

# Arquitecturas de Software para Aplicaciones Empresariales

**Crear clientes y listar clientes mediante angular  
y servicios REST**



## **PROGRAMA DE INGENIERIA DE SISTEMAS**

Ing. Daniel Eduardo Paz Perafán (danielp@Unicauca.edu.co)

Ing. Pablo A. Magé (pimage@Unicauca.edu.co)

## Uso compartido de recursos entre orígenes o CORS

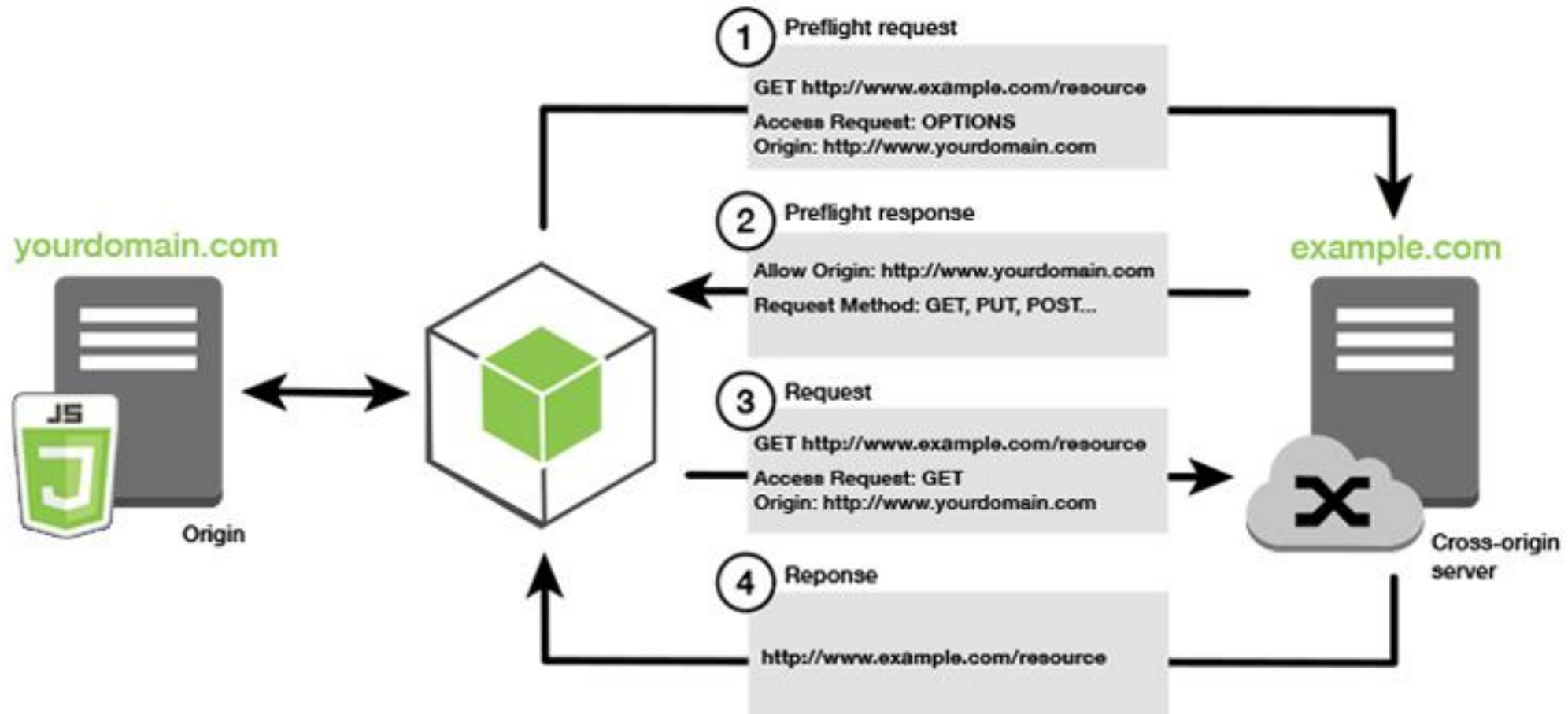
Por razones de seguridad, los navegadores prohíben las llamadas AJAX a los recursos que residen fuera del origen actual.

