responseType: 'text'})
// The Observable returned by get() is of type
// Observable<string> because a text response was specified.
// There's no need to pass a <string> type parameter to get().
.subscribe(data => console.log(data));
```

# POST

```
const body = {name: 'Brad'};
```

```
http
```

```
  .post('/api/developers/add', body)
```

```
  // See below - subscribe() is still necessary when using post().
```

```
  .subscribe();
```

# HEADERS

```
http
.post('/api/items/add', body, {
  headers:
    new HttpHeaders().set('Authorization', 'my-auth-token'),
})
.subscribe();
```

# Parámetros en la URL

```
http
.post('/api/items/add', body, {
    params: new HttpParams().set('id', '3'),
})
.subscribe();

//  /api/items/add?id=3
```



# PUT

```
this.http  
  .put(this.url, this.payload, {  
    params: new HttpParams().set('id', '56784')  
  })  
  .subscribe(...);
```

# DELETE

```
this.http  
  .delete(this.url, {  
    params: new HttpParams().set('id', '56784')  
  })  
  .subscribe(...);
```

# Práctica #9

- Convertir nuestro servicio de alumnos en un consumidor de Web API utilizando HttpClient
- Adaptar los componentes a resultados Observables

# Temas a estudiar

- Creación de pipes
- Scss/Sass
- Arquitectura y modularización
- Profundización router (sub rutas, guards y políticas de reuso)
- Intercepción de llamadas HTTP
- Componentes dinámicos y complejos
- Autenticación (IdentityServer, OAuth, JSON Web Token)