

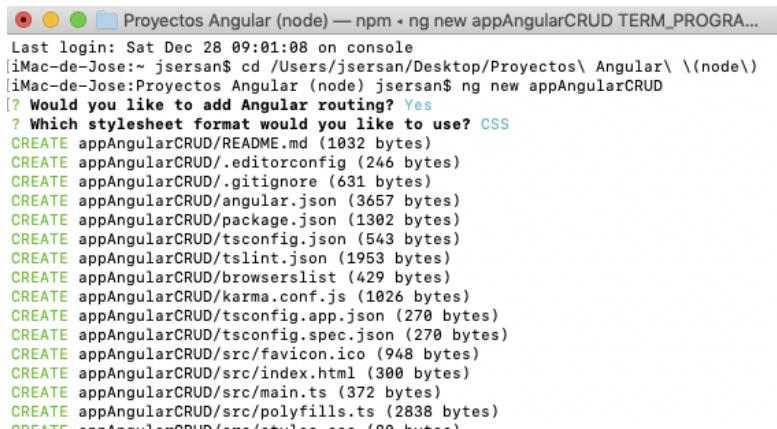
CRUD firebase.

Esto es lo que veremos a continuación:

1. Uso del modulo HTTP.
2. Utilizaremos los servicios restful de Firebase
3. POST.
4. GET.
5. DELETE.
6. PUT.
7. Configuraciones en la consola de Firebase.

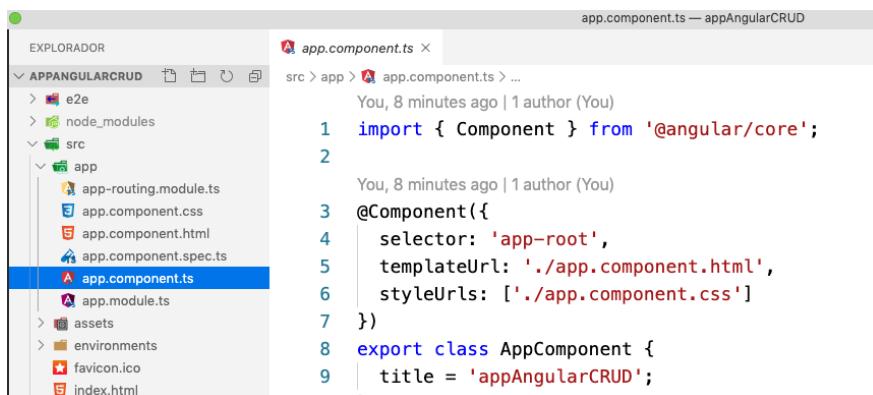
Al final de la sección tenemos un examen teórico.

1. Inicialización del proyecto:



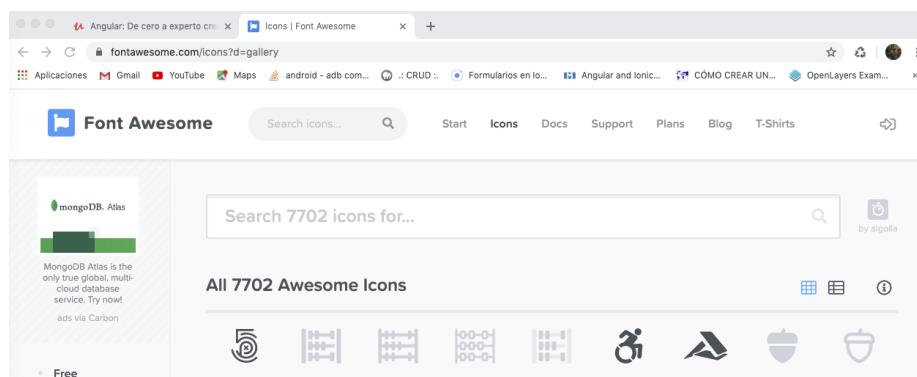
```
Last login: Sat Dec 28 09:01:08 on console
[iMac-de-Jose:~ jsersan$ cd /Users/jsersan/Desktop/Proyectos\ Angular\ \((node\)
[iMac-de-Jose:Proyectos Angular (node) jsersan$ ng new appAngularCRUD
[?] Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE appAngularCRUD/README.md (1032 bytes)
CREATE appAngularCRUD/.editorconfig (246 bytes)
CREATE appAngularCRUD/.gitignore (631 bytes)
CREATE appAngularCRUD/angular.json (3657 bytes)
CREATE appAngularCRUD/package.json (1302 bytes)
CREATE appAngularCRUD/tsconfig.json (543 bytes)
CREATE appAngularCRUD/tslint.json (1953 bytes)
CREATE appAngularCRUD/browserslist (429 bytes)
CREATE appAngularCRUD/karma.conf.js (1026 bytes)
CREATE appAngularCRUD/tsconfig.app.json (270 bytes)
CREATE appAngularCRUD/tsconfig.spec.json (270 bytes)
CREATE appAngularCRUD/src/favicon.ico (948 bytes)
CREATE appAngularCRUD/src/index.html (300 bytes)
CREATE appAngularCRUD/src/main.ts (372 bytes)
CREATE appAngularCRUD/src/polyfills.ts (2838 bytes)
CREATE appAngularCRUD/src/test.ts (204 bytes)
```

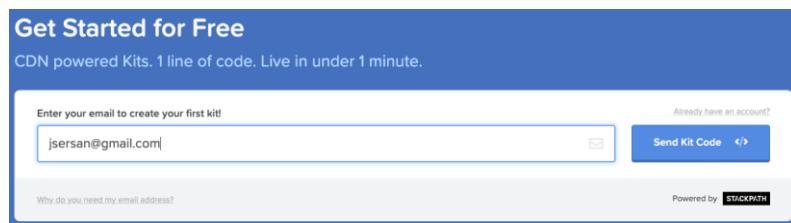
Cuando se termine de generar el proyecto lo abrimos con Visual Studio Code:



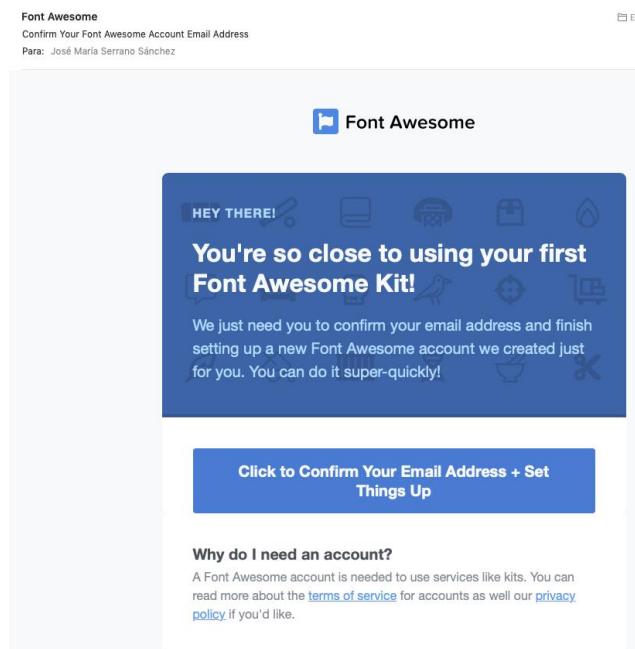
Utilizaremos las siguientes librerías:

a) Fontawesome:

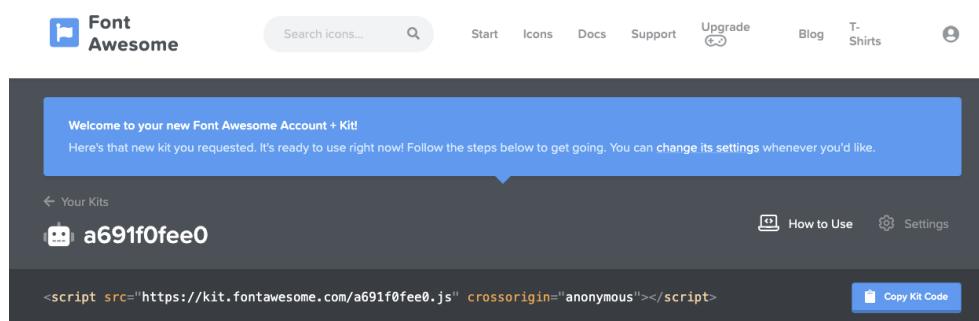




Confirmamos el email:

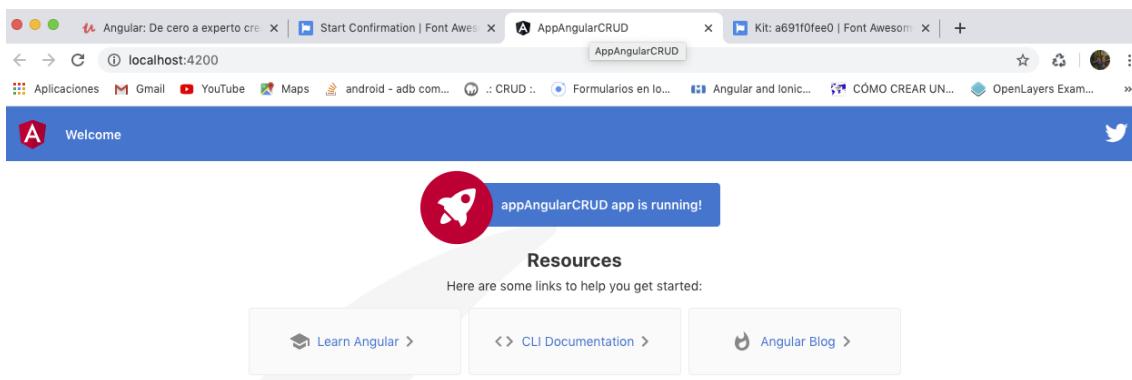


Ya lo tenemos disponible para utilizar:



Lanzamos el proyecto:

\$ ng serve -o



Copio el kitcode de mi Font-Awesome a **index.html**:

CSS

Copy-paste the stylesheet <link> into your <head> before all other stylesheets to load our CSS.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8X4CGsO3+QsH9t+YXnDGtD9TmHx07A7QFj5dIyWmEzDqKIQhQ==" crossorigin="anonymous"></head>
```

```

9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10
11  <!-- Bootstrap -->
12
13  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/
14    css/bootstrap.min.css" integrity="sha384-Vkoo8X4CGsO3+QsH9t+YXnDGtD9TmHx07A7QFj5dIyWmEzDqKIQhQ==" crossorigin="anonymous">
15
16  <!-- Fotawesome -->
17
18  <script src="https://kit.fontawesome.com/a691f0fee0.js" crossorigin="anonymous"></script>

```

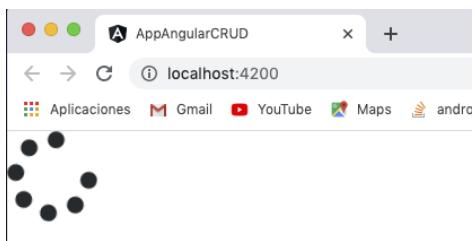
Para comprobar si el font-awesome está trabajando correctamente, abrimos **app.component.html** y la ponemos este contenido:

```

index.html app.component.html
src > app > app.component.html > i.fa.fa-spinner.fa-spin.fa-5x
You, a few seconds ago | 1 author (You)
1
2 | <i class="fa fa-spinner fa-spin fa-5x"></i>

```

El resultado es una rueda que gira lo que nos indica que hemos importado correctamente la librería de font-awesome



Vamos a generar las rutas desde el **angular-cli**:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
iMac-de-Jose:appAngularCRUD jsersan$ ng g m AppRoutingModule --flat
CREATE src/app/app-routing.module.ts (196 bytes)
iMac-de-Jose:appAngularCRUD jsersan$ █
```

Abrimos el archivo:

```
app-routing.module.ts • index.html app.component.html app.module.ts
src > app > app-routing.module.ts > AppRoutingModule
You, a few seconds ago | 1 author (You)
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 @NgModule({
5   declarations: [],
6   imports: [
7     CommonModule
8   ]
9 })
10 export class AppRoutingModule { }
```

Nos creamos un archivo de rutas:

```
iMac-de-Jose:17-angularCRUDfirebase jsersan$ ng g m AppRoutingModule --flat
```

Vamos a quitar el **CommonModule** que gestiona el `ngIf`, `ngFor`, etc. que no vamos a utilizar porque este módulo va a ser el encargado de manejar las rutas:

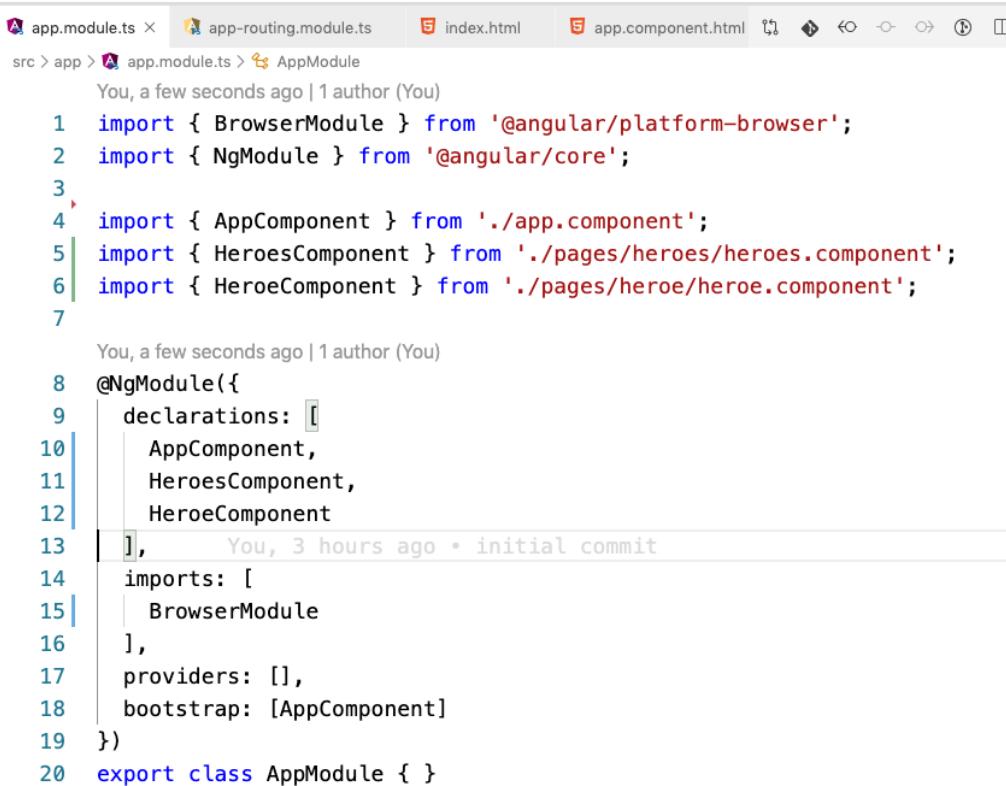
```
app-routing.module.ts • index.html app.component.html app.r
src > app > app-routing.module.ts > ...
You, a few seconds ago | 1 author (You)
1 import { NgModule } from '@angular/core';
2
3
4 @NgModule({
5   imports: []
6 })
7 export class AppRoutingModule { }
```