El uso compartido de recursos entre orígenes o CORS es una característica de seguridad de los navegadores web modernos. Habilita a los navegadores web para que puedan negociar qué dominios pueden realizar solicitudes de sitios web o servicios externos.

CORS determina si se permitirá el uso compartido de recursos en una solicitud entre orígenes basándose en:

- El dominio específico que efectúa la solicitud.
- El tipo de solicitud HTTP que se realiza (GET, PUT, POST, DELETE, etc.).

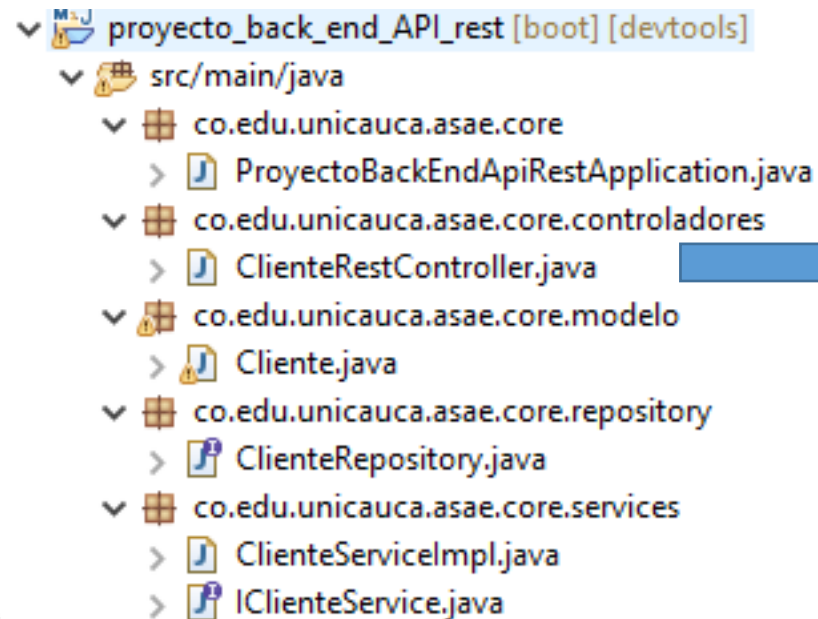
## Uso compartido de recursos entre orígenes o CORS



## Controlador

Con la anotación **@RestController**, Spring Boot automáticamente transformará a JSON las entidades devueltas por el método `index()`, que se anota con **@GetMapping**, y las enviará de vuelta al cliente en el cuerpo de la respuesta.

La anotación **@CrossOrigin** (`origins = "http: // localhost: 4200"`) permite solicitudes desde un origen determinado.



```
package co.edu.unicauca.asae.core.controladores;

import java.util.List;

@CrossOrigin(origins = { "http://localhost:4200" })
@RestController
@RequestMapping("/api")
public class ClienteRestController {

    @Autowired
    private IClienteService clienteService;

    @GetMapping("/clientes")
    public List<Cliente> index() {
        return clienteService.findAll();
    }

}
```

# Consumiendo el servicio

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS

The screenshot displays the Postman application interface. The top bar includes the Postman logo, menu items (File, Edit, View, Help), and workspace controls (My Workspace, Invite). The left sidebar shows the 'History' tab with two recent GET requests to `http://localhost:8085/api/clientes`. The main panel shows an 'Untitled Request' for the same endpoint. The 'Query Params' table is empty. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response status is 200 OK, with a time of 3.21 s and a size of 1.46 KB.

**Query Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description

**Response Body (JSON):**

```
1 {
2   {
3     "id": 1,
4     "nombre": "Andrés",
5     "apellido": "Guzmán",
6     "email": "profesor@bolsadeideas.com",
7     "createAt": "2018-01-01"
8   },
9   {
10    "id": 2,
11    "nombre": "Mr. John",
12    "apellido": "Doe",
13    "email": "iohn.doe@email.com".
```

# Consumiendo el servicio con angular

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS

## En el archivo app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, Component } from '@angular/core';

import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { DirectivaComponent } from './directiva/directiva.component';
import { ClientesComponent } from './clientes/clientes.component';
import { ClienteService } from './clientes/cliente.service';
import { RouterModule, Routes } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
```

➔ Agregar el import

```
@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    FooterComponent,
    DirectivaComponent,
    ClientesComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot(routes)
  ],
```

➔ Agregar el import

## En el archivo cliente.service.ts

