



# Angular

---

Santos Jiménez, Cristóbal Belda

# Angular Tools

---

# Contents

- Node.js, npm, Yarn
- TypeScript
- Browser
- Angular CLI
- Dev Environment

# Node.js, npm, Yarn

Needed **updated** versions of Node.js and npm ( at least 6.9 )

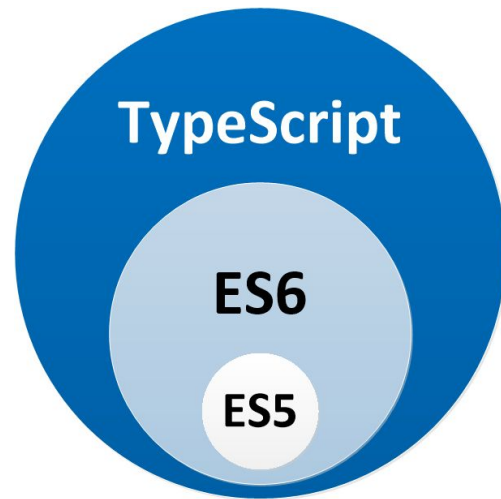
<https://nodejs.org/en/>



# TypeScript

Angular uses TypeScript as its main programming language ( at least 2.1 )

```
$ npm install -g typescript
```




# Browser

We highly recommend the use of *Chrome* browser during the development of your Angular application. Also, you'll be forced to use it while testing, as long as Angular needs Chrome to trigger its tests. See Angular's [browser support](#).



# Browser

Augury Chrome Dev Tools extension for debugging <https://augury.angular.io/>



Component Tree

Router Tree

NgModules

▼ AppComponent

▼ MdSidenavContainer

▼ div

router-outlet (name="")

▼ HomeComponent

▼ DemoComponent

▼ TdLayoutComponent

▼ MdSidenavContainer

► MdSidenav

▼ div

▼ TdLayoutNavListComponent

▼ MdSidenavContainer

► MdSidenav

▼ div

▼ MdToolBar

► md-toolbar-row

► TdSearchBoxComponent

► MdButton

ToolTipComponent

ToolTipComponent

ToolTipComponent

ToolTipComponent

► MdButton

► MdButton

► MdButton

md-divider

► TdDataTableComponent

► TdPagingBarComponent

▼ MdSidenav

FocusTrap

Properties

Injector Graph

Component Hierarchy

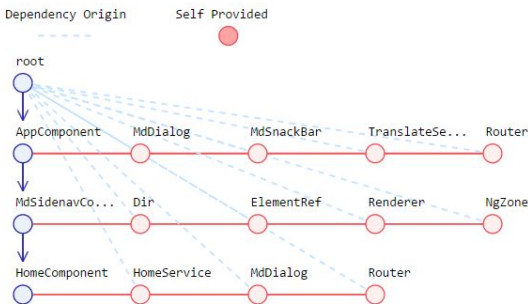
AppComponent » MdSidenavContainer » HomeComponent

