

# Angular 5 con API web - Operaciones CRUD

## (<http://www.dotnetmob.com/angular-5-tutorial/angular-5-with-web-api-crud-application/>)

Tutorial Angular 5 ([Http://Www.Dotnetmob.Com/Category/Angular-5-Tutorial/](http://www.Dotnetmob.Com/Category/Angular-5-Tutorial/)) |


5 Comentarios ([Http://Www.Dotnetmob.Com/Angular-5-Tutorial/Angular-5-With-Web-API-Crud-Application/#Comments](http://www.Dotnetmob.Com/Angular-5-Tutorial/Angular-5-With-Web-API-Crud-Application/#Comments))


(<http://www.dotnetmob.com/angular-5-tutorial/angular-5-with-web-api-crud-application/>)

**35**  
COMPARTE

 **Compartir** (<https://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.dotnetmob.com%2Fangular-5-tutorial%2Fangular-5-with-web-api-crud-application%2F>)

 **Pío** (<https://twitter.com/intent/tweet?text=Angular%205%20with%20Web%20API%20%E2%80%93%20CRUD%20Operations&url=http://www.dotnetmob.com/angular-5-tutorial/angular-5-with-web-api-crud-application/&via=DotnetMob>)

 **Linkedin** (<https://www.linkedin.com/shareArticle?trk=Angular+5+with+Web+API+%E2%80%93+CRUD+Operations&url=http%3A%2F%2Fwww.dotnetmob.com%2Fangular-5-tutorial%2Fangular-5-with-web-api-crud-application%2F>)

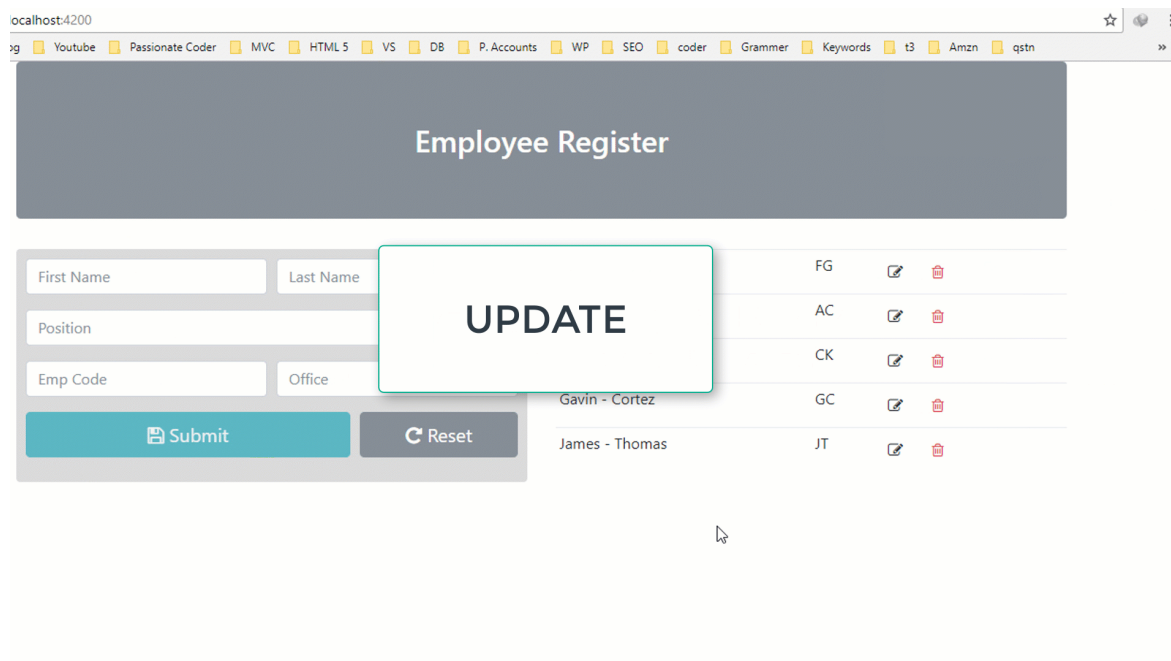
 **Google** (<https://plus.google.com/share?text=Angular+5+with+Web+API+%E2%80%93+CRUD+Operations&url=http%3A%2F%2Fwww.dotnetmob.com%2Fangular-5-tutorial%2Fangular-5-with-web-api-crud-application%2F>)

 **Suscribir**



En este artículo, crearemos una aplicación CRUD utilizando **Angular 5 con la API web**. La actualización y eliminación de Insertar operaciones CRUD se implementará dentro de una API web de Asp.net utilizando Entity Framework y luego se consumirá desde una aplicación Angular 5.

### Demostración de aplicación:

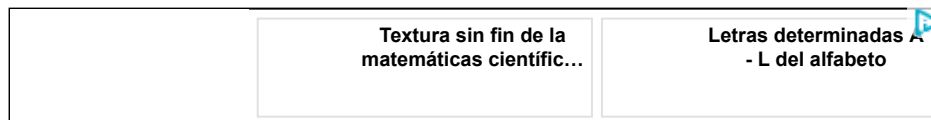


Las siguientes herramientas y módulos se utilizan para este proyecto:

- CLI angular
- Angular 5
- ngx-Toastr (paquete npm)
- Iconos Bootstrap y Font-Awesome
- VS Code y Visual Studio Editor

suponemos que ha instalado paquetes y software necesarios para el desarrollo angular de aplicaciones.

Descargue el código fuente del proyecto desde GitHub: Angular 5 con API web - Operaciones CRUD. (<https://github.com/DotnetMob/Angular-5-With-Web-API-CRUD-Application>)



## Crear base de datos SQL Server

Antes que nada, creemos una base de datos para trabajar. Uso Management Studio para crear y administrar bases de datos SQL.

Creó una base de datos con el nombre 'WebAPIDB', en esta aplicación trataremos los detalles de los empleados. por lo tanto , se crea una tabla de *empleados* utilizando el siguiente script SQL.