También quitamos las declaraciones. Esta aplicación va a tener sólo dos componentes: heroe y heroes:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
iMac-de-Jose:appAngularCRUD jsersan$ ng g c pages/heroes --skipTests -is
CREATE src/app/pages/heroes/heroes.component.html (21 bytes)
CREATE src/app/pages/heroes/heroes.component.ts (242 bytes)
UPDATE src/app/app.module.ts (486 bytes)
iMac-de-Jose:appAngularCRUD jsersan$ ng g c pages/heroe --skipTests -is
CREATE src/app/pages/heroe/heroe.component.html (20 bytes)
CREATE src/app/pages/heroe/heroe.component.ts (239 bytes)
UPDATE src/app/app.module.ts (486 bytes)
iMac-de-Jose:appAngularCRUD jsersan$ 
```

De esta manera, **app.module.ts** queda:

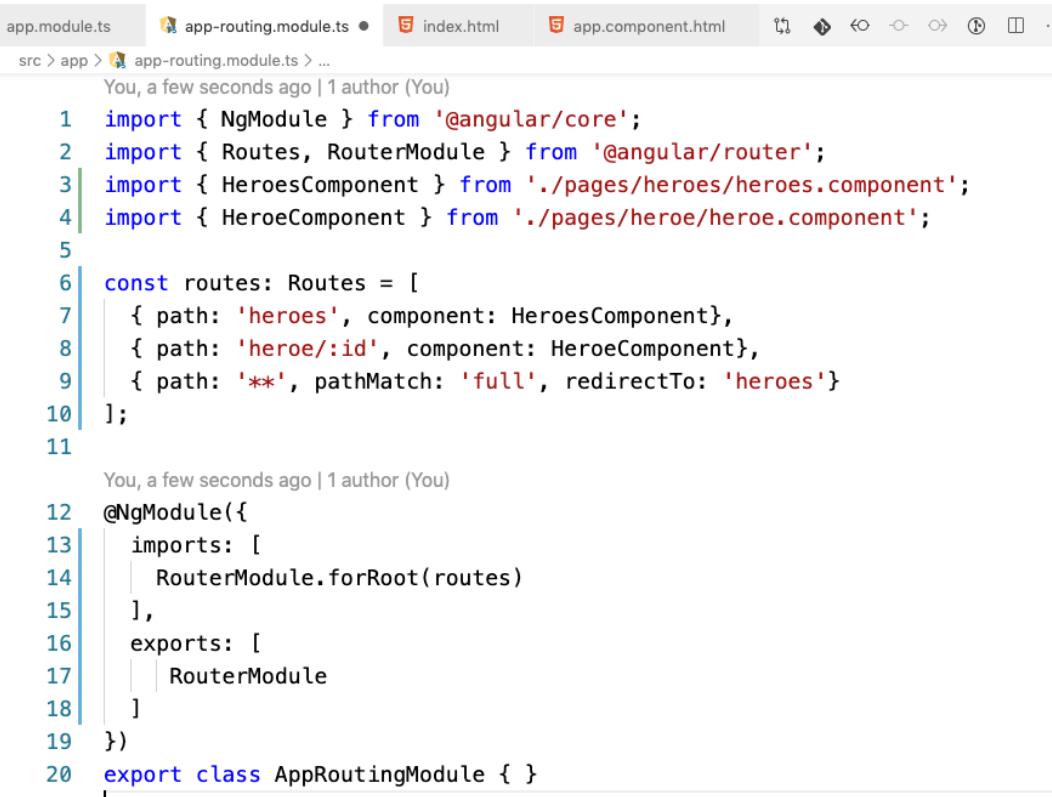


```

1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppComponent } from './app.component';
5 import { HeroesComponent } from './pages/heroes/heroes.component';
6 import { HeroeComponent } from './pages/heroe/heroe.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     HeroesComponent,
12     HeroeComponent
13   ],
14   imports: [
15     BrowserModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }

```

En una de las páginas se van a mostrar todos los héroes y en la otra visualizaremos uno en particular y podremos actualizar y editar. Ahora nos definimos las rutas en **app-routing.module.ts**:



```

1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3 import { HeroesComponent } from './pages/heroes/heroes.component';
4 import { HeroeComponent } from './pages/heroe/heroe.component';
5
6 const routes: Routes = [
7   { path: 'heroes', component: HeroesComponent },
8   { path: 'heroe/:id', component: HeroeComponent },
9   { path: '**', pathMatch: 'full', redirectTo: 'heroes' }
10 ];
11
12 @NgModule({
13   imports: [
14     RouterModule.forRoot(routes)
15   ],
16   exports: [
17     RouterModule
18   ]
19 })
20 export class AppRoutingModule { }

```

El *AppRoutingModule* de la última línea debo incluirlo en **app.modules.ts**:

```

7 import { AppRoutingModule } from './app-routing.module';
8
9 You, a few seconds ago | 1 author (You)
10 @NgModule({
11   declarations: [
12     AppComponent,
13     HeroesComponent,
14     HeroeComponent
15   ],
16   imports: [
17     BrowserModule,
18     AppRoutingModule      You, 4 hours ago • initial commit
19   ],

```

Para probar si las rutas funcionan, vamos a **app.component.html**:

```

app.component.html × app.module.ts × app-routing.modi
src > app > app > app.component.html > div.mt-5
You, a few seconds ago | 1 author (You)
1 <div class="mt-5">
2   <router-outlet>
3
4   </router-outlet>
5 </div> You, a few seconds ago

```

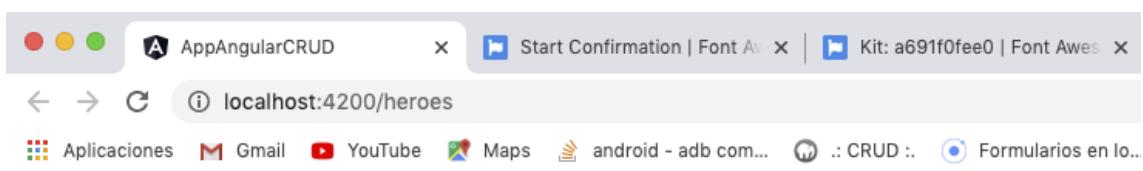
Si sale error:

```

✖ Uncaught Error: Template parse errors:
'router-outlet' is not a known element.
1. If 'router-outlet' is an Angular component, then verify that it is part of this module.
2. If 'router-outlet' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.schemas'
[ERROR ->] <router-outlet>

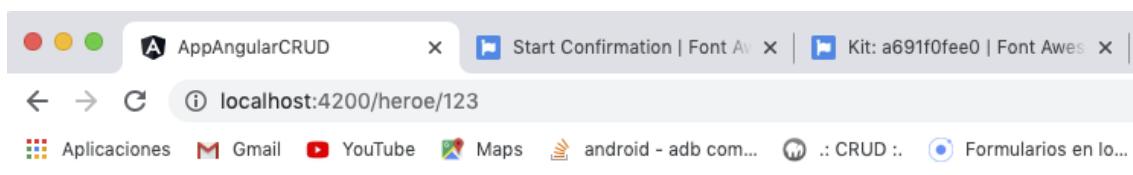
```

Bajamos el servidor y lo volvemos a subir:



heroes works!

Puedo intentar navegar a la página de héroe con cualquier valor en la url:



heroe works!

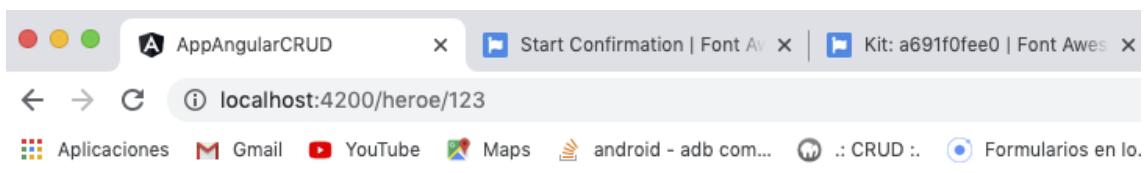
Comprobando que las rutas funcionan correctamente. Vamos a hacer cambios estéticos para evitar que quede muy pegado, en **index.html**:

```

19
20 <body class="container">
21   <app-root></app-root>
22 </body>

```

Guardamos los cambios:



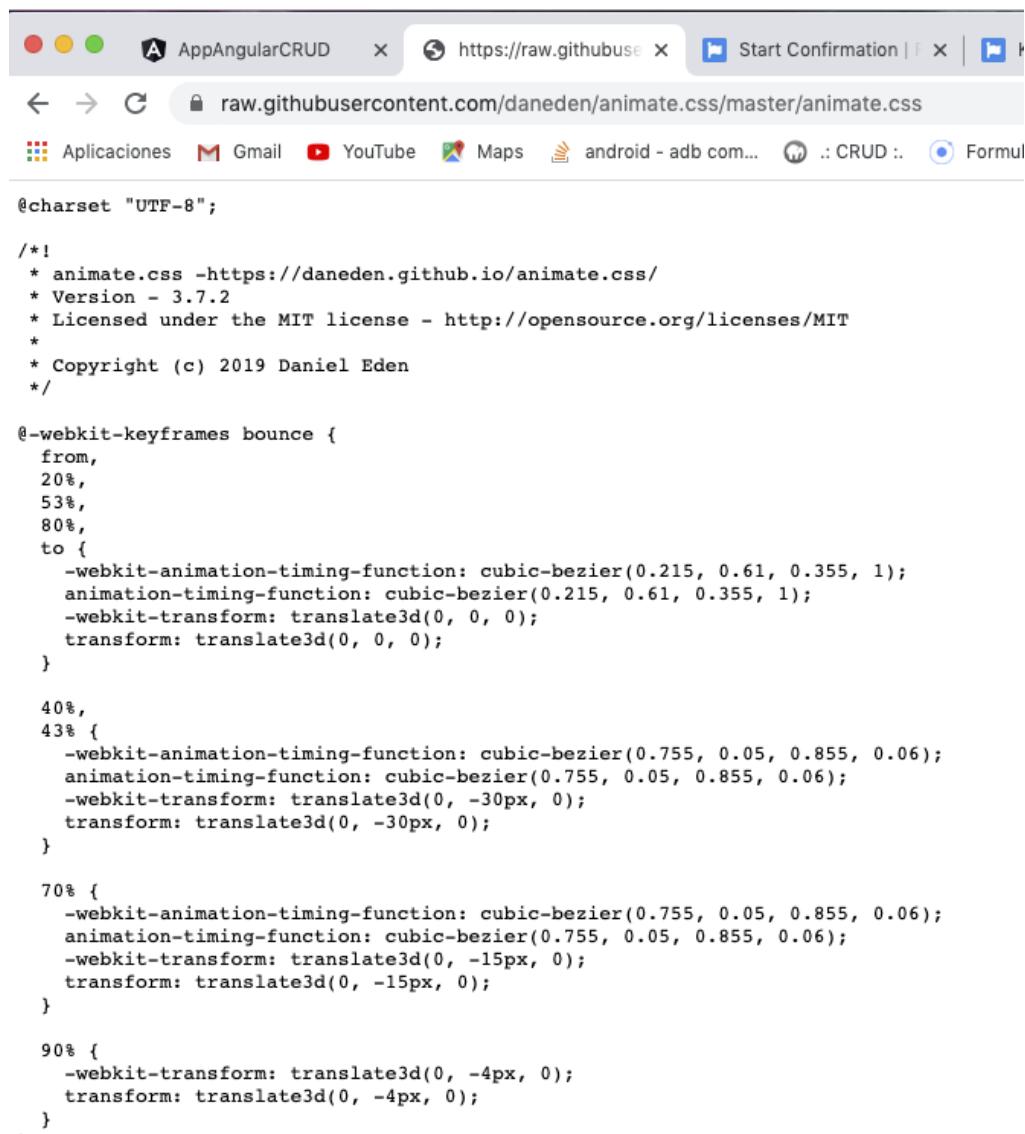
heroe works!

Animación: buscamos en Google **animate.css**:

Download Animate.css or View on GitHub

Another thing from [Daniel Eden](#).

Le damos al enlace **Download Animate.css**:



```

@charset "UTF-8";

/*
 * animate.css -https://daneden.github.io/animate.css/
 * Version - 3.7.2
 * Licensed under the MIT license - http://opensource.org/licenses/MIT
 *
 * Copyright (c) 2019 Daniel Eden
 */

@-webkit-keyframes bounce {
    from,
    20%,
    53%,
    80%,
    to {
        -webkit-animation-timing-function: cubic-bezier(0.215, 0.61, 0.355, 1);
        animation-timing-function: cubic-bezier(0.215, 0.61, 0.355, 1);
        -webkit-transform: translate3d(0, 0, 0);
        transform: translate3d(0, 0, 0);
    }