Injector Graph

Dependency Origin

Self

Provided



```

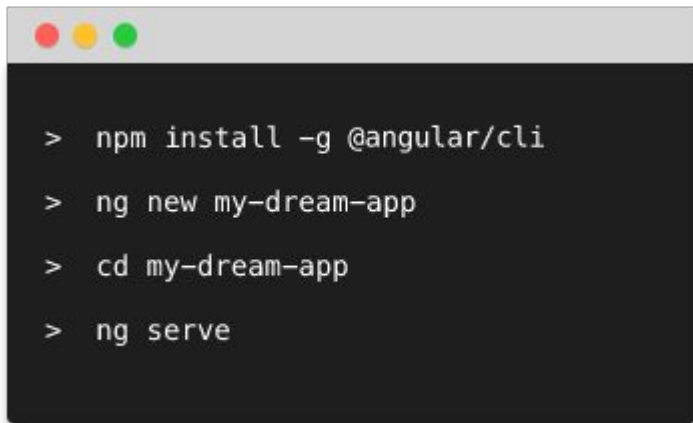
graph TD
    root((root)) --> AppComponent((AppComponent))
    root --> MdSidenavCo...((MdSidenavContainer))
    root --> HomeComponent((HomeComponent))
    AppComponent --> MdDialog1((MdDialog))
    AppComponent --> MdSnackBar1((MdSnackBar))
    AppComponent --> TranslateSe...((TranslateService))
    AppComponent --> Router1((Router))
    MdSidenavCo... --> Dir((Dir))
    MdSidenavCo... --> ElementRef((ElementRef))
    MdSidenavCo... --> Renderer((Renderer))
    MdSidenavCo... --> NgZone((NgZone))
    HomeComponent --> HomeService((HomeService))
    HomeComponent --> MdDialog2((MdDialog))
    HomeComponent --> Router2((Router))
    
```

```
Properties    Injector Graph
HomeComponent (View Source)
Change Detection: Default
▼ State
create: true
sidenavOpen: false
sidenavAddItem: false
enabledDelete: false
@ViewChild(addItem) addItem: undefined
@ViewChild(search) search: undefined
@ViewChild(mydatatable) table: undefined
► service: HomeService
► _dialog: MdDialog
► router: Router
newItem: {}
checkedItems: Array[0]
► @ViewChild(sidenav) sidenav: MdSidenav
► items: Array[12]
```



# Angular CLI

The Angular CLI makes it easy to create an application that already works, right out of the box <https://github.com/angular/angular-cli>