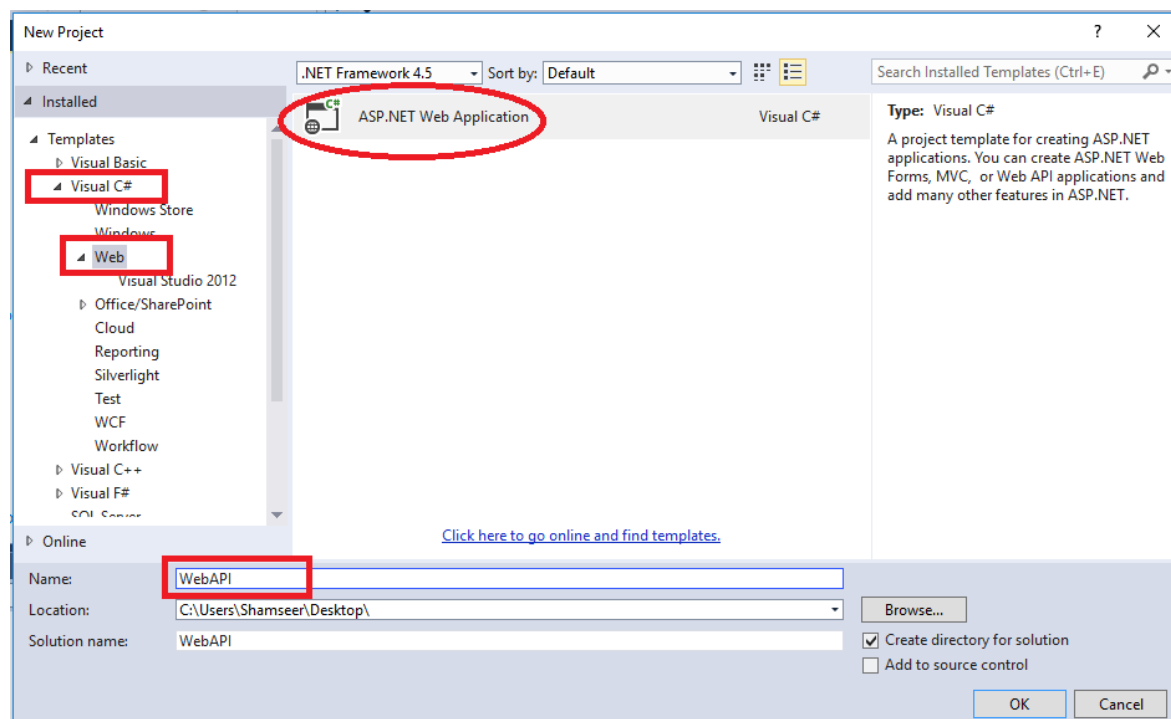
Transact-SQL	
1	CREATE TABLE [dbo].[Employee] (
2	[EmployeeID] [int] IDENTITY(1,1) NOT NULL primary key,
3	[FirstName] [varchar](50) NULL,
4	[LastName] [varchar](50) NULL,
5	[EmpCode] [varchar](50) NULL,
6	[Position] [varchar](50) NULL,
7	[Office] [varchar](50) NULL)

Aquí, la columna *EmployeeID* es la clave principal y la columna IDENTITY de la tabla. Debido a la especificación IDENTITY, no insertamos valores en la columna *EmployeeID* . SQL Server se encargará de eso. Comenzará desde 1 y se incrementará en 1 con la inserción de un nuevo registro.

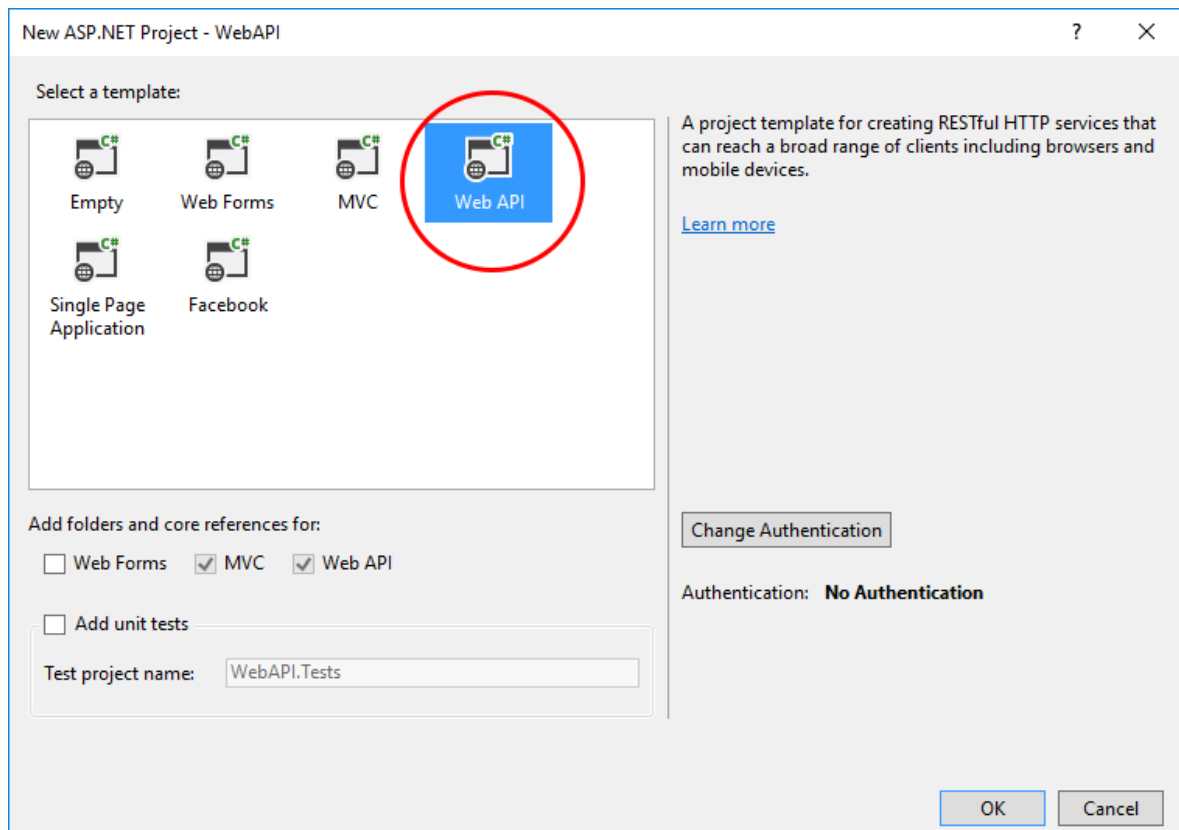
## Crear aplicación web API

La base de datos está lista, ahora vamos a crear un proyecto Asp.Net Web API.

Abra Visual Studio, vaya a *Archivo> Nuevo> Proyecto (Ctrl + Shift + N)*.