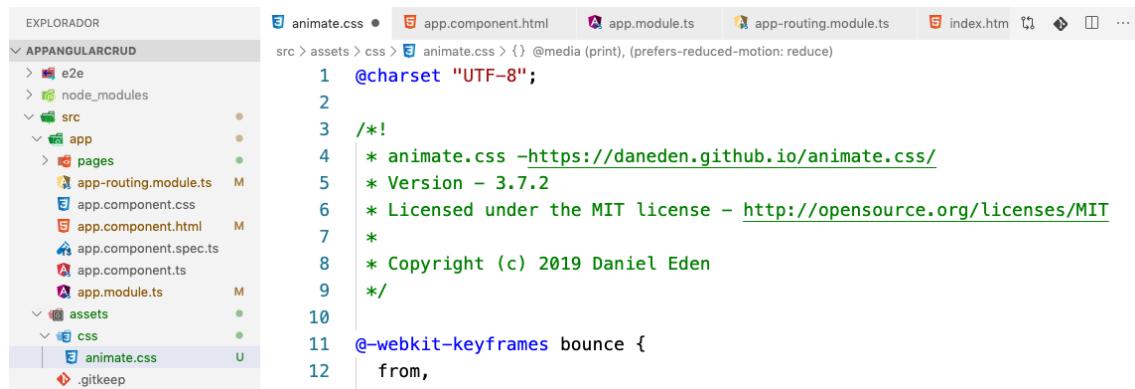
    40%,
    43% {
        -webkit-animation-timing-function: cubic-bezier(0.755, 0.05, 0.855, 0.06);
        animation-timing-function: cubic-bezier(0.755, 0.05, 0.855, 0.06);
        -webkit-transform: translate3d(0, -30px, 0);
        transform: translate3d(0, -30px, 0);
    }

    70% {
        -webkit-animation-timing-function: cubic-bezier(0.755, 0.05, 0.855, 0.06);
        animation-timing-function: cubic-bezier(0.755, 0.05, 0.855, 0.06);
        -webkit-transform: translate3d(0, -15px, 0);
        transform: translate3d(0, -15px, 0);
    }

    90% {
        -webkit-transform: translate3d(0, -4px, 0);
        transform: translate3d(0, -4px, 0);
    }
}

```

Copiamos el contenido y lo guardamos en un fichero **animate.css** que vamos a poner en **assets/css**:



```

EXPLORADOR
APPANGULARCRUD
> e2e
> node_modules
src
  > app
    pages
      app-routing.module.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
    assets
      css
        animate.css
        .gitkeep
src > assets > css > animate.css > {} @media (print), (prefers-reduced-motion: reduce)
1  @charset "UTF-8";
2
3  /*
4   * animate.css -https://daneden.github.io/animate.css/
5   * Version - 3.7.2
6   * Licensed under the MIT license - http://opensource.org/licenses/MIT
7   *
8   * Copyright (c) 2019 Daniel Eden
9   */
10
11 @-webkit-keyframes bounce {
12     from,
13     20%

```

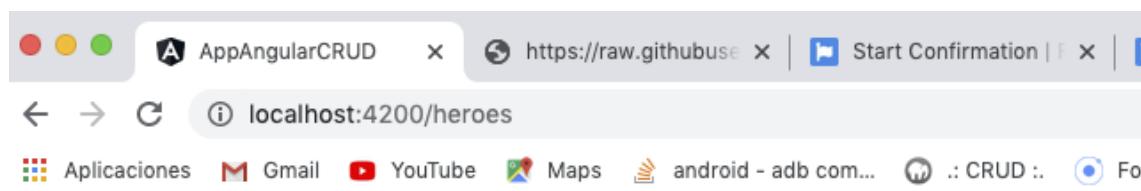
Lo referenciamos en **index.html** y le ponemos una animación al body:

```

index.html • animate.css app.component.html app.module.ts app-routing.module.ts
src > index.html > html > head
You, a few seconds ago | 1 author (You)
1  <!doctype html>
2  <html lang="en">
3
4  <head>
5      <meta charset="utf-8">
6      <title>AppAngularCRUD</title>
7      <base href="/">
8      <meta name="viewport" content="width=device-width, initial-scale=1">
9      <link rel="icon" type="image/x-icon" href="favicon.ico">
10
11     <!-- Bootstrap -->
12
13     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/
bootstrap.min.css" integrity="sha384-Vkoo8x4CGs03+Hhv8T/
Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9Mu0f23Q9Ifjh" crossorigin="anonymous">
14
15     <!-- Fontawesome -->
16
17     <script src="https://kit.fontawesome.com/a691f0fee0.js" crossorigin="anonymous"></script>
18
19     <!-- Animate -->
20
21     <link rel="stylesheet" href=".//assets/css/animate.css">
22 </head> You, 4 hours ago * initial commit
23
24 <body class="container animated fadeIn slow">
25     <app-root></app-root>
26 </body>
27
28 </html>

```

Guardamos los cambios y podemos ver la animación:



heroes works!

Si movéis la pantalla o el papel se ve la animación.



2.- Diseño de la página Héroes:

Diseñamos **heroes.component.html**:

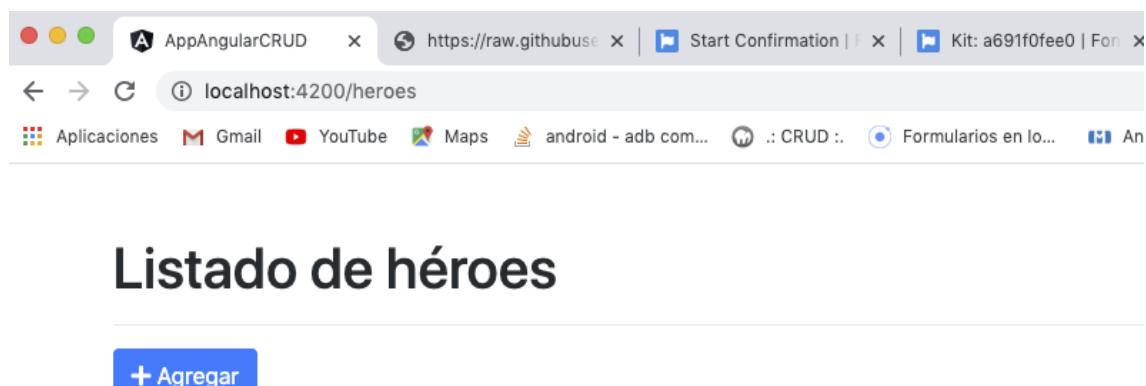


```

1 <h1>Listado de héroes</h1>
2 <hr>
3
4 <div class="row">
5   <div class="col">
6     <button class="btn btn-primary">
7       <i class="fa fa-plus"></i>
8       Agregar
9     </button>
10    </div>
11 </div>
12

```

Resultado:



Ahora debemos generar una tabla donde se ven los héroes. La tomamos de **Bootstrap**:

Table head options

Similar to tables and dark tables, use the modifier classes `.thead-light` or `.thead-dark` to make `<thead>`s appear light or dark gray.

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter
#	First	Last	Handle

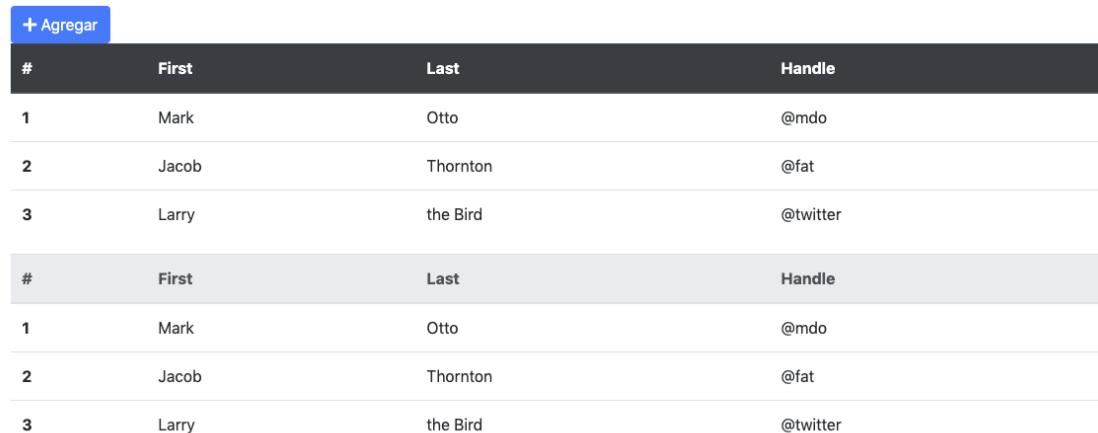
Copio y pego en **app.component.html**. Una parte del código:

```

4  <div class="row">
5      <div class="col">
6          <button class="btn btn-primary">
7              <i class="fa fa-plus"></i>
8              Agregar
9          </button>
10     </div>
11 </div>
12
13 <table class="table">
14     <thead class="thead-dark">
15         <tr>
16             <th scope="col">#</th>
17             <th scope="col">First</th>
18             <th scope="col">Last</th>
19             <th scope="col">Handle</th>
20         </tr>
21     </thead>
22     <tbody>
```

Guardamos los cambios:

Listado de héroes



The screenshot shows a web application interface. At the top left is a blue button labeled '+ Agregar'. Below it is a table with four columns: '#', 'First', 'Last', and 'Handle'. The table has two data rows and one header row. The data rows contain the following information:

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

Le pondremos una separación al botón y alineado a la derecha:

```

13  <table class="table mt-3">
14      <thead class="thead-dark">
15          <tr>
16              <th scope="col">#</th>
17              <th scope="col">First</th>
18              <th scope="col">Last</th>
19              <th scope="col">Handle</th>
20          </tr>
21      </thead>
```

Resultado:

Listado de héroes

+ Agregar			
#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

Cambiamos el título a las columnas y terminamos la página:

```

index.html heroes.component.html animate.css app.component.html app.module.ts
src > app > pages > heroes > heroes.component.html ...
1  <h1>Listado de héroes</h1>
2  <hr>
3
4  <div class="row">
5    <div class="col">
6      <button class="btn btn-primary">
7        <i class="fa fa-plus"></i>
8        Agregar
9      </button>
10     </div>
11   </div>
12
13 <table class="table mt-3 animated fadeIn faster">
14   <thead class="thead-dark">
15     <tr>
16       <th scope="col">#</th>
17       <th scope="col">Nombre</th>
18       <th scope="col">Poder</th>
19       <th scope="col">Vivo</th>
20       <th scope="col">Tools</th>
21     </tr>
22   </thead>
23   <tbody>
24     <tr>
25       <th scope="row">1</th>
26       <td>Mark</td>
27       <td>Otto</td>
28       <td>@mdo</td>
29       <td>@mdo</td>
30     </tr>
31   </tbody>
32 </table>
--
```

Para alinear botón al lado derecho, le añado la clase *text-right*:

```

4  <div class="row">
5      <div class="col text-right">
6          <button class="btn btn-primary">
7              <i class="fa fa-plus"></i>
8              Agregar
9          </button>
10     </div>
11 </div>
```

Resultado:

Listado de héroes



A screenshot of a web application titled "Listado de héroes". At the top right is a blue button labeled "+ Agregar". Below it is a table with columns: #, Nombre, Poder, Vivo, and Tools. There is one row with data: #1, Mark, Otto, @mdo, and @mdo.

#	Nombre	Poder	Vivo	Tools
1	Mark	Otto	@mdo	@mdo

Nosotros replicaremos el <tr> por cada registro que exista en la Base de datos.

Vamos a ser un *loading* para cuando cargue la página. Utilizaré un objeto del **Bootstrap** que es la alerta:

```

34  <div class="alert alert-info text-center mt-3">
35      <h4 class="alert-heading">Cargando</h4>
36      <p>
37          <i class="fa fa-sync-alt fa-spin fa-2x"></i>
38      </p>
39      <p class="mb-0">
40          Espere por favor
41      </p>
42 </div>
```

Resultado:

Listado de héroes



A screenshot of a browser showing the same "Listado de héroes" page. An alert dialog is overlaid on the table, containing the text "Cargando" with a circular progress icon, and "Espere por favor" below it.

#	Nombre	Poder	Vivo	Tools
1	Mark	Otto	@mdo	@mdo

Creamos otro:

```
34 <div class="alert alert-warning text-center mt-3">
35   <h4 class="alert-heading">No hay registros</h4>
36   <p>
37     <i class="fa fa-exclamation fa-2x"></i>
38   </p>
39 </div>
```

Guardamos los cambios:

Listado de héroes

The screenshot shows a table with columns: #, Nombre, Poder, Vivo, and Tools. There is one row with data: #1, Mark, Otto, @mdo, and @mdo. Below the table, there is a yellow box containing the text "No hay registros" with an exclamation mark.

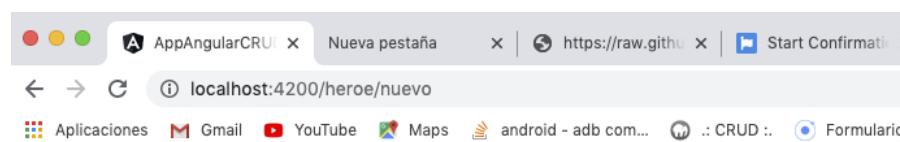
#	Nombre	Poder	Vivo	Tools
1	Mark	Otto	@mdo	@mdo

No hay registros !

Ahora enlazamos el botón con el formulario de Edición de Héroes en la página de héroes:

```
4 <div class="row">
5   <div class="col text-right">
6     <button routerLink="/heroe/nuevo"
7       class="btn btn-primary">
8       <i class="fa fa-plus"></i>
9       Agregar
10      </button>
11    </div>
12 </div>
```

Probamos:



3.- Diseño de la página Héroe:

```

hero.component.html ● index.html heroes.component.html animate.css app.compr
src > app > pages > hero > hero.component.html > div.row.text-right > div.col > button.btn.btn-danger
1 <h1>Héroe <small>Nombre del héroe</small></h1>
2 <hr>
3
4 <div class="row text-right">
5   <div class="col">
6     <button class="btn btn-danger">
7       <i class="fa fa-arrow-left"></i>
8       Regresar
9     </button>
10    </div>
11 </div>

```

Guardamos los cambios:

Héroe Nombre del héroe

[← Regresar](#)

Ese botón tiene que regresar, agregamos *routerLink*:

```

hero.component.html ● index.html heroes.component.html animate.css
src > app > pages > hero > hero.component.html > div.row.text-right
1 <h1>Héroe <small>Nombre del héroe</small></h1>
2 <hr>
3
4 <div class="row text-right">
5   <div class="col">
6     <button routerLink="/heroes"
7       class="btn btn-danger">
8       <i class="fa fa-arrow-left"></i>
9       Regresar
10      </button>
11    </div>
12 </div>

```

A continuación, realizamos la pantalla del formulario de héroes. Los campos a editar son el nombre, el poder, si está vivo o no y las tools.

En `heroe.component.html`:

```
<div class="form-group">
  <label>Firebase ID</label>
  <input type="text" class="form-control"
    placeholder="Firebase ID" [(ngModel)]="heroe.id"
    name="id" disabled="disabled">
  <small class="form-text text-muted">Este campo es
  autogenerado</small>
</div>
```

Resultado:

Firebase ID

Este campo es obligatorio

[← Regresar](#)

Clonamos este código para cada uno de los campos del formulario:

```
<div class="form-group">
  <label>Firebase ID</label>
  <input type="text" class="form-control"
    placeholder="Firebase ID" [(ngModel)]="heroe.id"
    name="id" disabled="disabled">
  <small class="form-text text-muted">Este campo es
  autogenerado</small>
</div>
```

Hacemos unas pequeñas modificaciones para el resto de los campos:

```
<div class="form-group">
  <label>Nombre</label>
  <input type="text" class="form-control"
    placeholder="Nombre del héroe"
    required>
</div>
```

Y así el resto de los campos:

```
<div class="form-group">
    <label>Poder</label>
    <input type="text" class="form-control" placeholder="Poder del héroe">
</div>

<div class="form-group">
    <label>Estado</label>
    <br>
    <button class="btn btn-outline-success w-50" type="button">
        <i class="fa fa-smile-wink"></i>
        Vivo
    </button>
</div>
```

Copiamos el botón para la opción de muerto:

```
<button class="btn btn-outline-danger w-50" type="button">
    <i class="fa fa-dizzy"></i>
    Muerto
</button>
```

Resultado:

Héroe Nombre del héroe

Este campo es obligatorio

Nombre

Poder

Estado

Vivo Muerto

Ahora nos hace falta un botón para el posteo de la información:

```
<hr>
<div class="form-group text-center">
    <button type="submit"
            class="btn btn-primary w-25">
        <i class="fa fa-save"></i>
        Guardar
    </button>
</div>
```

El resultado:

Héroe Nombre del héroe

[← Regresar](#)

Firebase ID

Este campo es obligatorio

Nombre

Poder

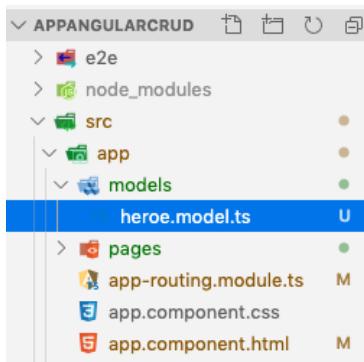
Estado

Vivo Muerto

[Guardar](#)

4.-Modelo para manejar Héroes y formulario de edición

Los modelos nos permiten controlar la información de los formularios. Cerramos todo y en la carpeta app creamos otra que se llamará **models**:



El código es el siguiente:

```

2  export class HeroeModel {
3
4      id: string;
5      nombre: string;
6      poder: string;
7      vivo: boolean;
8
9      constructor() {
10         this.vivo = true;
11     }
12
13 }
```

En nuestra página de **heroe.component.ts** nos creamos una nueva propiedad que se llamará **heroe**:

```

src > app > pages > hero > heroe.component.ts > HeroeComponent > hero
1  import { Component, OnInit } from '@angular/core';
2  import { HeroeModel } from '../../../../../models/heroe.model';
3
4  @Component({
5      selector: 'app-hero',
6      templateUrl: './heroe.component.html'
7  })
8  export class HeroeComponent implements OnInit {
9
10    hero: HeroeModel = new HeroeModel();
11
12    constructor() { }
13
14    ngOnInit() {
15    }
16
17 }
```

En **heroе.component.html** queremos asociar cada uno de los campos del formulario con una propiedad de la clase Heroe y trabajar el *submit* del formulario. Como estamos trabajando con formularios debemos hacer unos *imports* en **app.module.ts**:

```

4 import { FormsModule } from '@angular/forms';
5
6 import { AppComponent } from './app.component';
7 import { HeroesComponent } from './pages/heroes/heroes.component';
8 import { HeroeComponent } from './pages/heroе/heroе.component';
9 import { AppRoutingModule } from './app-routing.module';
10
11 You, a few seconds ago | 1 author (You)
12 @NgModule({
13   declarations: [
14     AppComponent,
15     HeroesComponent,
16     HeroeComponent
17   ],
18   imports: [
19     BrowserModule,
20     AppRoutingModule,
21     FormsModule
22   ]
23 })
```

Ahora lo que toca es que el botón Guardar del formulario no haga el *submit* del formulario sino que invoque a una función **guardar()** desarrollada por nosotros mismos en **heroе.component.html**.

```

13 <div class="row animated fadein faster">
14   <div class="col">
15     <form (ngSubmit)="guardar(f) " |
16       #f = "ngForm">
17       <div class="form-group">
```

Implementamos este método guardar en el *typeScript*: **heroе.component.ts**:

```

15 ngOnInit() {
16 }
17
18 guardar( form: NgForm ) {
19   console.log(form);
20   console.log(this.heroe);
21 }
22 }
```

ngModel nos va a dar un error en el parámetro porque necesita el atributo *name*. Para saber qué nombre de propiedad le va a asignar a cada uno de los campos de **heroе.component.html**.

```

<div class="form-group">
  <label>Firebase ID</label>
  <input type="text"
    class="form-control"
    [(ngModel)]="heroe.id"
    name="id"
    placeholder="Firebase ID"
    disabled="disabled">
  <small class="form-text text-muted">Este campo es obligatorio</small>
</div>
```

Para el nombre:

```
<div class="form-group">
  <label>Nombre</label>
  <input type="text"
    class="form-control"
    [(ngModel)]="heroe.nombre"
    name="nombre"
    placeholder="Nombre del héroe" required>
</div>
```

Para el poder:

```
<div class="form-group">
  <label>Poder</label>
  <input type="text"
    class="form-control"
    [(ngModel)]="heroe.poder"
    name="poder"
    placeholder="Poder del héroe">
</div>
```

Con *estado* lo que vamos a hacer es que si está vivo muestra el botón vivo y si está muerto el otro botón. Lo haremos con *ngIf*:

```
<div class="form-group">
  <label>Estado</label>
  <br>
  <button *ngIf="heroe.vivo"
    (click) = "heroe.vivo = false"
    class="btn btn-outline-success w-50" type="button">
    <i class="fa fa-smile-wink"></i>
    Vivo
  </button>

  <button *ngIf="!heroe.vivo"
    (click) = "heroe.vivo = true"
    class="btn btn-outline-danger w-50" type="button">
    <i class="fa fa-dizzy"></i>
    Muerto
  </button>
</div>
```

Lo probamos:

Héroe Nombre del héroe

Firebase ID

Este campo es obligatorio

Nombre

Poder

Estado

Vivo

Guardar

Tras darle al botón de Vivo para que cambie:

Héroe Nombre del héroe

The screenshot shows a web form titled "Héroe Nombre del héroe". At the top right is a red "Regresar" button. Below it is a "Firebase ID" input field with placeholder text "Firebase ID" and a note "Este campo es obligatorio". The "Nombre" field contains "Nombre del héroe". The "Poder" field contains "Poder del héroe". The "Estado" field has a dropdown menu with two options: "Muerto" (selected) and "Vivo". A blue "Guardar" button is at the bottom.

Probamos un formulario cualquiera:

Héroe Nombre del héroe

The screenshot shows a similar web form. The "Nombre" field contains "Txeminator". The "Poder" field contains "Incansable". The "Estado" field has a dropdown menu with two options: "Muerto" (disabled) and "Vivo" (selected). A blue "Guardar" button is at the bottom.

El contenido de la consola:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
A cookie associated with a cross-site resource at http://fontawesome.com was set without the
only deliver cookies with cross-site requests if they are set with 'SameSite=None' and 'Secure'.
Application>Storage>Cookies and see more details at https://www.chromestatus.com/feature/50882188032.
[WDS] Live Reloading enabled.
> NgForm {submitted: true, _directives: Array(3), ngSubmit: EventEmitter, form: FormGroup}
  > HeroeModel {vivo: true, nombre: "Txeminator", poder: "Incansable"} ⓘ
    vivo: true
    nombre: "Txeminator"
    poder: "Incansable"
    > __proto__: Object
```

El id del héroe nos lo va a proporcionar *Firebase*. Vamos a hacer que retorne si el formulario no es válido, en **heroе.component.ts**:

```

19  guardar( form: NgForm ) {
20
21    if ( form.invalid ) {
22      console.log('Formulario no válido');
23      return;
24
25    }
26    console.log(form);
27    console.log(this.heroe);
28
29 }

```

Hay un campo que es requerido: el nombre. Si pulsamos el botón:

Héroe Nombre del héroe

The form consists of several input fields:

- Nombre:** A text input field labeled "Nombre del héroe" which is currently empty.
- Poder:** A text input field labeled "Poder del héroe" which contains placeholder text.
- Estado:** A dropdown menu with an option selected: "Vivo".

At the top right is a red button labeled "Regresar" (Return). At the bottom is a blue button labeled "Guardar" (Save).

Y en la consola me sale un mensaje de *Formulario no válido*:

```

Angular is running in the development mode. Call enableProdMode() in your
main.ts file to enable the production mode.
A cookie associated with a cross-site resource at https://angularfire2.com only
delivers cookies with cross-site requests if the browser supports it. See
Application>Storage>Cookies and see more details at
https://github.com/angular/angular/issues/21880#issuecomment-2188032.
[WDS] Live Reloading enabled.
Formulario no válido

```

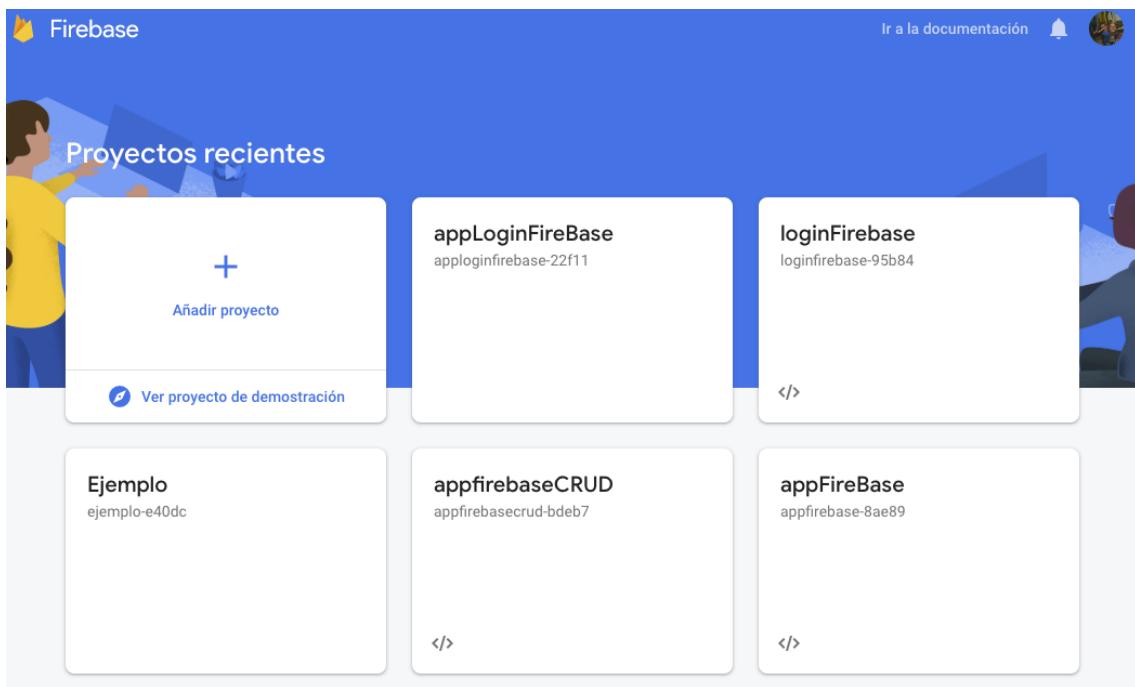
A modo de resumen, el formulario principal en **heroe.component.html**:

```

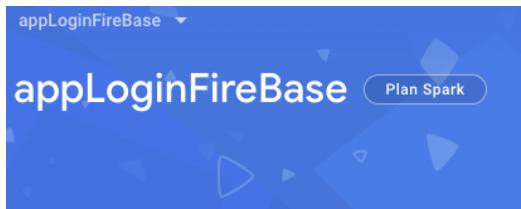
-->
13  <div class="row animated fadeIn faster">
14      <div class="col">
15
16          <form (ngSubmit)="guardar( f )" #f="ngForm">
17
18              <div class="form-group">
19                  <label>Firebase ID</label>
20                  <input type="text" class="form-control" placeholder="Firebase ID"
21                      [(ngModel)]="heroe.id" name="id" disabled="disabled">
22                      <small class="form-text text-muted">Este campo es autogenerado</small>
23              </div>
24
25              <div class="form-group">
26                  <label>Nombre</label>
27                  <input type="text" class="form-control" placeholder="Nombre del héroe"
28                      [(ngModel)]="heroe.nombre" name="nombre" required>
29              </div>
30
31              <div class="form-group">
32                  <label>Poder</label>
33                  <input type="text" class="form-control"
34                      [(ngModel)]="heroe.poder" name="poder" placeholder="Poder del héroe">
35              </div>
36
37              <div class="form-group">
38                  <label>Estado</label>
39                  <br>
40                  <button *ngIf="heroe.vivo" (click)="heroe.vivo = false"
41                      class="btn btn-outline-success w-50" type="button">
42                      <i class="fa fa-smile-wink"></i>
43                      Vivo
44                  </button>
45
46                  <button *ngIf="!heroe.vivo" (click)="heroe.vivo = true"
47                      class="btn btn-outline-danger w-50" type="button">
48                      <i class="fa fa-dizzy"></i>
49                      Muerto
50                  </button>
51
52              </div>
53
54              <hr>
55              <div class="form-group text-center">
56                  <button type="submit" class="btn btn-primary w-25">
57                      <i class="fa fa-save"></i>
58                      Guardar
59                  </button>
60              </div>
61          </form>

```

5.-Configuración de firebase como backend REST.