A terminal window with a dark background and a light gray title bar containing three colored window control buttons (red, yellow, green). The terminal displays a series of commands and their outputs, each preceded by a prompt character '>'.

```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

\$ ng new

\$ ng generate

\$ ng serve



# Dev Environment

IDEs that can support our Angular development.



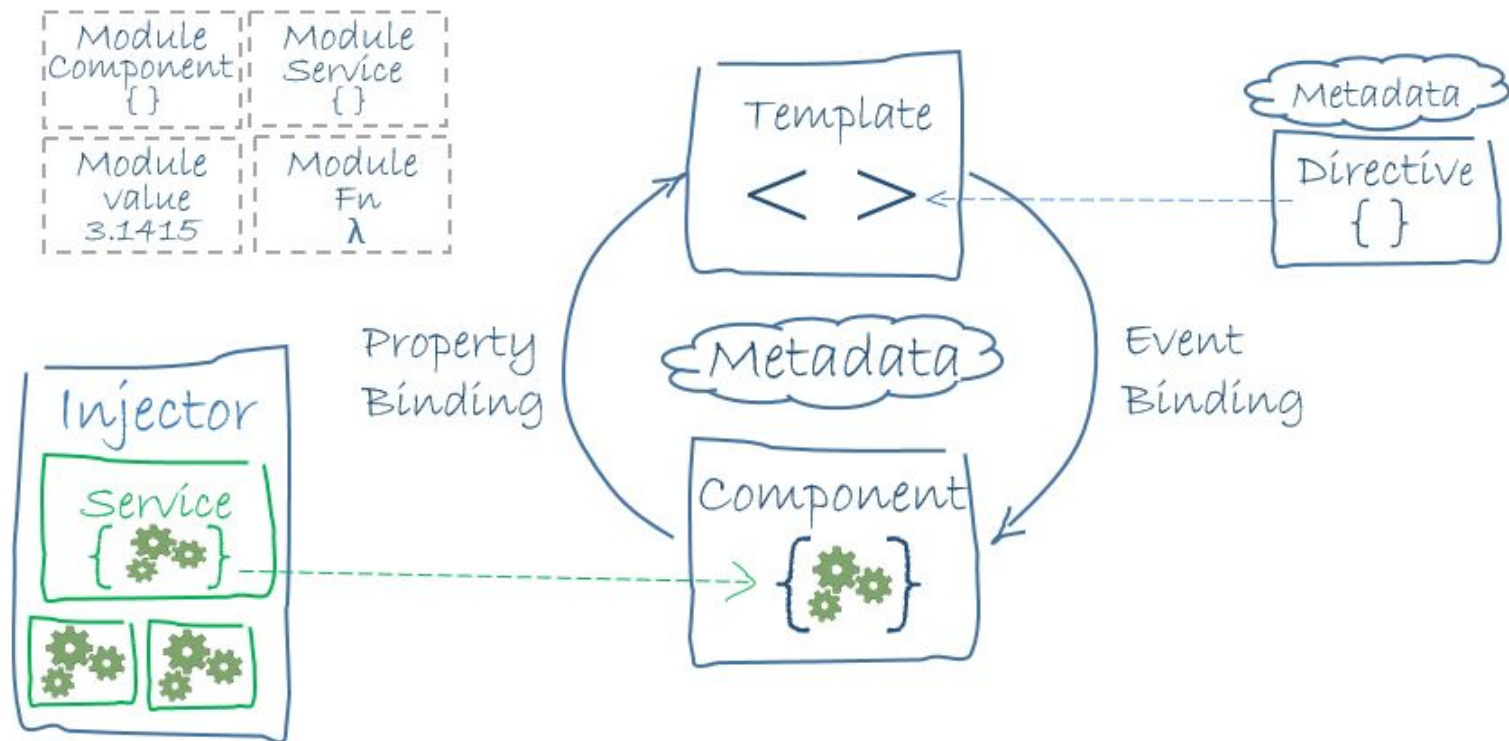
# Angular Concepts

---

# Contents

- Architecture Overview
- Modules
- Components
- Templates
- Metadata
- Data binding
- Directives
- Services
- Dependency injection

# Architecture Overview



# Modules

Angular apps are modular and Angular has its own modularity system called *Angular modules* or *NgModules*. **They help organize an application into cohesive blocks of functionality.**

An *NgModule* is a class adorned with the **@NgModule** decorator. It takes a metadata object that tells Angular how to compile and run module code. It identifies the module's own components, directives and pipes making some of them public so external components can use them.

<https://angular.io/docs/ts/latest/guide/ngmodule.html>

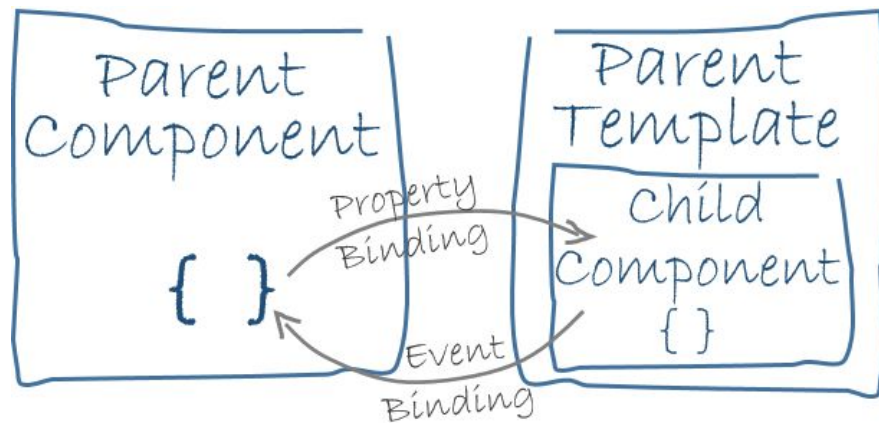
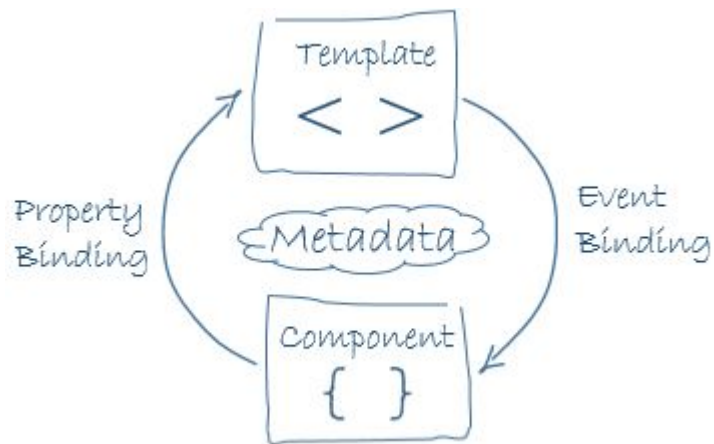
# Modules

Properties of an *NgModule*:

- **Declarations:** View classes (*components, directives and pipes*).
- **Exports:** Declarations that should be usable in other modules' component templates.
- **Imports:** Other modules whose exported classes are needed.
- **Providers:** Creators of *services*. They become accessible in the app.
- **Bootstrap:** Main application view (root component), that hosts all app views.

# Components

Angular applications are made up of *components*. A *component* is the combination of an HTML template and a component class that controls a portion of the screen.



# Components

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