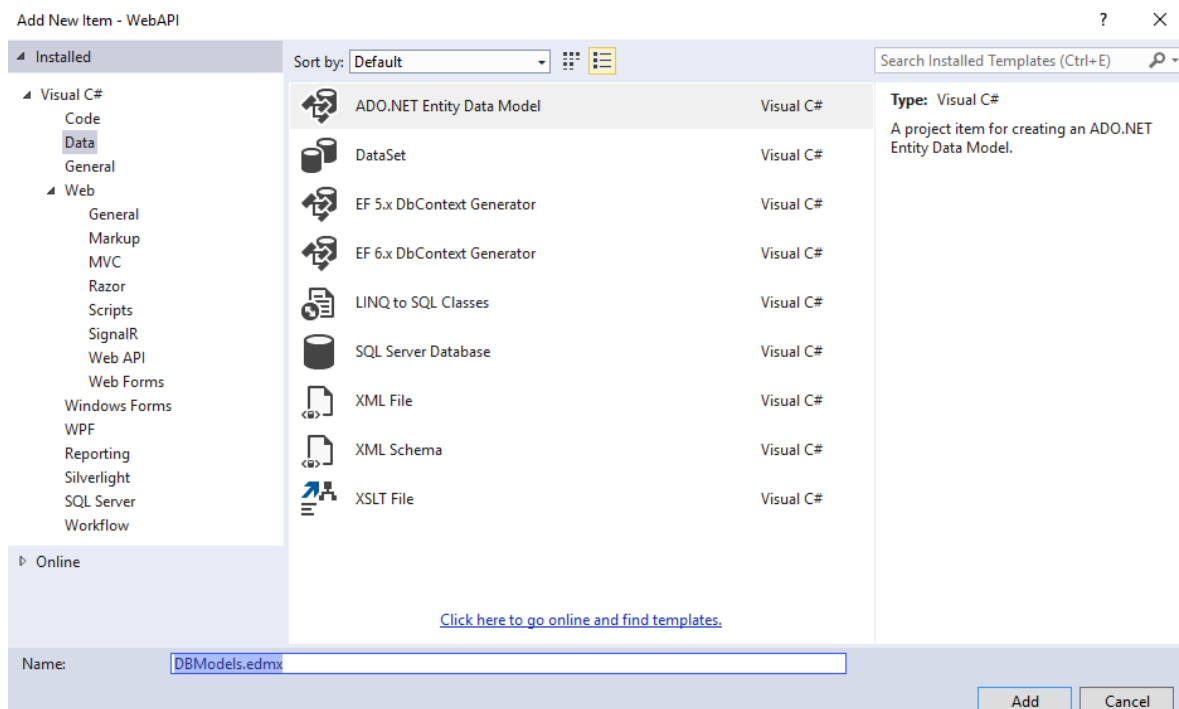
luego seleccione la plantilla de la API web.



Así que aquí hemos creado un nuevo proyecto de API web. Ahora agreguemos el Modelo de Entidad para el DB ' *WEPAPI*DB' dentro de la Carpeta de *Modelos* .


Haga clic derecho en la carpeta *Modelos* > Agregar> Nuevo elemento.

Denomine su modelo de entidad como *DBModels.edmx*.





Seleccione *Generar de la base de datos*.

Entity Data Model Wizard ✕

 Choose Model Contents

**What should the model contain?**

  
Generate from database

  
Empty model

Creates an entity model from a database. Object-layer code is generated from the model. This option also lets you specify the database connection, settings for the model, and database objects to include in the model.

< Previous Next > Finish Cancel

En la Ventana de *conexión de datos* , haga clic en *Nueva conexión*.



## Choose Your Data Connection

Which data connection should your application use to connect to the database?

desktop-ip79d51\sqli2012.WebAPIDB.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

- ☐ No, exclude sensitive data from the connection string. I will set it in my application code.
- ☐ Yes, include the sensitive data in the connection string.

Entity connection string:

```
metadata=res://*/Models.DBModels.csdl|res://*/Models.DBModels.ssdl|
res://*/Models.DBModels.msl;provider=System.Data.SqlClient;provider connection string="data
source=(local)\sqli2012;initial catalog=WebAPIDB;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save entity connection settings in Web.Config as:

DBModel

< Previous

Next >

Finish

Cancel

Proporcione detalles de la instancia de SQL Server y seleccione la base de datos.

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:  
Microsoft SQL Server (SqlClient) Change...

Server name:  
(local)\sql2012 Refresh

Log on to the server

☒ Use Windows Authentication  
☐ Use SQL Server Authentication

User name:   
Password:   
☐ Save my password

Connect to a database

☒ Select or enter a database name:  
WebAPIDB

☐ Attach a database file:  
 Browse...  
Logical name:


Advanced...

Test Connection OK Cancel

Como en la ventana anterior, *guardaremos la* cadena de conexión de DB en WebConfig como *DBModel*. Después de crear este Modelo de Entidad habrá una clase con este nombre (*DBModel*), crearemos un objeto de esta clase para interactuar con la base de datos.

Después de la ventana anterior, puede ver una ventana adicional de la siguiente manera, si tiene múltiples versiones de Entity Framework, luego seleccione una de ellas.

Entity Data Model Wizard ×

 Choose Your Version

**Which version of Entity Framework do you want to use?**

☐ Entity Framework 6.0

☒ Entity Framework 5.0

**i** Your project references an older version of Entity Framework. To use the latest version, exit this wizard and upgrade before performing this action.

< Previous

Next >

Finish

Cancel

luego seleccione las tablas que queremos agregar dentro del Modelo de Entidad.



Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ Tables
  - ☒ dbo
    - ☒ Employee
  - ☐ Views
  - ☐ Stored Procedures and Functions

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

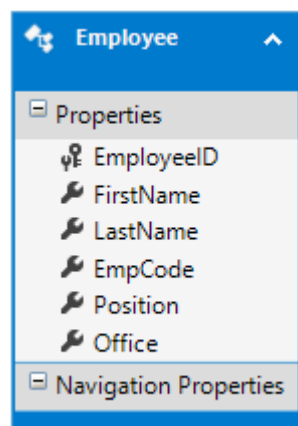
☐ Import selected stored procedures and functions into the entity model

Model Namespace:

WebAPIDBModel

< Previous   Next >   **Finish**   Cancel

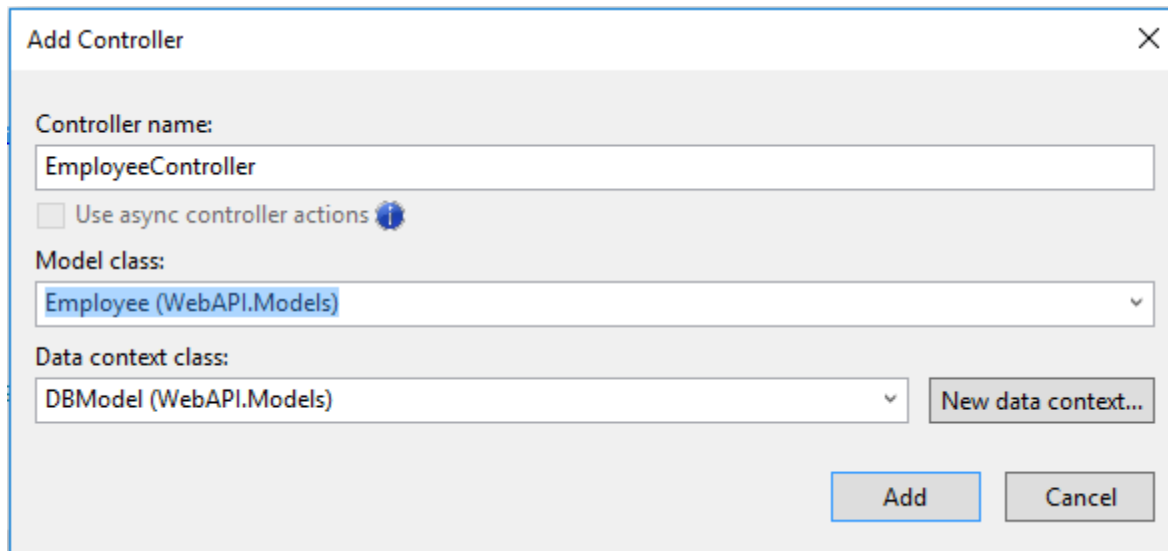
Haga clic en *Finalizar*. La *Representación Diagramática* del Modelo EF se ve así.



Dentro de este *DBModels.edmx*, se puede ver una *DBModels.Context.cs* archivo para *DBModel* clase. crearemos un objeto de esta clase para trabajar con la base de datos.

*DBModels.tt> Employee.cs* contiene *Employee* Class, que contiene propiedades correspondientes a SQL Table Columns, esta es nuestra clase de modelo.

Ahora agreguemos Employee Controller, antes de eso, no olvide *volver a generar* su solución. Para crear un *Controlador*, haga clic con el botón derecho en la carpeta de *controladores* y luego haga clic en *Agregar> Controlador ...*, luego seleccione *Controlador Web API 2 con acciones, utilizando Entity Framework Scaffolding Mechanism*. Luego se mostrará la siguiente ventana.



Crearé un *Empleador de* Controlador de Web API utilizando la *Clase de Empleado* del Modelo de Entidad. El controlador creado contiene los métodos web GET, POST, PUT y DELETE para las operaciones CRUD. LEER INSERTAR ACTUALIZAR Y ELIMINAR, respectivamente. Estos métodos web predeterminados contienen validaciones de modelo, no realizamos la validación de modelo en este proyecto de API web, la validación de formulario se puede realizar dentro de la aplicación angular 5, el controlador de *empleado* sin validación se ve así