```
import { Injectable } from '@angular/core';
import { CLIENTES } from './clientes.json';
import { Cliente } from './cliente';
import { Observable } from 'rxjs';
import { of } from 'rxjs';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class ClienteService {

  private urlEndPoint: string = 'http://localhost:8085/api/clientes';
  constructor(private http: HttpClient){

  }

  getClientes(): Observable<Cliente[]>
  {
    return this.http.get<Cliente[]>(this.urlEndPoint);
  }
}
```

Agregar la URL del servidor

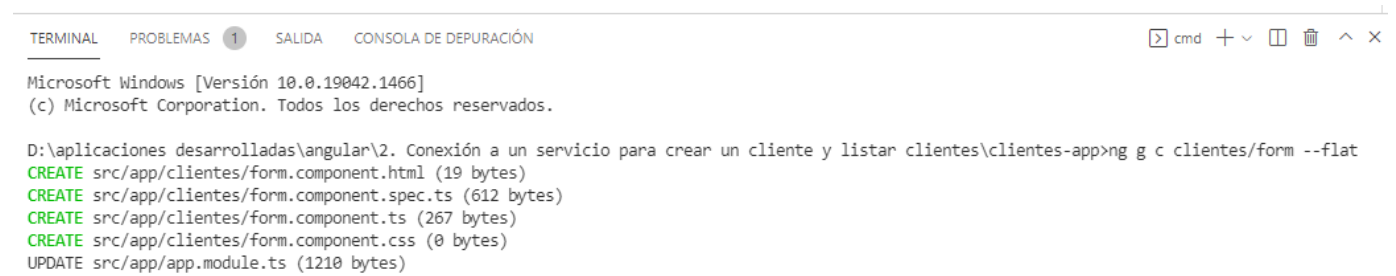
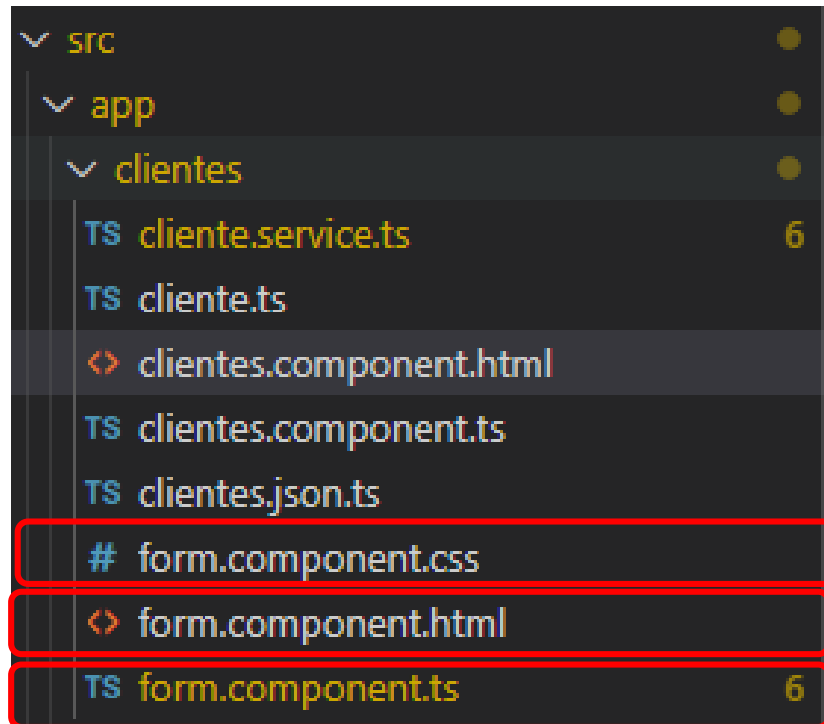
Atributo que permite la  
inyección de una dependencia

Consumir el servicio  
mediante un objeto de tipo  
HttpClient

# Creando el componente form

Creamos un nuevo componente con el comando `ng g c clientes/form --flat`

La bandera `--flat` omite crear la carpeta que contendrá el componente





# Creando el componente form

## Abrir el archivo app.module.ts

De manera automática se ha importado el archivo componente FormComponent y se ha registrado en la sección declarations

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, Component } from '@angular/core';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { DirectivaComponent } from './directiva/directiva.component';
import { ClientesComponent } from './clientes/clientes.component';
import { ClienteService } from './clientes/cliente.service';
import { RouterModule, Routes } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
import { FormComponent } from './clientes/form.component';

const routes: Routes = [
  {path: '', redirectTo: '/clientes', pathMatch: 'full'},
  {path: 'directivas', component: DirectivaComponent},
  {path: 'clientes', component: ClientesComponent}
];

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    FooterComponent,
    DirectivaComponent,
    ClientesComponent,
    FormComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule
  ],
  providers: [
    ClienteService
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

# Registrar modulo para trabajar con formularios

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS

Abrir el archivo app.module.ts, y registrar un modulo para trabajar con formularios

Importamos el FormsModule