```
@Component({  
  selector: 'example',  
  template: `  
    <h1>Example Component</h1>  
  `,  
})
```

```
export class ExampleComponent {  
  constructor() { }  
}
```



# Templates

You define a component's view with its companion **template**. A template is a kind of HTML that tells Angular how to render the component. A template looks like regular HTML, except for a few differences.

# Templates

```
<h2>Hero List</h2>
```

```
<p><i>Pick a hero from the list</i></p>
```

```
<ul>
```

```
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
```

```
    {{hero.name}}
```

```
  </li>
```

```
</ul>
```

```
<hero-detail *ngIf="selectedHero" [hero]="selectedHero"></hero-detail>
```

# Metadata

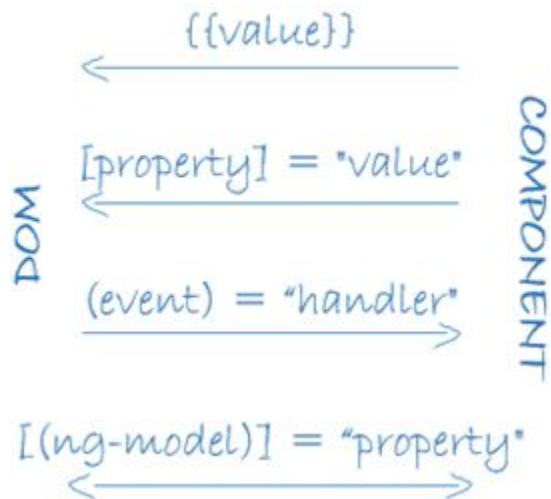
Metadata tells **Angular** how to process a class. Without this, our components would be just *normal* Typescript classes.

```
@Component({  
  selector:    'hero-list',  
  templateUrl: './hero-list.component.html',  
  providers:   [ HeroService ]  
})
```



# Data binding

Angular supports data binding. A mechanism for coordinating parts of a template with parts of a component.



# Data binding

Angular supports data binding. A mechanism for coordinating parts of a template with parts of a component.

```
<li>{{hero.name}}</li> // component -> DOM
<hero-detail [hero]="selectedHero"></hero-detail> // component -> DOM
<li (click)="selectHero(hero)"></li> // DOM -> component
<input [(ngModel)]="username"> // Two-way data binding
```

# Directives

Angular templates are *dynamic*. When Angular renders them, it transforms the DOM according to the instructions given by **directives**.

A directive is a class with a **@Directive** decorator. A component is a *directive-with-a-template*; a **@Component** decorator is actually a **@Directive** decorator extended with template-oriented features.



# Structural Directives

**Structural** directives alter layout by adding, removing, and replacing elements in DOM.

```
<li *ngFor="let hero of heroes">...</li>           // for loop  
<hero-detail *ngIf="selectedHero"></hero-detail>    // if
```



# Attribute Directives

**Attribute** directives alter the appearance or behavior of an existing element. In templates they look like regular HTML attributes, hence the name.

The **ngModel** directive, is an example of an attribute directive. **ngModel** modifies the behavior of an existing element (typically an `<input>`) by setting its display value property and responding to change events.