```
EmployeeController.cs C#
1 public class EmployeeController : ApiController
2 {
3     private DBModel db = new DBModel();
4
5     // GET api/Employee
6     public IQueryable<Employee> GetEmployees()
7     {
8         return db.Employees;
9     }
10
11    // PUT api/Employee/5
12    public IHttpActionResult PutEmployee(int id, Employee employee)
13    {
14        db.Entry(employee).State = EntityState.Modified;
15        try
16        {
17            db.SaveChanges();
18        }
19        catch (DbUpdateConcurrencyException)
20        {
21            if (!EmployeeExists(id))
22            {

```

```

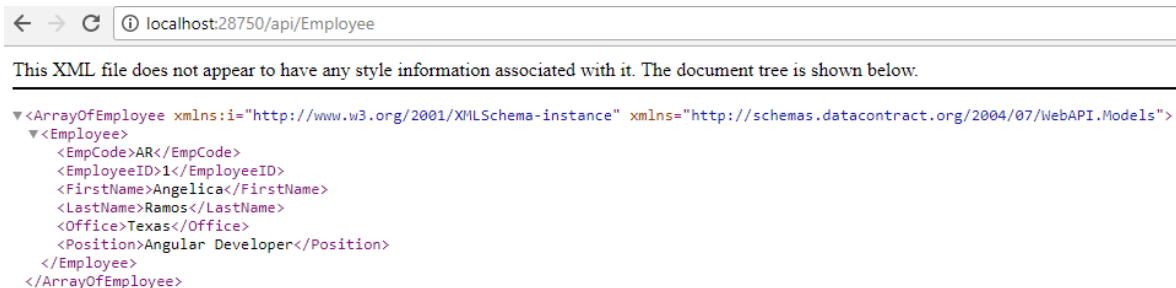
23         return NotFound();
24     }
25     else
26     {
27         throw;
28     }
29 }
30 return StatusCode(HttpStatusCode.NoContent);
31 }
32
33 // POST api/Employee
34 [ResponseType(typeof(Employee))]
35 public IHttpActionResult PostEmployee(Employee employee)
36 {
37     db.Employees.Add(employee);
38     db.SaveChanges();
39     return CreatedAtRoute("DefaultApi", new { id = employee.EmployeeID
40 }
41
42 // DELETE api/Employee/5
43 [ResponseType(typeof(Employee))]
44 public IHttpActionResult DeleteEmployee(int id)
45 {
46     Employee employee = db.Employees.Find(id);
47     if (employee == null)
48     {
49         return NotFound();
50     }
51     db.Employees.Remove(employee);
52     db.SaveChanges();
53     return Ok(employee);
54 }
55
56 protected override void Dispose(bool disposing)
57 {
58     if (disposing)
59     {
60         db.Dispose();
61     }
62     base.Dispose(disposing);
63 }
64
65 private bool EmployeeExists(int id)
66 {
67     return db.Employees.Count(e => e.EmployeeID == id) > 0;
68 }
69 }

```

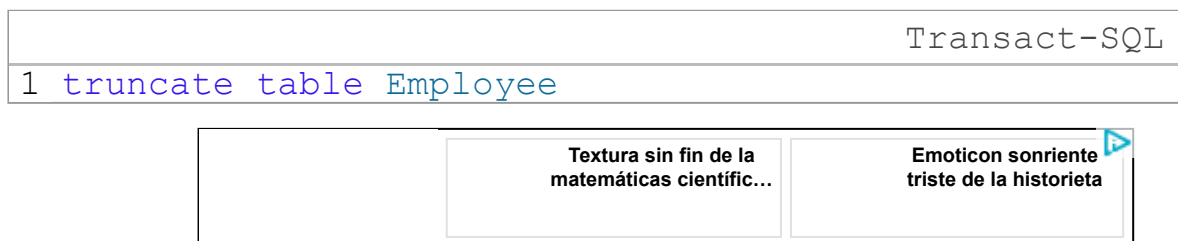
Todos estos métodos de API web se escriben utilizando DB First Approach en Entity Framework. Ahora revisemos el trabajo de este proyecto de API web, antes que nada permítame agregar un registro de empleado de prueba dentro de la tabla *Empleado* .

	Transact-SQL
1	<code>INSERT Employee ( [FirstName], [LastName], [EmpCode], [Posi</code>

Ahora ejecutemos nuestro proyecto de API web. luego navegue `/api/Employee` URL. Se llamará *GetEmployees* Método para recuperar la colección empleado de la tabla del servidor SQL *de los empleados*.



Aquí la URL base es localhost: 28750, necesitamos esta URL base para consumir esta API web de la aplicación Angular 5. Por lo tanto, nuestro Proyecto Web API funciona bien, finalmente déjenme truncar el registro de prueba de la tabla de empleados.



## Crear aplicación Angular 5

Utilizo Visual Studio Code Editor para Angular Project Development. Para crear una aplicación angular 5, puede usar el siguiente comando Angular CLI.

```
1 ng new AngularCRUD
```

Crearé la aplicación con el nombre *AngularCRUD* e instalará paquetes predeterminados de npm. Para ejecutar una aplicación angular, puede usar el siguiente comando.