Podemos crear uno nuevo, o coger uno ya existente. En mi caso voy a coger appLoginFireBase:



Le damos a Database y buscamos Realtime Database:

The left sidebar shows 'Desarrollo' (Development) mode selected, with options for 'Authentication', 'Database', and 'Storage'. The main area displays the 'Realtime Database' configuration, featuring a diagram of a database structure with nodes and values. Below it, there is a brief description of Realtime Database and a 'Crear base de datos' (Create database) button.

Realtime Database

Base de datos original de Firebase. Al igual que Cloud Firestore, admite la sincronización de datos en tiempo real.

[Consultar la documentación](#) [Más información](#)

[Crear base de datos](#)

Hacemos click en crear base de datos:



Seleccionamos el *Modo de Prueba*.

⚠ Tus reglas de seguridad están definidas como públicas, así que cualquiera puede robar, modificar o eliminar datos de tu base de datos.

Más información Cerrar

Debemos asegurar que las reglas estén así para que todo el mundo pueda leer y escribir, verificamos en la pestaña de reglas:

```

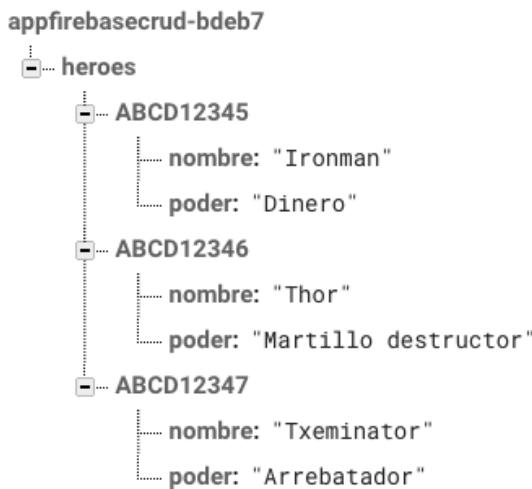
1 ▾   {
2 ▾     "rules": {
3       ".read": true,
4       ".write": true
5     }
6   }

```

Empezamos a trabajar con los datos:

Creamos la misma:

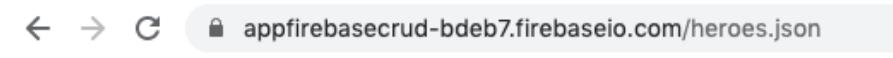
Le damos a añadir. El resultado es un nodo llamado Héroes donde voy a colocar una colección de héroes:



Si quisiéramos tener la colección de todos los Héroes, copiamos la siguiente URL:

<https://apploginfirebase-22f11.firebaseio.com/heroes>

Copiamos esta url en el navegador, le añado la extensión .json:



```
{  
  - ABCD12345: {  
      nombre: "Ironman",  
      poder: "Dinero",  
    },  
  - ABCD12346: {  
      nombre: "Thor",  
      poder: "Martillo destructor",  
    },  
  - ABCD12347: {  
      nombre: "Txeminator",  
      poder: "Arrebatador",  
    },  
}
```

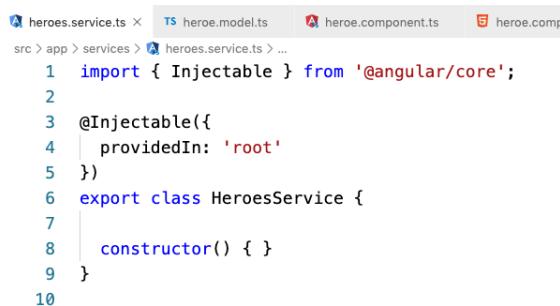
Refrescamos el navegador y vemos el objeto JSON de *heroes*. No es un array. Vamos a tener que transformar esta información para poder añadirlo a un ngFor.

6.-HTTP POST. Creando un nuevo registro en la Base de Datos.

Ya tenemos las páginas creadas. Debemos darle la lógica o la programación que el lugar ideal es un servicio. Creamos el servicio:

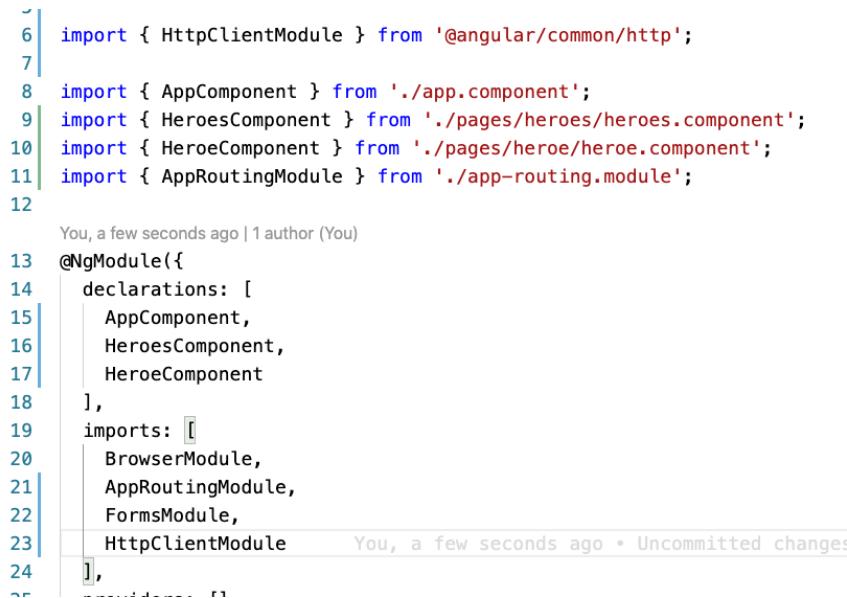
```
CREATE src/app/app.module.ts (400 bytes)
iMac-de-Jose:appAngularCRUD jsersan$ ng g s services/heroes
CREATE src/app/services/heroes.service.spec.ts (333 bytes)
CREATE src/app/services/heroes.service.ts (135 bytes)
iMac-de-Jose:appAngularCRUD jsersan$ █
```

Pasamos a programarlo:



```
heroes.service.ts < hero.model.ts hero.component.ts hero.comp
src > app > services > heroes.service.ts > ...
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class HeroesService {
7
8   constructor() { }
9 }
10
```

Como este servicio se va encargar de realizar peticiones http necesitamos importar el HttpClientModule en **app.module.ts**:



```
6 import { HttpClientModule } from '@angular/common/http';
7
8 import { AppComponent } from './app.component';
9 import { HeroesComponent } from './pages/heroes/heroes.component';
10 import { HeroeComponent } from './pages/heroe/heroe.component';
11 import { AppRoutingModule } from './app-routing.module';
12
13 You, a few seconds ago | 1 author (You)
14 @NgModule({
15   declarations: [
16     AppComponent,
17     HeroesComponent,
18     HeroeComponent
19   ],
20   imports: [
21     BrowserModule,
22     AppRoutingModule,
23     FormsModule,
24     HttpClientModule You, a few seconds ago • Uncommitted changes
25   ],
26   providers: []
27 })
```

Ahora lo debemos inyectar en nuestro servicio:

```
2 import { HttpClient } from '@angular/common/http';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class HeroesService {
8
9   constructor( private http: HttpClient) { }
10 }
```

Creamos una función crearHeroe() que realiza un post a una url de firebase:

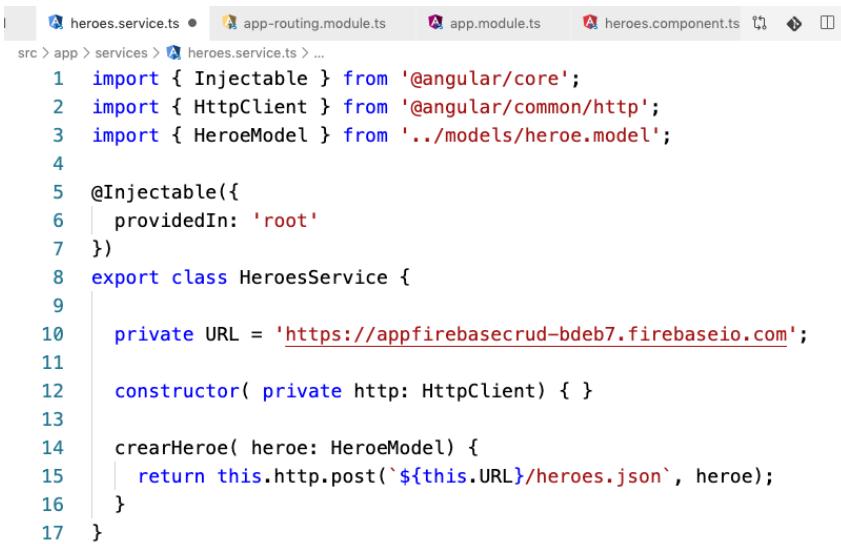
```
crearHeroe( hero: HeroeModel ) {
  return this.http.post(`https://appfirebasecrud-bdeb7.firebaseio.com/heroes.json`, hero);
}

(method) HttpClient.post(url: string, body: any, options: {
  headers?: HttpHeaders | {
    [header: string]: string | string[];
  };
  observe?: "body";
  params?: HttpParams | {
    [param: string]: string | string[];
  };
  reportProgress?: boolean;
})
```

Esta url es dentro de mi firebase:

<https://appfirebasecrud-bdeb7.firebaseio.com/>

Así, el servicio:



```
heroes.service.ts ● app-routing.module.ts ● app.module.ts ● heroes.component.ts
src > app > services > heroes.service.ts > ...
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { HeroeModel } from '../models/heroe.model';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class HeroesService {
9
10   private URL = 'https://appfirebasecrud-bdeb7.firebaseio.com';
11
12   constructor( private http: HttpClient ) { }
13
14   crearHeroe( hero: HeroeModel ) {
15     return this.http.post(`${this.URL}/heroes.json`, hero);
16   }
17 }
```

Aquí la respuesta va a ser el id único que está dentro de la base de datos. Vamos a **heroe.component.ts** e inyectamos el servicio en el constructor:

```
4 import { HeroesService } from '../../../../../services/heroes.service';
5
6 @Component({
7   selector: 'app-hero',
8   templateUrl: './heroe.component.html',
9   styles: []
10})
11 export class HeroeComponent implements OnInit {
12
13   heroe = new HeroeModel();
14
15   constructor( private heroesService: HeroesService ) { }
16
17   ngOnInit() {
18 }
```

Ahora lo que tenemos que hacer es llamar a través de este servicio el método de crear en `heroe.component.ts`.

```

23   |   |   console.log('Formulario no válido');
24   |   |   return;
25   |
26   | }
27   | this.heroesService.crearHeroe()
28

```

crearHeroe(`heroe: HeroeModel`):
Observable<Object>

Me pide que le pase un objeto heroe de tipo `HeroeModel`, pero ya tengo uno que es `this.heroe`:

```

-- 
27   |   |   this.heroesService.crearHeroe(this.heroe)
28   |   |   .subscribe ( resp => {
29   |   |   |   console.log(resp);
30   |   |   });

```

Lo probamos y nos da el error de siempre esperado cuando programamos servicios:

```

✖ > ERROR TypeError: this.heroesService.crearHeroe is not a function
    at HeroeComponent.guardar (heroe.component.ts:27)
    at Object.eval [as handleEvent] (HeroeComponent.html:15)
    at handleEvent (core.js:43993)
    at callWithDebugContext (core.js:45632)
    at Object.debugHandleEvent [as handleEvent] (core.js:45247)
    at dispatchEvent (core.js:29804)
    at core.js:31837
    at SafeSubscriber.schedulerFn [as _next] (core.js:35379)
    at SafeSubscriber._tryOrUnsub (Subscriber.js:185)
    at SafeSubscriber.next (Subscriber.js:124)
✖ > ERROR CONTEXT > DebugContext_ {view: {}, nodeIndex: 13, nodeDef: {}, elDef: {}, elView: {}}

```

Bajamos el servidor y lo volvemos a subir:

Héroe Nombre del héroe

← Regresar

Firebase ID

Este campo es obligatorio

Nombre

Poder

Estado

Guardar

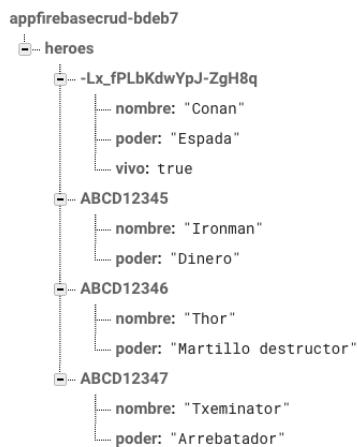
En la consola:

```

Angular is running in the development mode. Call enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.
▶ {name: "-LxCyqfLM21oaImedVM9"}

```

Comprobamos en *firebase*:



Por ahora todo bien, pero si vuelvo a tocar el botón, lo incluirá DUPLICADO con otro id:

Héroe Nombre del héroe

[← Regresar](#)

Firebase ID

Este campo es obligatorio

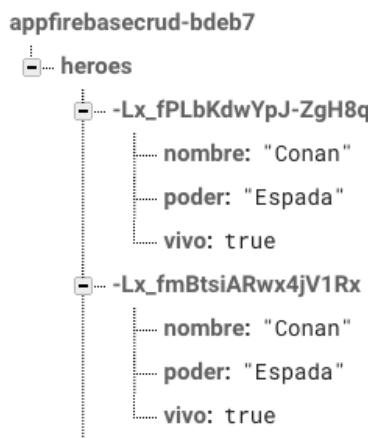
Nombre

Poder

Estado
 Muerto

Guarda

Y en *firebase* los dos valores duplicados:



Para evitar este problema, pregunto si el superhéroe ya tiene un id. En este caso sólo tengo que actualizar. Para manejar este problema, en el servicio sé que cuando la petición de `crearHeroe()` se haga, me va a retornar la id del héroe.