```
<input [(ngModel)]="hero.name">
```





# Services

A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

**Almost anything can be a service.**

There is nothing specifically *Angular* about services. Angular has no definition of a service. There is no service base class, and no place to register a service. Yet services are fundamental to any Angular application. Components are big consumers of services.



# Services

A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

- Logging service
- Data service
- Message bus
- Application configuration



# Services

A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

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

@Injectable()
export class ExampleService {
  constructor() { }
}
```



# Dependency Injection

*Dependency injection* is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services. Angular uses dependency injection to provide new components with the services they need.

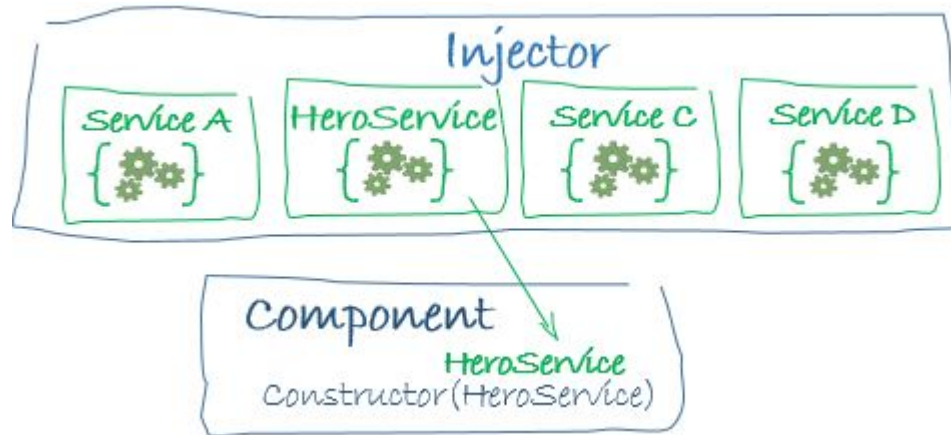
Angular can tell which services a component needs by looking at the types of its constructor parameters.



A hand-drawn diagram illustrating a component's constructor. It consists of a rectangular box with a blue border. Inside the box, the word "Component" is written in blue. Below it, the text "{Constructor(service)}" is written in blue. To the right of the box, the word "Service" is written in green, with three green lines extending from its top right corner, suggesting an arrow pointing towards the "service" parameter in the constructor.

```
Component  
{Constructor(service)}
```

# Dependency Injection



# Cross-platform usages

---

# Ionic

- Free & Open Source.
- Fully Cross-Platform.
- Premier Native Plugins.
- First-class Documentation.

```
$ npm install -g ionic cordova
```



# NativeScript

**NativeScript** is a free and open source framework for building native iOS and Android apps using JavaScript and CSS. NativeScript renders UIs with the native platform's rendering engine—no WebViews—resulting in native-like performance and UX.

```
$ npm install -g nativescript
```





# Links of Interest

---

ng-book: <https://www.ng-book.com/2/>

Documentation overview: <https://angular.io/docs/ts/latest/guide/>

NgModules: <https://angular.io/docs/ts/latest/guide/ngmodule.html>

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

Yarn: <https://yarnpkg.com/en/> and <https://www.sitepoint.com/yarn-vs-npm/>

Typescript: <https://www.typescriptlang.org/docs/tutorial.html>

Angular CLI: <https://github.com/angular/angular-cli>

Angular client + Express server: <https://medium.com/angular-client-express-server>

Ionic 2: <http://ionicframework.com/docs/intro/installation/>

Cordova: <https://cordova.apache.org/#getstarted>

NativeScript: <http://docs.nativescript.org/angular/tutorial/ng-chapter-0>

Augury: <https://augury.angular.io/>

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

MEAN app: <https://scotch.io/tutorials/mean-app-with-angular-2-and-the-angular-cli>

Santos Jiménez's TS workshop: <https://docs.google.com/presentation/typescript>

# Caso práctico

---

## Paso 0: Descargar starter (opcional)

Hay un “starter” de Angular Material disponible públicamente en GitHub:

<http://github.com/cbelda/ng-material-starter.git>

Los pasos que he seguido para configurarlo son los siguientes:

# Paso 1: Preparación del entorno

Clonarse el repositorio <https://github.com/cbelda/angular-environment-setup.git>

Ejecutar el script del directorio raíz: **setup.sh**

1. Crea proyecto de Angular utilizando el Angular CLI
2. Añade librerías para utilizar Angular Material en el proyecto
3. Sirve el proyecto y lo abre en Chrome y en Visual Studio Code.
4. Seguir los pasos del [getting started](#) de Material a partir del paso “Animations”

# Paso 1: Preparación del entorno

Añadir los estilos necesarios para que la aplicación tenga un *look* totalmente de material:

1. Crear un tema en el archivo .scss principal de la aplicación (creado por defecto con el comando del Angular CLI)
2. Customizar los 3 colores base de Material (**primary**, **accent** y **warn**).
3. Comparar con la paleta de colores de material:

<https://material.io/guidelines/style/color.html#color-color-palette>

# Paso 1: Preparación del entorno

Instalamos el paquete de “Animations” de Angular Material

```
$ yarn add @angular/animations
```

Añadimos el módulo en el *NgModule*

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

@NgModule({
  ...
  imports: [BrowserAnimationsModule],
  ...
})
export class AppModule { }
```



# Paso 1: Preparación del entorno

```
@import '~@angular/material/theming';  
@include mat-core();  
  
$primary: mat-palette($mat-brown, 900);  
$accent:  mat-palette($mat-green, 700, A100, A400);  
$warn:    mat-palette($mat-red, 600);  
  
$theme: mat-light-theme($primary, $accent, $warn);  
  
@include angular-material-theme($theme);
```

# Paso 1: Preparación del entorno

Añadir módulos básicos al *NgModule*

```
import {MatButtonModule, MatCheckboxModule} from '@angular/platform-browser/animations';

@NgModule({
  ...
  imports: [MatButtonModule, MatCheckboxModule],
  ...
})
export class AppModule { }
```

## Paso 2: Añadir componente

El Angular CLI tiene una API con la que podemos generar componentes, servicios, clases etc. para nuestro proyecto:

```
$ ng generate component card
```

## Paso 2: Añadir componente

Podemos probar añadiendo el código relacionado con el componente the “*Card*”, un elemento visual bastante característico de este framework. Queremos conseguir esto:



## Paso 2: Añadir componente

Vamos a la guía y visitamos un ejemplo:

<https://material.angular.io/components/card/examples>

Añadimos el código fuente donde pertoque.

# Paso 2: Añadir componente

En card.component.html

```
<mat-card class="example-card">
  <mat-card-header>
    
    <mat-card-title>Shiba Inu</mat-card-title>
    <mat-card-subtitle>Dog Breed</mat-card-subtitle>
  </mat-card-header>
  
  <mat-card-content>
    <p>
      blablabla...
    </p>
  </mat-card-content>
  <mat-card-actions>
    <button mat-button>LIKE</button>
    <button mat-button>SHARE</button>
  </mat-card-actions>
</mat-card>
```

## Paso 3: Añadir componente “complejo”

El Angular CLI tiene una API con la que podemos generar componentes, servicios, clases etc. para nuestro proyecto:

```
$ ng generate component datepicker
```

Introducir el [código de datepicker](#) en la template del recién creado componente.

## Paso 3: Añadir componente “complejo”

Angular Material dispone de una documentación **de 10** que se puede consultar aquí: <https://material.angular.io/>

La forma de proceder podría ser:

1. Encontrar el componente deseado
2. Añadir el módulo correspondiente al NgModule
3. Comprobar si además son necesarios más módulos. Esto suele mostrarse en tiempo de ejecución en la consola del navegador.



## Paso 3: Añadir componente “complejo”

En nuestro caso añadimos el componente “*datepicker*” de Material. Por tanto, habrá que añadir su módulo correspondiente. Al introducir el código en la template y esperar a que el proceso de Nodejs actualice la aplicación, la consola del navegador nos dará un error haciendo referencia a otro módulo: “*MartFormFieldModule*”.

Deberemos introducirlo en el NgModule también.

## Paso 3: Añadir componente “complejo”

La consola del navegador nos dará otro aviso:

```
✖ ▶ ERROR Error: MatDatepicker: No provider found for DateAdapter. You must import one of the DatepickerComponent.html:6  
following modules at your application root: MatNativeDateModule, MatMomentDateModule, or provide a custom implementation.  
  at createMissingDateImplError (datepicker.es5.js:40)  
  at new MatDatepickerInput (datepicker.es5.js:2237)  
  at createClass (core.js:12449)  
  at createDirectiveInstance (core.js:12284)  
  at createViewNodes (core.js:13742)  
  at callViewAction (core.js:14176)  
  at execComponentViewsAction (core.js:14085)  
  at createViewNodes (core.js:13770)  
  at callViewAction (core.js:14176)  
  at execComponentViewsAction (core.js:14085)
```

Debemos importar los módulos que nos “aconseja” también.

## Paso 3: Añadir componente “complejo”

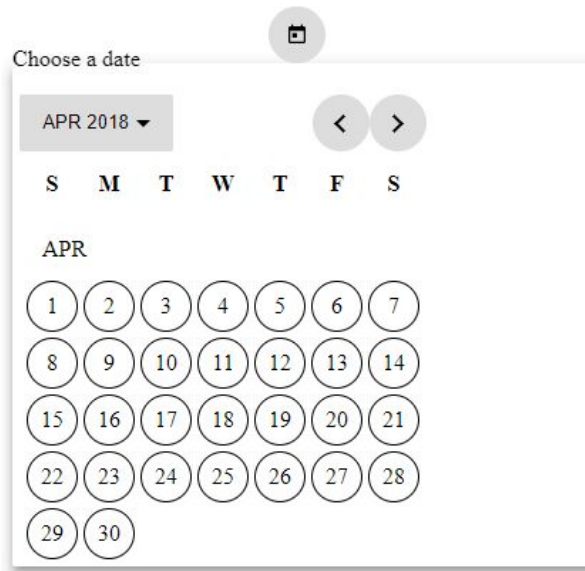
El siguiente error:

```
✖ ERROR Error: mat-form-field must contain a MatFormFieldControl.  
  at getMatFormFieldMissingControlError (form-field.es5.js:111)  
  at MatFormField._validateControlChild (form-field.es5.js:637)  
  at MatFormField.ngAfterContentInit (form-field.es5.js:392)  
  at callProviderLifecycles (core.js:12699)  
  at callElementProvidersLifecycles (core.js:12673)  
  at callLifecycleHooksChildrenFirst (core.js:12656)  
  at checkAndUpdateView (core.js:13806)  
  at callViewAction (core.js:14153)  
  at execComponentViewsAction (core.js:14085)  
  at checkAndUpdateView (core.js:13808)
```

Por probar, buscamos soluciones en algún issue de GitHub de Angular (práctica muy recomendada). Obtenemos [esta información](#). Probamos la solución en el proyecto.

## Paso 3: Añadir componente “complejo”

Al fin tenemos algo!



## Paso 3: Añadir componente “complejo”

Pero tenemos un último aviso de la consola:

```
✖ ▶ ERROR Error: Found the synthetic property @transitionMessages. Please include either
  "BrowserAnimationsModule" or "NoopAnimationsModule" in your application.
    at checkNoSyntheticProp (platform-browser.js:3005)
    at DefaultDomRenderer2.setProperty (platform-browser.js:2962)
    at DebugRenderer2.setProperty (core.js:15412)
    at setElementProperty (core.js:10754)
    at checkAndUpdateElementValue (core.js:10673)
    at checkAndUpdateElementInline (core.js:10607)
    at checkAndUpdateNodeInline (core.js:13889)
    at checkAndUpdateNode (core.js:13836)
    at debugCheckAndUpdateNode (core.js:14729)
    at debugCheckRenderNodeFn (core.js:14708)
```

Debemos incluir los módulos de Animaciones de Angular. Explicado en la sección de “Getting Started” de la [guía de Material](#).



# Thank you!

---

Santos Jiménez, Cristóbal Belda