```
1 ng serve --open
```

abrirá nuestra aplicación desde el número de puerto predeterminado 4200, es decir, <http://localhost:4200>.

## Agregar componentes, modelo y clase de servicio requeridos Angular 5 CRUD

Ahora tenemos que agregar 3 componentes. Para agregar un componente angular puedes hacer esto

```
1 //from root component
2 ng g c employees
```

Aquí creamos el componente de *empleados*, los dos componentes restantes serán componentes secundarios para este componente de *empleados*. El siguiente comando crea los componentes secundarios *de listas de empleados* y *empleados*.

```

1 //switch to parent component directory
2 cd src\app\employees
3 //create child components
4 ng g c employee
5 ng g c employee-list

```

Abra el archivo *appmodule.ts*, asegúrese de que los componentes recién agregados se agreguen a la matriz de *declaraciones*.

```

/src/app/app.module.ts
1 ...
2 import { EmployeesComponent } from '../employees/employees.component';
3 import { EmployeeComponent } from '../employees/employee/employee.component';
4 import { EmployeeListComponent } from '../employees/employee-list/employee-list.component';
5 @NgModule({
6   ...
7   declarations: [
8     ...
9     EmployeesComponent,
10    EmployeeComponent,
11    EmployeeListComponent
12  ],
13  ...

```

## Comencemos el diseño

Utilizaremos Bootstrap y Font-Awesome Icons para el diseño de aplicaciones. Entonces, primero agregue la referencia CDN para estas hojas de estilo dentro de *index.html*.

```

/src/index.html
1 <!-- Bootstrap -->
2 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css">
3 <!-- font-awesome -->
4 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">

```

Actualice *app.component.html* de la siguiente manera

```

/src/app/app.component.html
1 <div class="container">
2   <app-employees></app-employees>
3 </div>

```

Agregue lo siguiente al archivo *employees.component.html*.

```

/src/app/employees/employees.component.html
1 <div style="text-align:center">
2   <h2 class="jumbotron bg-secondary text-white">Employee Register</h2>
3 </div>
4 <div class="row">
5   <div class="col-md-6">
6     <app-employee></app-employee>
7   </div>
8   <div class="col-md-6">
9     <app-employee-list></app-employee-list>
10  </div>
11 </div>

```

Necesitamos modelos y servicios para diseñar los componentes secundarios restantes.

## Crear clases de servicio y modelo

Para crear estas clases, agreguemos una nueva carpeta *compartida*, dentro de la carpeta de *empleados* (/ src / app / employees /).

Ahora crea una clase de modelo de *empleado*

```
1 //switch to shared folder
2 cd src\app\employees\shared
3 //create employee model class
4 ng g class employee --type=model
5 //create employee service class
6 ng g s employee
```

en esta aplicación tratamos con detalles de los empleados como

- Nombre de pila
- Apellido
- Código Emp
- Posición
- Localización de la oficina

Entonces, tenemos que agregar propiedades correspondientes a los detalles de estos empleados dentro del archivo *employee.model.ts* .

```
/src/app/employees/shared/employee.model.ts
1 export class Employee {
2   EmployeeID : number;
3   FirstName:string;
4   LastName:string;
5   EmpCode:string;
6   Position:string;
7   Office:string;
8 }
```

La propiedad *EmployeeID* se usa para identificar cada registro de empleado individualmente dentro de la aplicación.

Dentro de la clase de servicio *Employee* , definimos funciones para cada operación CRUD en Angular 5 con Web API. Esto significa que consumimos métodos de Web API de esta clase de servicio.

```
/src/app/employees/shared/employee.service.ts
```

```

1 import { Injectable } from '@angular/core';
2 import { Http, Response, Headers, RequestOptions, RequestMethod } from '@angu
3 import { Observable } from 'rxjs/Observable';
4 import 'rxjs/add/operator/map';
5 import 'rxjs/add/operator/toPromise';
6
7 import {Employee} from './employee.model'
8
9 @Injectable()
10 export class EmployeeService {
11     selectedEmployee : Employee;
12     employeeList : Employee[];
13     constructor(private http : Http) { }
14
15     postEmployee(emp : Employee){
16         var body = JSON.stringify(emp);
17         var headerOptions = new Headers({'Content-Type':'application/json'});
18         var requestOptions = new RequestOptions({method : RequestMethod.Post, head
19         return this.http.post('http://localhost:28750/api/Employee',body,requestO
20     }
21
22     putEmployee(id, emp) {
23         var body = JSON.stringify(emp);
24         var headerOptions = new Headers({ 'Content-Type': 'application/json' });
25         var requestOptions = new RequestOptions({ method: RequestMethod.Put, head
26         return this.http.put('http://localhost:28750/api/Employee/' + id,
27             body,
28             requestOptions).map(res => res.json());
29     }
30
31     getEmployeeList() {
32         this.http.get('http://localhost:28750/api/Employee')
33             .map((data : Response) =>{
34                 return data.json() as Employee[];
35             }).toPromise().then(x => {
36                 this.employeeList = x;
37             })
38     }
39
40     deleteEmployee(id: number) {
41         return this.http.delete('http://localhost:28750/api/Employee/' + id).map(
42     }
43 }

```

En esta clase de servicio, hemos importado clases relacionadas con *http* y *rxjs*. La clase *http* se utiliza para consumir los métodos de la API web para insertar operaciones de actualización y eliminación.

## Pero hay un problema - CORS

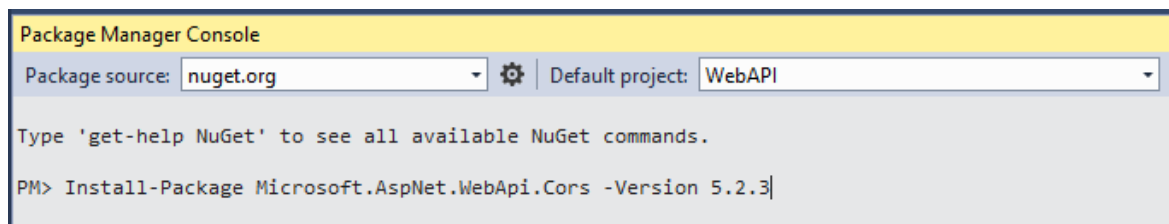
**CORS (Cross-Origin Resource Sharing)** : it is a mechanism to let a user agent (browser) gain permission to access selected resources from a server on a different origin (domain) than the site currently in use. **cross-origin HTTP request** occurs when it requests a resource from a different domain, protocol, or port than the one from which the current document originated.

In this application, our web API project will block request from angular 5 application, since they are cross-origin HTTP request(from different port numbers – 4200 and 28750). In-order to allow cross-origin HTTP request, we

have to configure Web API project for this localhost:4200 request. so let's look how we can do that.

Primero tenemos que instalar NuGet Package: WebApi.Cors. Vuelva a Visual Studio, seleccione su proyecto de API web desde el explorador de soluciones, luego vaya a *Herramientas> Administrador de paquetes de biblioteca> Consola de administrador de paquetes*. use el siguiente comando NuGet para instalar WebApi.Cors.