Utilizaremos el operador map en el servicio para transformar esa respuesta. Debemos importar antes el operador map:

```
heroes.service.ts x app-routing.module.ts app.module.ts heroes.component.ts
src > app > services > heroes.service.ts > HeroesService > crearHeroe > map() callback
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { HeroeModel } from '../models/heroe.model';
4 import { map } from 'rxjs/operators';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class HeroesService {
10
11   private URL = 'https://appfirebasecrud-bdeb7.firebaseio.com';
12
13   constructor( private http: HttpClient) { }
14
15   crearHeroe( heroe: HeroeModel) {
16     return this.http.post(` ${this.URL}/heroes.json` , heroe)
17       .pipe(
18         map((resp: any ) => {
19           heroe.id = resp.name;
20           return heroe;
21         })
22       );
23   }
24 }
```

Ahora a retornar toda la instancia del héroe. Probamos de nuevo:

Héroe Nombre del Héroe

Firebase ID
-Lx_hwQaJ_LhtbpnYRgJ
Este campo es autogenerado

Nombre
Hulk

Poder
Poder verde

Estado
Vivo

Guardar

En la consola:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.  
[WDS] Live Reloading enabled.  
▶ NgForm {submitted: true, _directives: Array(3), ngSubmit: EventEmitter, form: FormGroup}  
▶ HeroeModel {vivo: true, nombre: "Hulk", poder: "Poder verde"}  
▼ HeroeModel {vivo: true, nombre: "Hulk", poder: "Poder verde", id: "-Lx_hwQaJ_LhtbpnYRgJ"} ⓘ  
  vivo: true  
  nombre: "Hulk"  
  poder: "Poder verde"  
  id: "-Lx_hwQaJ_LhtbpnYRgJ"  
▶ __proto__: Object
```

Si por algún problema el servicio no me crea el id, podemos forzarlo en el componente **heroe.component.ts**

```
--  
19  guardar( form: NgForm ) {  
20  
21    if ( form.invalid ) {  
22      console.log('Formulario no válido');  
23      return;  
24    }  
25    console.log(form);  
26    console.log(this.heroe);  
27    this.heroesService.crearHeroe(this.heroe)  
28      .subscribe( resp => {  
29        console.log(resp);  
30  
31        // Si no crea el id en el servicio podemos poner:  
32  
33        // this.heroe = resp;  
34      });  
35  }  
36  
37 }  
--
```

En principio esto no sería necesario ya que en JavaScript todos los objetos son pasados por referencia.

7.-HTTP PUT. Actualizar la Base de Datos.

El id que aparece el que tengo que encontrar para saber si quiero insertar uno nuevo o actualizar uno existente. Para actualizarlo, en el servicio creamos una función casi igual a **crearHeroe()** que se llamará **actualizarHeroe()**.

Es más fácil porque el id del héroe ya existe y no debemos hacer ningún pipe. Debemos hacer un put a un registro en particular, desde la url con todo el id:



```

23
24     actualizarHeroe(heroe: HeroeModel) {
25
26
27     // Método pipe de los observable transforma una data en otra con un formato deseado
28
29     return this.http.put(` ${this.URL}/heroes/${heroe.id}.json`, heroe);
30   }
31
32
33
34
35 }
```

Debe apuntar a toda la url de la aplicación más el id del héroe. Entonces en **heroes.service.ts**:

Es aconsejable ponerle la extensión .json para que utilice el RCPI de FireBase. Hay que tener en cuenta que el heroe, tiene todos los campos tal y como indica el modelo. El id no lo vamos a necesitar porque no es un campo propiamente dicho tal y como podemos observar en la captura de firebase. El id pertenece al URL. Entonces, debemos determinar cuando llamar a *crearHeroe* o *actualizarHeroe* desde app.component.html. En **hero.component.ts** en la función **guardar()**:

```

27
28   if (this.heroe.id) {
29     // Actualizar
30     this.heroesService.actualizarHeroe( this.heroe)
31       .subscribe( resp => {
32         console.log(resp);
33       });
34
35   } else {
36     // crearHeroe
37     this.heroesService.crearHeroe(this.heroe)
38       .subscribe ( resp => {
39         console.log(resp);
40         this.heroe = resp;
41       });
42   }
43 }
```

Borramos toda la base de datos para empezar de cero e inserto Ironman:

https://appfirebasecrud-bdeb7.firebaseio.com/

appfirebasecrud-bdeb7: null + ×

Firebase ID
-Lx_oXla7TBUt18HokQK
Este campo es autogenerado

Nombre
Ironman

Poder
Dinero

Estado
Muerto

Resp:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.  
[WDS] Live Reloading enabled.  
▶ HeroeModel {vivo: false, nombre: "Ironman", poder: "Dinero", id: "-Lx_oXla7TBUt18HokQK"}
```

En FireBase:

https://appfirebasecrud-bdeb7.firebaseio.com/

appfirebasecrud-bdeb7

heroes

-Lx_oXla7TBUt18HokQK

nombre: "Ironman"

poder: "Dinero"

vivo: false

Le cambiamos el nombre del superhéroe, poder y vivo: le damos a Guardar:

The form contains the following fields:

- Firebase ID:** -LxDCXRzN8NPdRSKeK-a
- Nombre:** Ironman 2
- Poder:** Forrado hasta las trancas
- Estado:** Vivo
- Guardar:** A blue button.

La respuesta es sólo dar el id del usuario como podemos ver en el código.

```
▼ HeroeModel {vivo: false, nombre: "Ironman", poder: "Dinero", id: "-LxDCXRzN8NPdRSKeK-a"} ⓘ
  vivo: true
  nombre: "Ironman 2"
  poder: "Forrado hasta las trancas"
  id: "-LxDCXRzN8NPdRSKeK-a"
▶ __proto__: Object
▶ {name: "-LxDccInPmM0jLMXMLE"}
```

En la base de datos actualiza la información, pero me añadió el id. De hora en adelante todos los objetos que sean actualizados tendrán este id. Esta propiedad id no debo enviarla a fireBase.

```
apploginfirebase-22f11
  heroes
    -LxDCXRzN8NPdRSKeK-a
      -LxDDhs1I-OW2kWU4GeN
        id: "-LxDCXRzN8NPdRSKeK-a"
        nombre: "Ironman"
        poder: "Dinero"
        vivo: false
```

Para que no lo incluya, en el servicio:

```
29 actualizarHeroe( heroe: HeroeModel ) {
30
31   // Me creo un heroe temporal
32
33   const heroeTemp = {
34     ...heroe           // Copia todas las propiedades
35   };
36
37   // Ahora no tenemos la referencia del javascript. Puedo borrar la propiedad id
38
39   delete heroeTemp.id;
40
41   // Método pipe de los observable que transforma una data en otra con un formato deseado
42   return this.http.post(` ${this.URL}/heroes/${ heroe.id }.json`, heroeTemp);
43
44 }
```

Para probarlo, eliminamos completamente la base de datos:

Probamos:

Firebase ID

Este campo es autogenerado

Nombre

Poder

Estado

Vivo

La base de datos:

```

appfirebasecrud-bdeb7
  heroes
    -Lx_qkKb4aX-mFZm_h0A
      nombre: "El Hombre Invisible"
      poder: "Invisibilidad"
      vivo: true
  
```

Ahora le digo que está muerto y que además de invisible posee cisión de rayos X.

Firebase ID

Este campo es autogenerado

Nombre

Poder

Estado

Muerto

Angular is running in the development mode. Call enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.
▶ HeroeModel {vivo: true, nombre: "El hombre invisible", poder: "Invisibilidad", id: "-LxDIm1s7fK-l3FGWfQ"}
▶ {name: "-LxDJG0qYs4DUFKT_9eN"}
▶

Si le damos aceptar en la consola:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.  
[WDS] Live Reloading enabled.  
▶ HeroeModel {vivo: true, nombre: "El Hombre Inviible", poder: "Invisiibilidad", id: "-Lx_qkKb4aX-mFZm_h0A"}  
▶ {nombre: "El Hombre Inviible", poder: "Invisibilidad", vivo: true}  
▶ {nombre: "Visión Ratos X", poder: "Invisibilidad", vivo: false}
```

La base de datos queda:

```
🔗 https://appfirebasecrud-bdeb7.firebaseio.com/
```

```
appfirebasecrud-bdeb7  
└── heroes  
    └── -Lx_qkKb4aX-mFZm_h0A  
        ├── nombre: "Visión Ratos X"  
        ├── poder: "Invisibilidad"  
        └── vivo: false
```

8.- Información del estado de las peticiones.

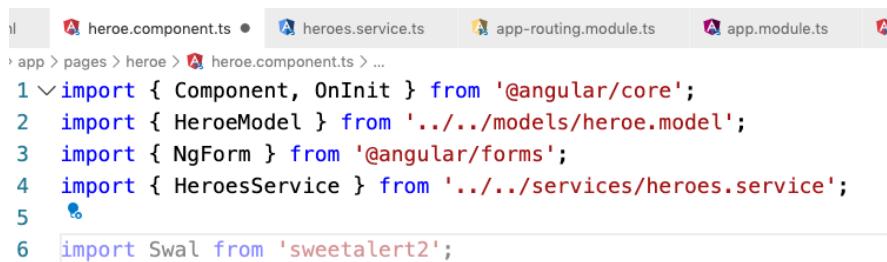
De alguna forma, debemos dar información al usuario del estado de sus peticiones. Hay que avisar al usuario si ha creado un superhéroe o lo ha actualizado. Emplearemos Sweet Alert.

```
iMac-de-Jose:17-angularCRUDfirebase jsersan$ npm install sweetalert
npm WARN karma-jasmine-html-reporter@1.5.1 requires a peer of jasm
f.

+ sweetalert2@9.5.4
added 1 package from 6 contributors and audited 18836 packages in
22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Lo importo en el componente donde lo quiero utilizar **heroe.component.ts**:



```
hero.component.ts
1 import { Component, OnInit } from '@angular/core';
2 import { HeroeModel } from '../../../../../models/heroe.model';
3 import { NgForm } from '@angular/forms';
4 import { HeroesService } from '../../../../../services/heroes.service';
5
6 import Swal from 'sweetalert2';
```

Lo voy a invocar antes de disparar la petición y cuando se resuelva la misma. En **heroe.component.ts**:

```
21 guardar( form: NgForm ) {
22
23   if ( form.invalid ) {
24     console.log('Formulario no válido');
25     return;
26   }
27
28   // Mostramos un mensaje
29   Swal.fire({
30     title: 'Espere',
31     text: 'Guardando información',
32     icon: 'info',
33     allowOutsideClick: false
34   });
35
36   // Mostramos un loading
37   Swal.showLoading();
--
```

Guardamos los cambios e insertamos a Nick Furia:

Firebase ID

Este campo es autogenerado

Nombre

Poder

Estado

😊 Vivo

💾 Guardar

El resultado:



Para poder cerrar el *Sweet Alert* nos creamos una variable petición de la siguiente forma y modifco la invocación de las funciones en **heroе.component.ts**:

```

37  // Mostramos un loading
38  Swal.showLoading();
39
40  let peticion: Observable<any>;
41
42  if ( this.heroe.id ) {
43    peticion = this.heroesService.actualizarHeroe(this.heroe);
44  } else {
45    peticion = this.heroesService.crearHeroe(this.heroe);
46  }
47

```

Independientemente de lo que haga, lo voy a controlar a través de la petición. Como todo pasar por referencia se actualiza el id. Todo junto con la petición en **heroе.component.ts**:

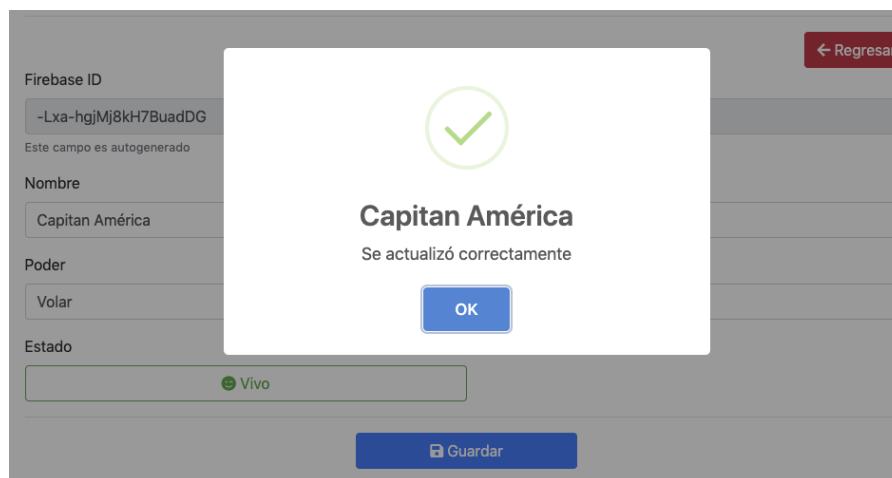
```

40  let peticion: Observable<any>;
41
42  if ( this.heroe.id ) {
43    peticion = this.heroesService.actualizarHeroe(this.heroe);
44  } else {
45    peticion = this.heroesService.crearHeroe(this.heroe);
46  }
47
48  // En cuanto se resuelva la respuesta mostraré el mensaje al usuario
49  peticion.subscribe( resp => {
50    Swal.fire({
51      title: this.heroe.nombre,
52      text: 'Se actualizó correctamente',
53      icon: 'success'
54    });
55  });
56}

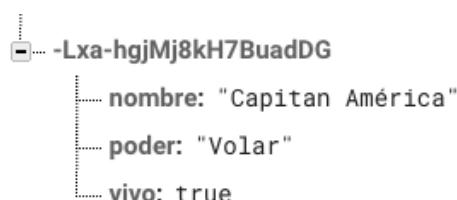
```

Probamos:

1. Creamos Capitán América:



En la base de datos:



Si actualizamos la Base de Datos:

Firebase ID

Este campo es autogenerado

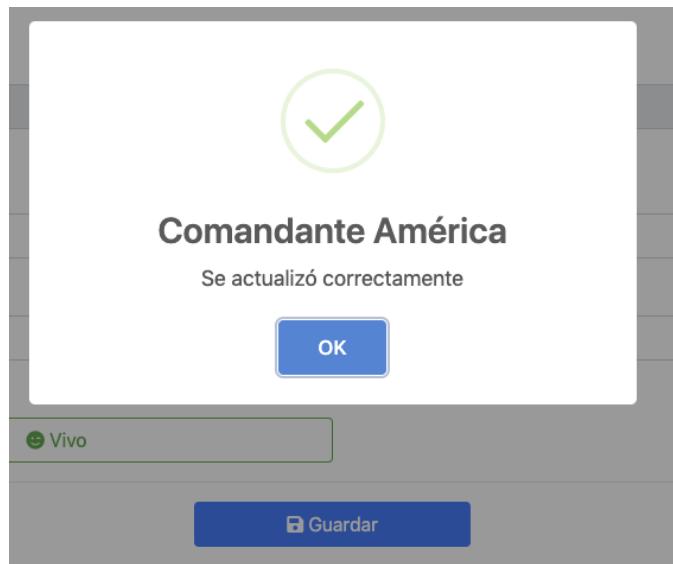
Nombre

Poder

Estado

Vivo

Le damos a aceptar:



En la Base de Datos:

```
  ...
  ... vivo: true
-Lxa-hgjMj8kH7BuadDG
  ...
  ... nombre: "Comandante América"
  ...
  ... poder: "Sacudir con el escudo"
  ...
  ... vivo: true
```

9.- HTTP GET – Obtener un listado de todos los héroes.