```
import {RouterModule, Routes} from '@angular/router';
import {HttpClientModule} from '@angular/common/http';
import { FormComponent } from './clientes/form.component';
import {FormsModule} from '@angular/forms';

const routes: Routes = [
  {path: '', redirectTo: '/clientes', pathMatch: 'full'},
  {path: 'directivas', component: DirectivaComponent},
  {path: 'clientes', component: ClientesComponent}
];

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    FooterComponent,
    DirectivaComponent,
    ClientesComponent,
    FormComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot(routes)
  ],
```

Lo registramos en la sección imports



# Modificando el FormComponent

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS

## Clase cliente del modelo

TS cliente.ts X

src > app > clientes > TS cliente.ts > ...

```
1  export class Cliente {  
2      id!: number;  
3      nombre!: string;  
4      apellido!: string;  
5      createAt!: string;  
6      email!: string;  
7  
8  }
```

# Modificando el FormComponent

Creamos en el FormComponent un atributo de tipo cliente (clase del modelo), el cual va a almacenar la información del cliente a crear o a actualizar, de la siguiente forma:

```
private cliente: Cliente = new Cliente();
```

Importamos la clase cliente



```
import { Component, OnInit } from '@angular/core';  
import { Cliente } from '../cliente';
```

Atributo nuevo



```
public cliente: Cliente = new Cliente();
```

```
import { Component, OnInit } from '@angular/core';  
import { Cliente } from '../cliente';  
  
@Component({  
  selector: 'app-form',  
  templateUrl: './form.component.html',  
  styleUrls: ['./form.component.css']  
})  
export class FormComponent implements OnInit {  
  
  public cliente: Cliente = new Cliente();  
  
  ~~~  
  ngOnInit(): void {  
  }  
  
}
```

# Modificando el FormComponent

Creamos un atributo en el FormComponent el cual va a almacenar el titulo del formulario.

```
import { Component, OnInit } from '@angular/core';
import { Cliente } from '../cliente';

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

  public cliente: Cliente = new Cliente();
  public titulo: string = 'Crear cliente';

  ngOnInit(): void {
  }
}
```

En el archivo form.component.html creamos el formulario mediante html. A continuación se muestra la estructura para cada campo del cliente.

```
<div class="card bg-dark text-white">
  <div class="card-header">{{ titulo }}</div>
  <div class="card-body">

    <form>
      <div class="form-group row">
        <label for="" class="col-form-label col-sm-2"></label>
        <div class="col-sm-6">
          <input type="text" class="form-control" name="">
        </div>
      </div>

    </form>

  </div>
</div>
```

EL formulario completo quedaría de la siguiente forma:

```
<div class="card bg-dark text-white">
  <div class="card-header">{{ titulo }}</div>
  <div class="card-body">
    <form>
      <div class="form-group row">
        <label for="nombre" class="col-form-label col-sm-2">Nombre</label>
        <div class="col-sm-6">
          <input type="text" class="form-control" name="nombre">
        </div>
      </div>

      <div class="form-group row">
        <label for="apellido" class="col-form-label col-sm-2">Apellido</label>
        <div class="col-sm-6">
          <input type="text" class="form-control" name="apellido">
        </div>
      </div>

      <div class="form-group row">
        <label for="email" class="col-form-label col-sm-2">Email</label>
        <div class="col-sm-6">
          <input type="text" class="form-control" name="email">
        </div>
      </div>
    </form>
  </div>
</div>
```