```
Install-Package Microsoft.AspNet.WebApi.Cors -Version
```



Ahora veamos cómo podemos usar este paquete. En orden permitir a petición de origen cruzado en el controlador de API Web *del Empleado* podemos hacer esto.

```
1 ...
2 using System.Web.Http.Cors;
3
4 [EnableCors(origins: "http://localhost:4200", headers: "*", methods: "*")]
5 public class EmployeeController : ApiController
6 {
7     .....
8 }
```

Aquí hemos otorgado permiso para la solicitud http desde 'http: // localhost: 4200', no es una buena idea agregar este atributo *EnableCors* para todos los controles API web si su proyecto es grande. En ese caso, haz esto.

Vaya a *App\_Start>* archivo *WebApiConfig.cs* . agregue las siguientes líneas de código

```
/App_Start/WebApiConfig.cs C#
1 ...
2 using System.Web.Http.Cors;
3
4 public static class WebApiConfig
5 {
6     public static void Register(HttpConfiguration config)
7     {
8         config.EnableCors(new EnableCorsAttribute("http://localhost:4200", head
9         ...
```

ahora el proyecto web API está listo para solicitud de origen cruzado desde nuestra aplicación angular 5.

intente navegar este URL */api/Employee* desde su proyecto de API web, algunos de ustedes pueden tener este problema.



**Server Error in '/' Application.**

Could not load file or assembly 'System.Web.Http, Version=5.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The located assembly's manifest definition does not match the assembly reference. (Exception from HRESULT: 0x80131040)

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.IO.FileLoadException: Could not load file or assembly 'System.Web.Http, Version=5.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The located assembly's manifest definition does not match the assembly reference. (Exception from HRESULT: 0x80131040)

**Source Error:**

```
Line 19:         RouteConfig.RegisterRoutes(RouteTable.Routes);
Line 20:         BundleConfig.RegisterBundles(BundleTable.Bundles);
Line 21:     }
Line 22: }
Line 23: }
```

Se debe a una versión diferente de WebApi.Cors (5.2.3) y System.Web.Http (5.0.0), así que vamos a instalar la misma versión de WebApi.Core (Not Cors es Core). resolverá este problema de ensamblaje. para eso puedes ejecutar el comando NuGet desde la *consola del administrador de paquetes*.

```
Install-Package Microsoft.AspNet.WebApi.Core -Version
```

Así que aquí lo hemos completado con Web API Project. Volver al proyecto angular 5.

## Estructura del proyecto Angular 5

```
• src
+---• app
|   +---• employees
|       |--employees.component.ts|.html|.css
|       +---• employee (employee form)
|           |--employee.component.ts|.html|.css
|       +---• employee-list (list inserted employees)
|           |--employee-list.component.ts|.html|.css
|       +---• shared
|           |--employee.service.ts
|           |--employee.model.ts
|       |--app.module.ts (configured firebase connection)
+---• environments
|   |--environment.ts
|--index.html (cdn path for bootstrap and font awesome icon)
```

Esta es nuestra estructura de aplicación, los *empleados* componentes serán el componente principal para el componente *empleado* y *lista de empleados*.

Vamos a inyectar *EmployeeService* dentro de los *empleados* del componente principal. y allí podemos acceder a la misma instancia de servicio inyectado desde los componentes secundarios *Employee* and *Employee-List*. así que cada vez que hagamos cambios en un componente secundario, el mismo cambio también se puede ver desde otro componente secundario.

## Inyectar servicio de empleado en componentes:

en primer lugar, inyecte la clase de servicio dentro del componente principal *Empleados*

```
/src/employees/employees.component.ts
1  ...
2  import { EmployeeService } from '../shared/employee.service'
3  @Component({
4    ...
5    providers : [EmployeeService]
6  })
7  export class EmployeesComponent implements OnInit {
8    constructor(private employeeService : EmployeeService) { }
9    ...
10 }
```

Para inyectar una clase dentro de un componente, mencione la clase dentro de la matriz de *proveedores de componentes* y luego cree un parámetro privado dentro del *constructor del componente* .

Ahora podemos usar esta instancia inyectada en componentes secundarios, dentro del *archivo employee.component.ts*

```
/src/app/employee/employee.component.ts
1  ...
2  import { EmployeeService } from '../shared/employee.service';
3  @Component({
4    ...
5  })
6  export class EmployeeComponent implements OnInit {
7    constructor(private employeeService: EmployeeService) {
8    }
9    ...
10 }
```

Y dentro *del archivo employee-list.component.ts*

```
/src/app/employee/employee-list.component.ts
1  ...
2  import { EmployeeService } from '../shared/employee.service';
3  @Component({
4    ...
5  })
6  export class EmployeeListComponent implements OnInit {
7    constructor(private employeeService: EmployeeService) {
8    }
9    ...
10 }
```

## Formulario de operaciones Angular 5 CRUD

crearemos un formulario de empleado para implementar Insertar y actualizar la operación con el Componente del *empleado* . Así que vamos a crear un **Formulario Impulsado por Plantilla (TDF)** usando la propiedad del *Empleado seleccionado* de la Clase de Servicio del *Empleado* inyectada .

Entonces, antes que nada, tenemos que importar *FormsModule* y *HttpModule* en el archivo *appmodule.ts* .

```

/src/app/app.module.ts
1  ...
2  import { FormsModule } from '@angular/forms';
3  import { HttpClientModule } from '@angular/http';
4  ...
5  @NgModule({
6    ...
7    imports: [
8      ...
9      FormsModule,
10     HttpClientModule
11   ],
12   ...

```

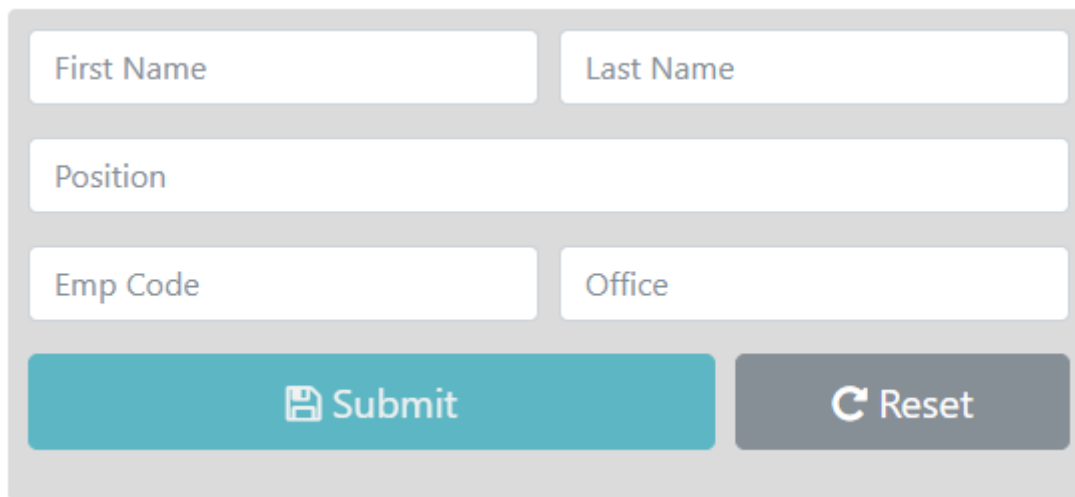
Para que pueda usar el siguiente html en el archivo *employee.component.html*