En el servicio **heroes.service.ts** nos creamos un nuevo método `getHeroes()`

```
40  getHeroes() {
41
42    return this.http.get(` ${this.URL}/heroes.json`);
43
44 }
```

Esto apunta en fireBase a:

<https://appfirebasecrud-bdeb7.firebaseio.com/heroes>

[appfirebasecrud-bdeb7](#) > heroes

```
heroes
└── -Lx_qkKb4aX-mFZm_h0A
    ├── nombre: "Visión Ratos X"
    ├── poder: "Invisibilidad"
    └── vivo: false
```

Para probarlo en **heroes.component.ts**:

```
10 constructor( private heroesService: HeroesService) { }
11
12 ngOnInit() {
13
14   this.heroesService.getHeroes()
15     .subscribe( resp => {
16       console.log(resp);
17     });
18
19 }
```

Mostramos la respuesta en la consola:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.
▼ {-Lx_qkKb4aX-mFZm_h0A: {}, -Lx_xcoK1bSAkPbKTGNz: {...}, -Lxa-hgjMj8kH7BuadDG: {...}} ⓘ
  ▼ -Lx_qkKb4aX-mFZm_h0A:
    nombre: "Visión Ratos X"
    poder: "Invisibilidad"
    vivo: false
    ► __proto__: Object
  ► -Lx_xcoK1bSAkPbKTGNz: {nombre: "Nick Furia", poder: "Tener un ojo", vivo: true}
  ► -Lxa-hgjMj8kH7BuadDG: {nombre: "Comandante América", poder: "Sacudir con el escudo", vivo: true}
  ► __proto__: Object
```

Ahora, debemos aprovechar esta información, pero el objeto no lo puedo mandar así como está porque el `ngFor` así no trabaja. El servicio `heroes.service.ts` me debe regresar un arreglo o una colección de héroes:

```

40  getHeroes() {
41
42    return this.http.get(` ${ this.URL }/heroes.json `)
43    .pipe(
44      map( resp => {
45
46        })
47      );
48  }

```

Dentro del `map` me creo un arreglo privado de héroes:

```

42  return this.http.get(` ${ this.URL }/heroes.json `)
43  .pipe(
44    map( resp => {
45
46      })
47    );
48  }
49
50  private crearArreglo( heroesObj: object ) {
51
52    return 'Hola Mundo';
53  }
54

```

El `map` va a regresar lo que retorne la función `crearArreglo`:

```

40  getHeroes() {
41
42    return this.http.get(` ${ this.URL }/heroes.json `)
43    .pipe(
44      map( resp => this.crearArreglo( resp ) )
45    );
46  }
47
48  private crearArreglo( heroesObj: object ) {
49
50    return 'Hola Mundo';
51  }
52

```

Probamos:

```

Angular is running in the development mode. Call enableProdMode() to enable the core.js:38781
production mode.
[WDS] Live Reloading enabled.                                         client:52
Hola Mundo                                                       heroes.component.ts:16

```

El map nos devuelve lo que ha recibido: 'Hola mundo'. Teniendo en cuenta que:

```
map(resp => this.crearArreglo(resp))
```

Es igual a:

```
map(this.crearArreglo)
```

En crearArreglo debemos devolver transformado el objeto heroesObj:

```
48  private crearArreglo( heroesObj: object ) {
49
50    const heroes: HeroeModel[] = [];
51    console.log( heroesObj );
52
53    return 'Hola Mundo';
54
55 }
```

La salida:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.
▼ { -Lx_qkKb4aX-mFZm_h0A: { ... }, -Lx_xcoK1bSAkPbKTGNz: { ... }, -Lxa-hgjMj8kH7BuadDG: { ... } } ⓘ
  ► -Lx_qkKb4aX-mFZm_h0A: { nombre: "Visión Ratos X", poder: "Invisibilidad", vivo: false }
  ► -Lx_xcoK1bSAkPbKTGNz: { nombre: "Nick Furia", poder: "Tener un ojo", vivo: true }
  ► -Lxa-hgjMj8kH7BuadDG: { nombre: "Comandante América", poder: "Sacudir con el escudo", vivo: true }
  ► __proto__: Object
```

Debemos retornar un array vacío si el objeto no existe:

```
48  private crearArreglo( heroesObj: object ) {
49
50    const heroes: HeroeModel[] = [];
51    console.log( heroesObj );
52
53    if ( heroesObj === null ) { return []; }
54
55    return 'Hola Mundo';
56
57 }
```

Necesitamos obtener la información de cada superhéroe:

```
Object.keys( heroesObj ).forEach (key => {
  const heroe: HeroeModel = heroesObj[key];
});
```

Debemos añadirlos al arreglo heroes y retornar el mismo:

```

48  private crearArreglo( heroesObj: object ) {
49
50    const heroes: HeroeModel[] = [];
51    console.log( heroesObj );
52
53    if ( heroesObj === null ) { return []; }
54
55    Object.keys( heroesObj ).forEach( key => {
56
57      const hero: HeroeModel = heroesObj[ key ];
58      hero.id = key;
59      heroes.push( hero );
60
61    });
62
63    return heroes;
64
65  }

```

Actualizamos y visualizamos la consola:

```

heroes.component.ts:16
▼ (3) [{}]
  ▶ 0: {nombre: "Visión Ratos X", poder: "Invisibilidad", vivo: false, id: "-Lx_qKKb4aX-mFZm_h0A"}
  ▶ 1: {nombre: "Nick Furia", poder: "Tener un ojo", vivo: true, id: "-Lx_xcOK1bSAkPbKTGNz"}
  ▶ 2: {nombre: "Comandante América", poder: "Sacudir con el escudo", vivo: true, id: "-Lxa-hgjMj8kH7BuadDG"}
  length: 3

```

Ahora debemos llevar a cabo la manipulación del HTML. En el **heroes.component.ts**:

```

3  import { HeroeModel } from '../../../../../models/heroe.model';
4
5  @Component({
6    selector: 'app-heroes',
7    templateUrl: './heroes.component.html'
8  })
9  export class HeroesComponent implements OnInit {
10
11  heroes: HeroeModel[] = [];
12
13  constructor( private heroesService: HeroesService ) { }
14
15  ngOnInit() {
16
17    this.heroesService.getHeroes()
18      .subscribe( resp => {
19        console.log( resp );
20        this.heroes = resp;
21      });
22  }
23}

```

Ya tenemos el arreglo de heroes, así que vamos a utilizar para llenar la tabla HTML:

```

13 <table class="table mt-3">
14   <thead class="thead-dark">
15     <tr>
16       <th scope="col">#</th>
17       <th scope="col">Nombre</th>
18       <th scope="col">Poder</th>
19       <th scope="col">Vivo</th>
20       <th scope="col">Tools</th>
21     </tr>
22   </thead>
23   <tbody>
24     <tr *ngFor="let heroe of heroes">
25       <th scope="row">{{ i+1 }}</th>
26       <td>{{ heroe.nombre }}</td>
27       <td>{{ heroe.poder }}</td>
28       <td>{{ heroe.vivo }}</td>
29     </tr>
30   </tbody>
31 </table>

```

Guardo los cambios:

Listado de Héroes



#	Nombre	Poder	Vivo	Tools
NaN	Visión Ratos X	Invisibilidad	false	
NaN	Nick Furia	Tener un ojo	true	
NaN	Comandante América	Sacudir con el escudo	true	

Para quitar el NAN de #:

```

<tr *ngFor="let heroe of heroes; let i = index">
  <th scope="row">{{ i+1 }}</th>
  <td>{{ heroe.nombre }}</td>
  <td>{{ heroe.poder }}</td>
  <td>

```

Con el estado Vivo o no, haremos una modificación estética.

```
<tr *ngFor="let heroe of heroes; let i = index">
  <th scope="row">{{ i+1 }}</th>
  <td>{{ heroe.nombre }}</td>
  <td>{{ heroe.poder }}</td>
  <td>
    <label class="badge badge-success">Vivo</label>
    <label class="badge badge-danger">Muerto</label>
  </td>
</tr>
```

El resultado es:

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Vivo Muerto	
2	Nick Furia	Tener un ojo	Vivo Muerto	
3	Comandante América	Sacudir con el escudo	Vivo Muerto	

Lo resolvemos con un ngIf:

```
<tr *ngFor="let heroe of heroes">
  <th scope="row">{{ i+1 }}</th>
  <td>{{ heroe.nombre }}</td>
  <td>{{ heroe.poder }}</td>
  <td>
    <label *ngIf="heroe.vivo" class="badge badge-success">Vivo</label>
    <label *ngIf="!heroe.vivo" class="badge badge-danger">Muerto</label>
  </td>
</tr>
```

Resultado:

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	
2	Nick Furia	Tener un ojo	Vivo	
3	Comandante América	Sacudir con el escudo	Vivo	

10.- HTTP GET – Obtener un nodo específico.

Es el momento de editar un héroe específico. Cuando hagamos click en cualquier fila, me lleve a la ventana de edición que es la misma que la de nuevo héroe pero con la variación de que le tengo que pasar el id del héroe.

En `heroes.component.html`:

```
<td>
  <label *ngIf="heroe.vivo" class="badge badge-success">Vivo</label>
  <label *ngIf="!heroe.vivo" class="badge badge-danger">Muerto</label>
</td>
<td>
  <button class="btn btn-info mr-1">
    <i class="fa fa-pen"></i>
  </button>
</td>
```

La clase `fa-pen` es como un lápiz. Grabamos los cambios:

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	
2	Nick Furia	Tener un ojo	Vivo	
3	Comandante América	Sacudir con el escudo	Vivo	

Copiamos el código para hacer el borrado (trash).

```
<td>
  <button class="btn btn-info mr-1">
    <i class="fa fa-pen"></i>
  </button>

  <button class="btn btn-danger">
    <i class="fa fa-trash"></i>
  </button>
</td>
```

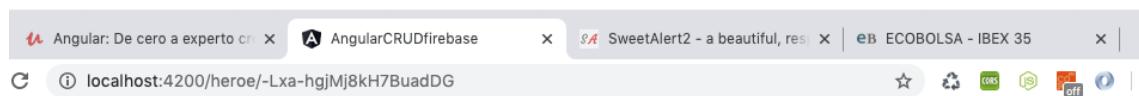
El resultado:

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	
2	Nick Furia	Tener un ojo	Vivo	
3	Comandante América	Sacudir con el escudo	Vivo	

Le añadimos funcionalidad al botón que no es más que un RouterLink:

```
<button class="btn btn-info mr-1"
        [routerLink]=["'/heroe',heroe.id"]>
    <i class="fa fa-pen"></i>
</button>
```

La ruta va entre corchetes porque la ruta no es estática. Va a variar según la fila que seleccione. Guardo los cambios y si pulso en Capitán América:



Héroe Nombre del Héroe

[← Regresar](#)

Firebase ID

Este campo es autogenerado

Nombre

Poder

Estado

😊 Vivo

En la URL tenemos el id. Con este id vamos a trabajar. En heroes.service.ts creamos un servicio que nos proporcione un héroe a partir de una id. Si tomamos la url del Capitán América.