Campo para capturar el  
nombre del cliente

Campo para capturar el  
apellido del cliente

Campo para capturar el  
email del cliente

La directiva **ngModel** permite hacer un binding entre el modelo y la vista

En el siguiente ejemplo el valor que el usuario escriba en el input identificado como nombre, se va a almacenar en el campo nombre del atributo cliente del FormComponent. Además, el valor almacenado en el campo nombre del atributo cliente se va a almacenar en el input identificado como nombre.

```
<div class="form-group row">
  <label for="nombre" class="col-form-label col-sm-2">Nombre</label>
  <div class="col-sm-6">
    <input type="text" class="form-control" [(ngModel)]="cliente.nombre" name="nombre">
  </div>
</div>
```



# Binding entre el formulario y el modelo

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS

```
<div class="card bg-dark text-white">
  <div class="card-header">{{ titulo }}</div>
  <div class="card-body">

    <form>
      <div class="form-group row">
        <label for="nombre" class="col-form-label col-sm-2">Nombre</label>
        <div class="col-sm-6">
          <input type="text" class="form-control" [(ngModel)]="cliente.nombre" name="nombre">
        </div>
      </div>

      <div class="form-group row">
        <label for="apellido" class="col-form-label col-sm-2">Apellido</label>
        <div class="col-sm-6">
          <input type="text" class="form-control" [(ngModel)]="cliente.apellido" name="apellido">
        </div>
      </div>

      <div class="form-group row">
        <label for="email" class="col-form-label col-sm-2">Email</label>
        <div class="col-sm-6">
          <input type="text" class="form-control" [(ngModel)]="cliente.email" name="email">
        </div>
      </div>

    </form>

  </div>
</div>
```

Binding hacia el nombre  
del cliente

Binding hacia el apellido  
del cliente

Binding hacia el email del  
cliente

El formulario almacena los datos en el atributo cliente del FormComponent y para enviarlos al back end debemos crear un botón que invoque un método de la siguiente forma:

```
<div class="form-group row">  
  <div class="col-sm-6">  
    <button class="btn btn-primary" role="button" (click)='crearCliente()'>Crear</button>  
  </div>  
</div>
```

Al dar click en el botón crear se ejecutara el método `crearCliente()` del FormComponent

El método crearCliente() se conectará con el API REST para almacenar el cliente

```
import { Component, OnInit } from '@angular/core';
import { Cliente } from '../cliente';

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

  private cliente: Cliente = new Cliente();
  private titulo: string = 'Crear cliente';
  constructor() { }

  ngOnInit(): void {
  }

  public crearCliente(): void{
    console.log('invocando al método crear cliente');
    console.log(this.cliente);
  }
}
```

En el archivo `app.module.ts` debemos crear una ruta hacia el componente `FormComponent` con el objetivo de mostrar el formulario que permite crear clientes.

```
const routes: Routes = [  
  {path: '', redirectTo: '/clientes', pathMatch: 'full'},  
  {path: 'directivas', component: DirectivaComponent},  
  {path: 'clientes', component: ClientesComponent},  
  {path: 'cliente/form', component: FormComponent}  
];
```

En el archivo `clientes.components.html` el cual permite mostrar el listado de clientes, debemos agregar un botón que permita mostrar el formulario para agregar un nuevo cliente.

```
<div class="my-2 text-left">  
  <button class="btn btn-rounded btn-primary" type="button" > Crear Cliente </button>  
</div>
```

Utilizando la directiva `[routerLink]` podemos crear un enlace hacia las rutas del proyecto, previamente creadas en el archivo `app.module.ts`

```
<div class="my-2 text-left">  
  <button class="btn btn-rounded btn-primary" type="button" [routerLink]="['/clientes/form']">  
    Crear Cliente  
  </button>  
</div>
```

# Consumiendo el servicio con angular

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS

En el archivo cliente.service debemos modificar la clase clienteService para que consuma el API REST

Importar las clases HttpClient y HttpHeaders

```
import {HttpClient, HttpHeaders} from '@angular/common/http';
```

Crear un atributo de tipo HttpHeaders con el fin de que las peticiones tengan como objetivo que el servidor nos devuelva información de tipo json

```
private httpHeaders = new HttpHeaders({'Content-Type': 'application/json'});
```

Crear un método que consuma el servicio web que permite crear un cliente.

```
create(cliente: Cliente) : Observable<Cliente> {  
    return this.http.post<Cliente>(this.urlEndPoint, cliente,{headers: this.httpHeaders})  
}
```