```

/src/app/employees/employee/employee.component.html
1  <form class="emp-form" #employeeForm="ngForm" (ngSubmit)="onSubmit(employeeFo
2    <input type="hidden" name="EmployeeID" #EmployeeID="ngModel" [(ngModel)]="e
3    <div class="form-row">
4      <div class="form-group col-md-6">
5        <input class="form-control" name="FirstName" #FirstName="ngModel" [(ngM
6          placeholder="First Name" required>
7        <div class="validation-error" *ngIf="FirstName.invalid && FirstName.tou
8      </div>
9      <div class="form-group col-md-6">
10       <input class="form-control" name="LastName" #LastName="ngModel" [(ngMod
11         required>
12       <div class="validation-error" *ngIf="LastName.invalid && LastName.touch
13     </div>
14   </div>
15   <div class="form-group">
16     <input class="form-control" name="Position" #Position="ngModel" [(ngModel
17   </div>
18   <div class="form-row">
19     <div class="form-group col-md-6">
20       <input class="form-control" name="EmpCode" #EmpCode="ngModel" [(ngModel
21     </div>
22     <div class="form-group col-md-6">
23       <input class="form-control" name="Office" #Office="ngModel" [(ngModel)]
24     </div>
25   </div>
26   <div class="form-row">
27     <div class="form-group col-md-8">
28       <button [disabled]="!employeeForm.valid" type="submit" class="btn btn-l
29       <i class="fa fa-floppy-o"></i> Submit</button>
30     </div>
31     <div class="form-group col-md-4">
32       <button type="button" class="btn btn-lg btn-block btn-secondary" (click
33       <i class="fa fa-repeat"></i> Reset</button>
34     </div>
35   </div>
36 </form>

```

y este formulario de empleado se verá así.



Formulario de empleado con los siguientes campos:

- First Name
- Last Name
- Position
- Emp Code
- Office
- Submit (botón con ícono de disco)
- Reset (botón con ícono de flecha circular)

Este diseño de formulario está inspirado en los estilos de formularios personalizados de Bootstrap (<https://goo.gl/5sDXZj>) . Se agrega un campo oculto para la propiedad *EmployeeID* .

## Validación de formulario

El atributo *obligatorio* se agrega a los cuadros de texto *Nombre* y *Apellido* , por lo que estos dos campos son obligatorios para enviar este formulario. Cuando estos cuadros de texto no sean válidos, la clase *ng-invalid* y *ng-sucia* se agregará automáticamente. por lo tanto, basado en estas clases, hemos implementado la validación de formularios.

cuando estos cuadros de texto no son válidos, el formulario de empleado como conjunto no es válido, por lo que agregamos el atributo de deshabilitar condicional al botón Enviar.

```
1 <button
2   [disabled]="!employeeForm.valid"
3   type="submit" class="btn btn-lg btn-block btn-info">
4     <i class="fa fa-floppy-o"></i> Submit</button>
```

para mostrar el error de validación, mostraremos un borde rojo alrededor de este cuadro de texto usando CSS. complete las reglas de css para esta aplicación en el archivo css global *styles.css* .

```
/src/styles.css CSS
1 form.emp-form{
2   background-color: #dbdbdb;
3   border-radius: 4px;
4   padding: 10px;
5 }
6 div.validation-error{
7   color: red;
8   text-align: center;
9 }
10 button:hover,a.btn:hover{
11   cursor: pointer;
12 }
```

# Insertar, actualizar y restablecer la operación

Dentro del archivo *employee.component.ts* , escribiremos el código para Insertar, Actualizar y Eliminar Operación. Antes de eso, voy a instalar *ngx-toastr* del paquete npm. este paquete nos ayuda a mostrar el mensaje de notificación dentro de aplicaciones angulares.

## Instalación del paquete ngx-toastr

Para instalar el paquete, puede usar el siguiente comando npm.

```
npm install ngx-toastr --save
```

luego agregue ToastrModule dentro del archivo *appmodule.ts* .

```
1 ...  
2 import { ToastrModule } from 'ngx-toastr';  
3 @NgModule({  
4   ...  
5   imports: [  
6     ...  
7     ToastrModule.forRoot()  
8   ],  
9   ...
```

Ahora puede agregar el siguiente código dentro del archivo de mecanografía del componente del *empleado* .

```
/src/app/employees/employee/employee.component.ts
```

```
1 import { Component, OnInit } from '@angular/core';
2 import { NgForm } from '@angular/forms'
3
4 import { EmployeeService } from '../shared/employee.service'
5 import { ToastrService } from 'ngx-toastr'
6 @Component({
7   selector: 'app-employee',
8   templateUrl: './employee.component.html',
9   styleUrls: ['./employee.component.css']
10 })
11 export class EmployeeComponent implements OnInit {
12
13   constructor(private employeeService: EmployeeService, private toastr: Toast
14
15   ngOnInit() {
16     this.resetForm();
17   }
18
19   resetForm(form?: NgForm) {
20     if (form != null)
21       form.reset();
22     this.employeeService.selectedEmployee = {
23       EmployeeID: null,
24       FirstName: '',
25       LastName: '',
26       EmpCode: '',
27       Position: '',
28       Office: ''
29     }
30   }
31
32   onSubmit(form: NgForm) {
33     if (form.value.EmployeeID == null) {
34       this.employeeService.postEmployee(form.value)
35         .subscribe(data => {
36           this.resetForm(form);
37           this.employeeService.getEmployeeList();
38           this.toastr.success('New Record Added Successfully', 'Employee Reg
39         })
40     }
41     else {
42       this.employeeService.putEmployee(form.value.EmployeeID, form.value)
43         .subscribe(data => {
44           this.resetForm(form);
45           this.employeeService.getEmployeeList();
46           this.toastr.info('Record Updated Successfully!', 'Employee Register')
47         });
48     }
49   }
50 }
```

La función *resetForm* se utiliza para restablecer el valor de los controles de formulario a la etapa inicial. Llamamos a esta función desde el evento de clic del botón de reinicio y desde *ngOnInit* Lifecycle Hook para inicializar el formulario.

Dentro de la función de envío de formulario submit *OnSubmit*, implementamos operaciones de inserción y actualización basadas en el valor de *EmployeeID*. Para mostrar el mensaje de éxito, utilizamos el objeto de clase *ToastrService* *toastr*.

# Lista de registros insertados y operación de eliminación

Usando el componente de la lista de empleados, enumeraremos todos los empleados insertados e implementaremos la operación Eliminar.

puede agregar el siguiente componente dentro de la lista de empleados.