Así en **heroes.service.ts**:

```
40  getHeroe(id: string ) {
41      return this.http.get(` ${ this.URL}/heroes/${ id }.json` );
42  }
```

En **hero.component.ts** debo hacer una pequeña modificación en el `ngOnInit()`. Como voy a obtener el id por la url debo inyectar un servicio que me permita obtener la ruta y realizar la navegación:

```
18  constructor( private heroesService: HeroesService,
19    private route: ActivatedRoute ) { }
```

La función `ngOnInit()`

```
21  ngOnInit() {
22
23    // Obtengo el id de la ruta
24    const id = this.route.snapshot.paramMap.get('id');
25
26    // Con el id llamo al servicio
27
28    console.log(id);
29
30 }
```

Probamos:

The screenshot shows a browser window at `localhost:4200/hero/-Lx_qkKb4aX-mFZm_h0A`. The page displays a form for editing a hero with fields for Nombre, Poder, and Estado. The Estado field has a green border and contains the value "Vivo". Below the form is a large blue button labeled "Guardar". At the bottom of the screen, the developer tools' Console tab is open, showing the following log output:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode. core.js:38781
-Lxa-hgjMj8kh7BuadDG hero.component.ts:28
[WDS] Live Reloading enabled. client:52
▶ -Lx_qkKb4aX-mFZm_h0A: {...}, -Lx_xcoK1bSAkPbKTGNz: {...}, -Lxa-hgjMj8kh7BuadDG: {...} heroes.service.ts:55
▶ (3) {...}, {...}, {...} heroes.component.ts:19
-Lx_qkKb4aX-mFZm_h0A hero.component.ts:28
```

Si le doy a nuevo, en la url ya no está el id:

The screenshot shows a browser window with the URL `localhost:4200/heroe/nuevo`. The page displays a form for adding a new hero. The 'Nombre' field contains 'Nombre del héroe'. The 'Poder' field contains 'Poder del héroe'. The 'Estado' dropdown is set to 'Vivo'. A blue 'Guardar' button is at the bottom. Below the form, the browser's developer tools Console tab is open, displaying logs from the application. The logs show the creation of a new hero with the ID `-Lxa-hgjMj8kh7BuadDG`, which corresponds to the 'nuevo' value in the URL.

En base a esto, en el `ngOnInit()`:

```

21  ngOnInit() {
22
23    // Obtengo el id de la ruta
24    const id = this.route.snapshot.paramMap.get('id');
25
26    // Con el id llamo al servicio
27
28    if ( id !== 'nuevo' ) {
29      // Me voy a firebase para recuperar toda la información del héroe
30
31      this.heroesService.getHeroe(id)
32        .subscribe( resp =>{
33          console.log(resp);
34        });
35
36    }
37
38  }

```

Guardamos los cambios:

```
▼ {nombre: "Comandante América", poder: "Sacudir con el escudo", vivo: true} ⓘ heroе.component.ts:33
  nombre: "Comandante América"
  poder: "Sacudir con el escudo"
  vivo: true
▶ __proto__: Object
```

Para ello:

```
if ( id !== 'nuevo' ) {
  // Me voy a firebase para recuperar toda la información del héroe
  this.heroesService.getHeroe(id)
    .subscribe( resp: HeroeModel ) => {
      this.heroe = resp;
      this.heroe.id = id;
    });
}
```

El objeto está casi completo:

```
▼ (3) [{...}, {...}, {...}] ⓘ heroes.component.ts:19
  ▼ 0:
    nombre: "Visión Ratos X"
    poder: "Invisibilidad"
    vivo: false
    id: "-Lx_qkKb4aX-mFZm_h0A"
  ▶ __proto__: Object
  ▼ 1:
    nombre: "Nick Furia"
    poder: "Tener un ojo"
    vivo: true
    id: "-Lx_xcoK1bSAkPbKTGNz"
  ▶ __proto__: Object
  ▼ 2:
    nombre: "Comandante América"
    poder: "Sacudir con el escudo"
    vivo: true
    id: "-Lxa-hgjMj8kH7BuadDG"
  ▶ __proto__: Object
```

Ya me carga toda la información:

Héroe Nombre del Héroe

← Regresar

Firebase ID

Este campo es autogenerado

Nombre

Poder

Estado

Realizo las modificaciones:

Héroe Nombre del Héroe

[← Regresar](#)

Firebase ID
-Lxa-hgjMj8kH7BuadDG

Este campo es autogenerado

Nombre
Comandante América

Poder
Volar encima del escudo

Estado
 Muerto

[Guardar](#)

Le doy a guardar:

[← Regresar](#)

Firebase ID
-Lxa-hgjMj8kH7BuadDG

Este campo es autogenerado

Nombre
Comandante América

Poder
Volar encima del escudo

Estado
 Muerto

✓

Comandante América

Se actualizó correctamente

[OK](#)

[Guardar](#)

Regresamos:

Listado de Héroes

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	 
2	Nick Furia	Tener un ojo	Vivo	 
3	Comandante América	Volar encima del escudo	Muerto	 

Lo compruebo en FireBase:

[appfirebasecrud-bdeb7](#) > heroes > [-Lxa-hgjMj8kH7BuadDG](#)

-Lxa-hgjMj8kH7BuadDG

```
  nombre: "Comandante América"
  poder: "Volar encima del escudo"
  vivo: false
```

11.- HTTP DELETE – Eliminar registros.

Lo haremos sencillo, que sólo sea llamar a un servicio y que sea una petición DELETE. Nos creamos en el servicio el método de borrarHeroe(id) en **heroes.service.ts**.

```
40  borrarHeroe( id: string ) {
41
42      // Mismo url que para obtener un sólo héroe
43      return this.http.delete(`${ this.URL }/heroes/${ id }.json`);
44 }
```

Ahora en **heroes.component.ts**:

```
24     borrarHeroe( hero: HeroeModel ) {  
25         this.heroesService.borrarHeroe( hero.id).subscribe();  
26     }  
27 }  
28 }
```

Los cambios no se van a ver inmediatamente. Debemos llamar al método `borrarHeroe` en el HTML `heroes.component.html`:

```
37         <button class="btn btn-danger"  
38             (click)="borrarHeroe(heroe)">  
39             <i class="fa fa-trash"></i>
```

Probamos:

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	 
2	Nick Furia	Tener un ojo	Vivo	 
3	Comandante América	Volar encima del escudo	Muerto	 

Borramos a Nick Furia y parece que nada ha pasado. Vamos a FireBase:

appfirebasecrud-bdeb7 → hero

```
heroes
  -Lx_qkKb4aX-mFZm_h0A
    nombre: "Visión Ratos X"
    poder: "Invisibilidad"
    vivo: false
  -Lxa-hgjMj8kH7BuadDG
    nombre: "Comandante América"
    poder: "Volar encima del escudo"
    vivo: false
```

Si actualizamos la página:

#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	 
2	Comandante América	Volar encima del escudo	Muerto	 

Para borrar el registro del arreglo, me hace falta saber la posición i (index):

```
<button class="btn btn-danger" (click)="borrarHeroe(heroe, i)">
    <i class="fa fa-trash"></i>
</button>
```

Para tener en cuenta este segundo parámetro en heroes.component.ts :

```
24     borrarHeroe( heroe: HeroeModel, i: number ) {
25
26         this.heroes.splice(i, 1);
27         this.heroesService.borrarHeroe( heroe.id).subscribe();
28
29     }
```

Realizo una eliminación. Por ejemplo, Comandante América. Instantáneamente se elimina:

				
#	Nombre	Poder	Vivo	Tools
1	Visión Ratos X	Invisibilidad	Muerto	 

En la Base de Datos:

[appfirebasecrud-bdeb7](#) > [heroes](#)

```
heroes
  -Lx_qkKb4aX-mFZm_h0A
    nombre: "Visión Ratos X"
    poder: "Invisibilidad"
    vivo: false
```

Debiéramos preguntarle al usuario si de verdad lo quiere borrar:

```

24
25     borrarHeroe( heroe: HeroeModel, i: number ) {
26
27         Swal.
28             fire
29                 this.getActions
30                 this.getCancelButton
31                 this.getCloseButton
32                 this.getConfirmButton
33                 this.getContent
34                 this.getFocusableElements
35                 this.getFooter
36                 this.getHtmlContainer
37                 this.setIcon
38                 this.getIcons
39                 this.getImage

```

function Swal.fire(title?: string, message?: string, icon?: SweetAlertIcon): Promise<SweetAlertResult> (+1 overload)
Function to display a simple SweetAlert2 modal.
ex. import Swal from 'sweetalert2';
Swal.fire('The Internet?', 'That thing is still around?', 'question');

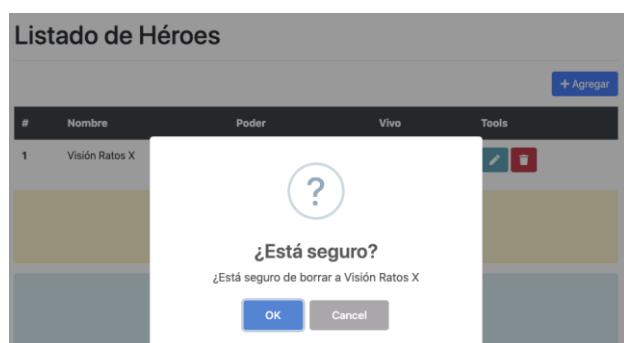
La función completa:

```

25     borrarHeroe( heroe: HeroeModel, i: number ) {
26
27         Swal.fire({
28             title: '¿Está seguro?',
29             text: `¿Está seguro de borrar a ${heroe.nombre}`,
30             icon: 'question',
31             showConfirmButton: true,
32             showCancelButton: true
33         }).then( resp => {
34             if (resp.value) {
35                 this.heroes.splice(i, 1);
36                 this.heroesService.borrarHeroe( heroe.id).subscribe();
37             }
38         });
39     }

```

Probamos:



Si le damos a OK:



12.- Aspectos estéticos.

Hagamos funcionar las dos cajas de alertar de la parte inferior. Necesito saber en qué momento estoy cargando información y en qué momento dejé de cargarla. Creo una nueva variable en `heroes.component.ts` que se llamará *cargando*.

```
-- 
11  export class HeroesComponent implements OnInit {
12
13  | heroes: HeroeModel[] = [];
14  | cargando = false;
```

Cuando esta variable esté a true, voy a mostrar este *loading*. Este cargando se va a disparar en el `ngOnInit()`. Antes de llamar a `getHeroes()` del servicio lo voy a llamar y tendré que cancelarla en el subscribe. Así, el `ngOnInit()` de `heroes.component.ts`:

```
18  ngOnInit() {
19
20    this.cargando = true;
21    this.heroesService.getHeroes()
22      .subscribe( resp => {
23        console.log(resp);
24        this.heroes = resp;
25        this.cargando = false;
26      });
27 }
```

Después de esto, en el HTML, solo se mostrará si está cargando:

```
52 <div *ngIf="cargando"
53   class="alert alert-info text-center mt-3">
54   <h4 class="alert-heading">Cargando</h4>
55   <p>
56     <i class="fa fa-sync-alt fa-spin fa-2x"></i>
57   </p>
58   <p class="mb-0">
59     Espere por favor
60   </p>
61 </div>
```

Grabamos los cambios y el div desaparece en cuanto obtiene la respuesta:

Listado de Héroes

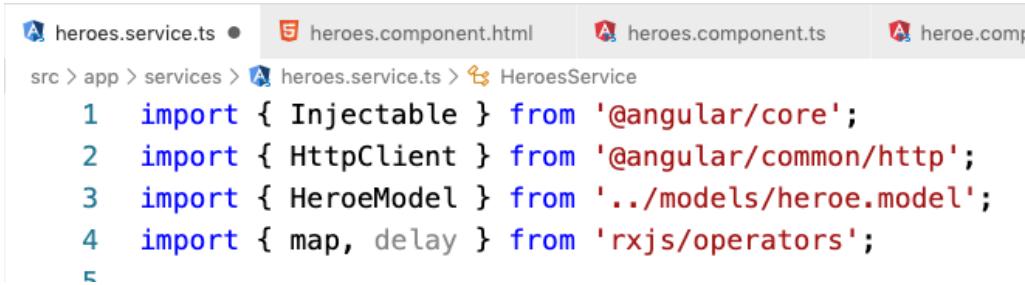


The screenshot shows a table with columns: #, Nombre, Poder, Vivo, and Tools. Two rows are present: Ironman (Dinero, Vivo) and Spiderman (Intuición, Muerto). Each row has edit and delete icons in the Tools column. Below the table, a yellow box displays the message "No hay registros!" (No records!).

#	Nombre	Poder	Vivo	Tools
1	Ironman	Dinero	Vivo	 
2	Spiderman	Intuición	Muerto	 

No hay registros!

Lo hace super rápido, para poder ver algo, utilizamos la función delay del operador rxjs en el servicio:



```

1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { HeroeModel } from '../models/heroe.model';
4 import { map, delay } from 'rxjs/operators';
5

```

Lo utilizamos en `getHeroes()` del servicio **heroes.service.ts**:

```

50  getHeroes() {
51
52      return this.http.get(` ${ this.URL }/heroes.json`)
53          .pipe(
54              map(this.crearArreglo),
55              delay(1500)
56          );
57

```

Suavizamos un poco la carga del div del mensaje:

```

52 <div *ngIf="cargando"
53 | class="alert alert-info text-center mt-3 animated fadeIn faster">
54 |     <h4 class="alert-heading">Cargando</h4>
55 |     <p>
56 |         <i class="fa fa-sync-alt fa-spin fa-2x"></i>
57 |     </p>
58 |     <p class="mb-0">
59 |         Espere por favor
60 |     </p>
61 </div>

```

Lo mismo haré con la tabla:

```

-->
13 <table class="table mt-3 animated fadeIn faster">
14     <thead class="thead-dark">
15         <tr>
16             <th scope="col">#</th>
17             <th scope="col">Nombre</th>
18             <th scope="col">Poder</th>
19             <th scope="col">Vivo</th>
20             <th scope="col">Tools</th>

```

La tabla debe estar oculta hasta que se cargue:

```
-->
13 <table *ngIf="!cargando" class="table mt-3 animated fadeIn faster">
14   <thead class="thead-dark">
15     <tr>
16       <th scope="col">#</th>
17       <th scope="col">Nombre</th>
18       <th scope="col">Poder</th>
19       <th scope="col">Vivo</th>
20       <th scope="col">Tools</th>
21     </tr>
```

Si actualizamos aparece el div de cargando y luego desaparece:

#	Nombre	Poder	Vivo	Tools
1	Ironman	Dinero	Vivo	
2	Spiderman	Intuición	Muerto	

No hay registros
 !

Ahora, debemos controlar el div de “No hay registros”. Aparecerá cuando no esté el div de “cargando” y el array de *heroes* recuperado sea cero.

```
45 <div *ngIf="!cargando && heroes.length === 0"
46   class="alert alert-warning text-center mt-3 animated fadeIn
47   faster">
48   <h4 class="alert-heading">No hay registros</h4>
49   <p>
50     <i class="fa fa-exclamation fa-2x"></i>
51   </p>
52 </div>
```

Si borramos todos los registros:

Listado de Héroes

#	Nombre	Poder	Vivo	Tools
---	--------	-------	------	-------

No hay registros
 !

En caso contrario no aparece:

Listado de Héroes

#	Nombre	Poder	Vivo	Tools
1	Ironman	Dinero	Muerto	 
2	Catwoman	Agilidad	Muerto	 
3	Hulk	Super Fuerza	Muerto	 