```
/src/app/employees/employee-list/employee-list.component.ts
1 import { Component, OnInit } from '@angular/core';
2
3 import { EmployeeService } from '../shared/employee.service'
4 import { Employee } from '../shared/employee.model';
5 import { ToastrService } from 'ngx-toastr';
6 @Component({
7   selector: 'app-employee-list',
8   templateUrl: './employee-list.component.html',
9   styleUrls: ['./employee-list.component.css']
10 })
11 export class EmployeeListComponent implements OnInit {
12
13   constructor(private employeeService: EmployeeService, private toastr : Toastr
14
15   ngOnInit() {
16     this.employeeService.getEmployeeList();
17   }
18
19   showForEdit(emp: Employee) {
20     this.employeeService.selectedEmployee = Object.assign({}, emp);
21   }
22
23
24   onDelete(id: number) {
25     if (confirm('Are you sure to delete this record ?') == true) {
26       this.employeeService.deleteEmployee(id)
27         .subscribe(x => {
28           this.employeeService.getEmployeeList();
29           this.toastr.warning("Deleted Successfully", "Employee Register");
30         })
31     }
32   }
33 }
```

Dentro de esto hemos inyectado EmployeeService y ToastrService Class.

Dentro de *ngOnint* Lifecycle Hook, llamamos a *getEmployeeList* de la clase *EmployeeService*. Almacenará la colección de *empleados de la tabla Employee* dentro de la lista *employeeList*. Ahora podemos usar esta matriz para enumerar la colección de empleados. Puede agregar el siguiente código html dentro *del archivo employee-list.component.html*.

```
/src/app/employees/employee-list/employee-list.component.html
```

```

1 <table class="table table-sm table-hover">
2   <tr *ngFor="let employee of employeeService.employeeList">
3     <td>{{employee.FirstName}} - {{employee.LastName}}</td>
4     <td>{{employee.EmpCode}}</td>
5     <td>
6       <a class="btn" (click)="showForEdit(employee)">
7         <i class="fa fa-pencil-square-o"></i>
8       </a>
9       <a class="btn text-danger" (click)="onDelete(employee.EmployeeID)">
10        <i class="fa fa-trash-o"></i>
11      </a>
12    </td>
13  </tr>
14 </table>

```

El diseño de los componentes se ve así

Fiona - Green	FG		
Ashton - Cox	AC		
Cedric - Kelly	CK		
Gavin - Cortez	GC		
James - Jones	JJ		

cuando hacemos clic en el botón del lápiz llamaremos a la función *showForEdit* para llenar el registro correspondiente dentro del formulario del empleado. Al usar el icono de la papelera implementamos la operación de eliminación con la función *onDelete*.

Descargue el código fuente del proyecto desde GitHub: Angular 5 con API web - Operaciones CRUD. (<https://github.com/DotnetMob/Angular-5-With-Web-API-CRUD-Application>)

En mi artículo anterior hablamos sobre la implementación de Angular 5 CRUD Operations With Firebase (<http://www.dotnetmob.com/angular-5-tutorial/angular-5-crud-operations-with-firebase/>). Por favor, lea el artículo si no ha leído antes.

## Video Tutorial paso a paso



## Angular 4 CRUD With Web API



Gracias por la visita, espero que les haya gustado, háganme saber sus comentarios.

**35**  
COMPARTE

**f** Compartir (<https://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.dotnetmob.com%2Fangular-5-tutorial%2Fangular-5-with-web-api-crud-application%2F>)

**🐦** Pfo (<https://twitter.com/intent/tweet?text=Angular%205%20with%20Web%20API%20%E2%80%93%20CRUD%20Operations&url=http://www.dotnetmob.com/angular-5-tutorial/angular-5-with-web-api-crud-application/&via=DotnetMob>)

**in** LinkedIn (<https://www.linkedin.com/shareArticle?trk=Angular+5+with+Web+API+%E2%80%93+CRUD+Operations&url=http%3A%2F%2Fwww.dotnetmob.com%2Fangular-5-tutorial%2Fangular-5-with-web-api-crud-application%2F>)

**g** Google (<https://plus.google.com/share?text=Angular+5+with+Web+API+%E2%80%93+CRUD+Operations&url=http%3A%2F%2Fwww.dotnetmob.com%2Fangular-5-tutorial%2Fangular-5-with-web-api-crud-application%2F>)

**✉** Suscribirse



**Tags:** angular 5 crud tutorial (<http://www.dotnetmob.com/tag/angular-5-crud-tutorial/>) , angular 5 crud con Web API (<http://www.dotnetmob.com/tag/angular-5-crud-with-web-api/>) , angular 5 Web API (<http://www.dotnetmob.com/tag/angular-5-web-api/>) , angular 5 Web API ejemplo (<http://www.dotnetmob.com/tag/angular-5-web-api->

example/), angular 5 con API web (<http://www.dotnetmob.com/tag/angular-5-with-web-api/>), asp.net web api angular crud (<http://www.dotnetmob.com/tag/asp-net-web-api-angular-crud/>), llamada web api desde angular 5 (<http://www.dotnetmob.com/tag/call-web-api-from-angular-5/>)

### Compartir esta publicacion:

### Deja una respuesta

#### 5 comentarios sobre "Angular 5 con API web - Operaciones CRUD"

Notificar de

nuevos comentarios de seguimi

Email



Join the discussion

Ordenar por: más nuevos | más antiguo | el más votado

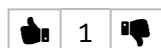


vivek GC



Buen ejemplo muchas gracias por estos artículos

Huésped



1



Hace 18 días 12 horas

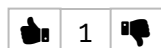


Chris K



Muy bien escrito para mi Simple, limpio e informativo. Gracias.

Huésped



1



Hace 18 días 9 horas



Shamseer

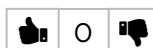


(<http://www.dotnetmob.com/author/Shamseer/>)

Gracias, Cris.

(<http://www.dotnetmob.com/author/Shamseer/>)

Autor



0



Hace 10 días 18 horas



Huésped

Facundo D



Excelente ejemplo ... cosas muy

lindas están detrás de la rama Angular v5.

Tenga en cuenta que el componente Http -

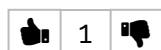
<https://angular.io/api/http/Http> (<https://angular.io/api/http/Http>)

- está en desuso en favor de HttpClient -

<https://angular.io/api/common/http/HttpClient>(<https://angular.io/api/common/http/HttpClient>) - que también

tiene algo de magia json constrúyelo.

¡Aclamaciones!



1



RESPUESTA

🕒 Hace 10 días 21 horas



(http://www.dotnetmob.com/author/Shamseer/)

Autor

Shamseer



(http://www.dotnetmob.com/author/Shamseer/)

Si, los próximos tutoriales /

artículos angulares usarán HttpClient. Gracias por la

sugerencia.



0



RESPUESTA

🕒 Hace 10 días 18 horas

## DOTNET MOB

Todo para apasionados .Net Developers. Tengo una inmensa experiencia en tecnología .Net. Así que pensé compartir mis ideas y experiencias como tutoriales y artículos aquí. Happy Coding :)