En el `form.component.ts` inyectar una dependencia de tipo **ClienteService**

```
constructor(private clienteService: ClienteService) { }
```

Inyectar una dependencia de tipo **Router**

```
constructor(private clienteService: ClienteService, private router:Router) { }
```

Invocar el método `create` del atributo `clienteService`

```
public crearCliente(): void{  
    this.clienteService.create(this.cliente)  
}
```

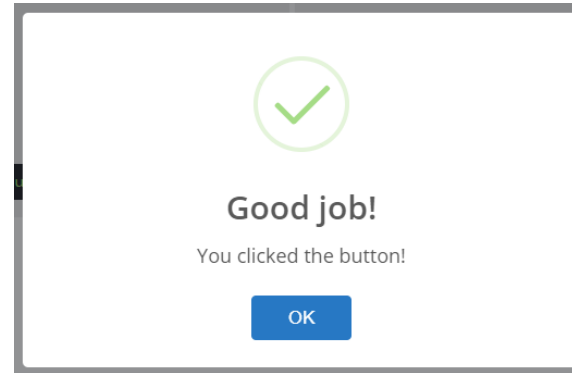
Suscribirse al método `create` del atributo `clienteService` y redireccional al usuario a la vista donde se listan los clientes

```
public crearCliente(): void{  
    this.clienteService.create(this.cliente).subscribe(  
        response => this.router.navigate(['/clientes'])  
    )  
}
```

<https://sweetalert2.github.io/>

Para instalarlo utilizamos el comando

```
npm install sweetalert2
```



Luego de la instalación en el archivo package.json aparece la dependencia

```
"dependencies": {  
  "@angular/animations": "^15.1.0",  
  "@angular/common": "^15.1.0",  
  "@angular/compiler": "^15.1.0",  
  "@angular/core": "^15.1.0",  
  "@angular/forms": "^15.1.0",  
  "@angular/platform-browser": "^15.1.0",  
  "@angular/platform-browser-dynamic": "^15.1.0",  
  "@angular/router": "^15.1.0",  
  "rxjs": "~7.8.0",  
  "sweetalert2": "^11.7.0",  
  "tslib": "^2.3.0",  
  "zone.js": "~0.12.0"
```



En el archivo form.component importamos la librería de la siguiente manera:

```
import swal from 'sweetalert2';
```

En el archivo form.component despues crear un cliente mostramos el mensaje de confirmación de la siguiente manera:

```
public crearCliente(): void{  
  this.clienteService.create(this.cliente)  
    .subscribe(  
      response =>  
      {  
        this.router.navigate(['/clientes']);  
        swal.fire('Nuevo cliente', `Cliente ${response.nombre} creado con éxito!`, 'success');  
      }  
    )  
}
```

Con la directiva `ngIf` preguntamos si no hay clientes registrados

```
<div *ngIf="clientes?.length==0" class="alert alert-info">  
  No hay registros en la base de datos  
</div>
```

Si no hay clientes registrados no se muestra la tabla

```
<table class="table table-bordered table-striped" *ngIf="clientes.length>0">
```

Lanzar la aplicación con el comando `ng serve -o --port 4444`

Proyecto de Clase Directivas Clientes

Clientes

Listado de clientes

Crear Cliente

No hay registros en la base de datos

Proyecto de Clase Directivas Clientes

Crear cliente

Nombre

Apellido

Email

Crear

Proyecto de Clase Directivas Clientes

Crear cliente

Nombre

JUAN

Apellido

PEREZ

Email

juan@unicauca.edu.co

Crear


Proyecto de Clase Directivas Clientes

Cientes

Listado de clientes

Crear Cliente

id	nombres	ap	fecha de creación
1	JUAN	PE	2023-01-24



**Nuevo cliente**

Cliente JUAN creado con éxito!

OK

© 2022 Proyecto de Clase | WebApp desarrollada con Angular Spring5 | Autor: ASAE


Proyecto de Clase Directivas Clientes

Cientes

Listado de clientes

Crear Cliente

id	nombres	ap	fecha de creación
1	JUAN	PE	2023-01-24



**Nuevo cliente**

Cliente JUAN creado con éxito!

OK

© 2022 Proyecto de Clase | WebApp desarrollada con Angular Spring5 | Autor: ASAE

Proyecto de Clase   Directivas   Clientes

Clientes

## Listado de clientes

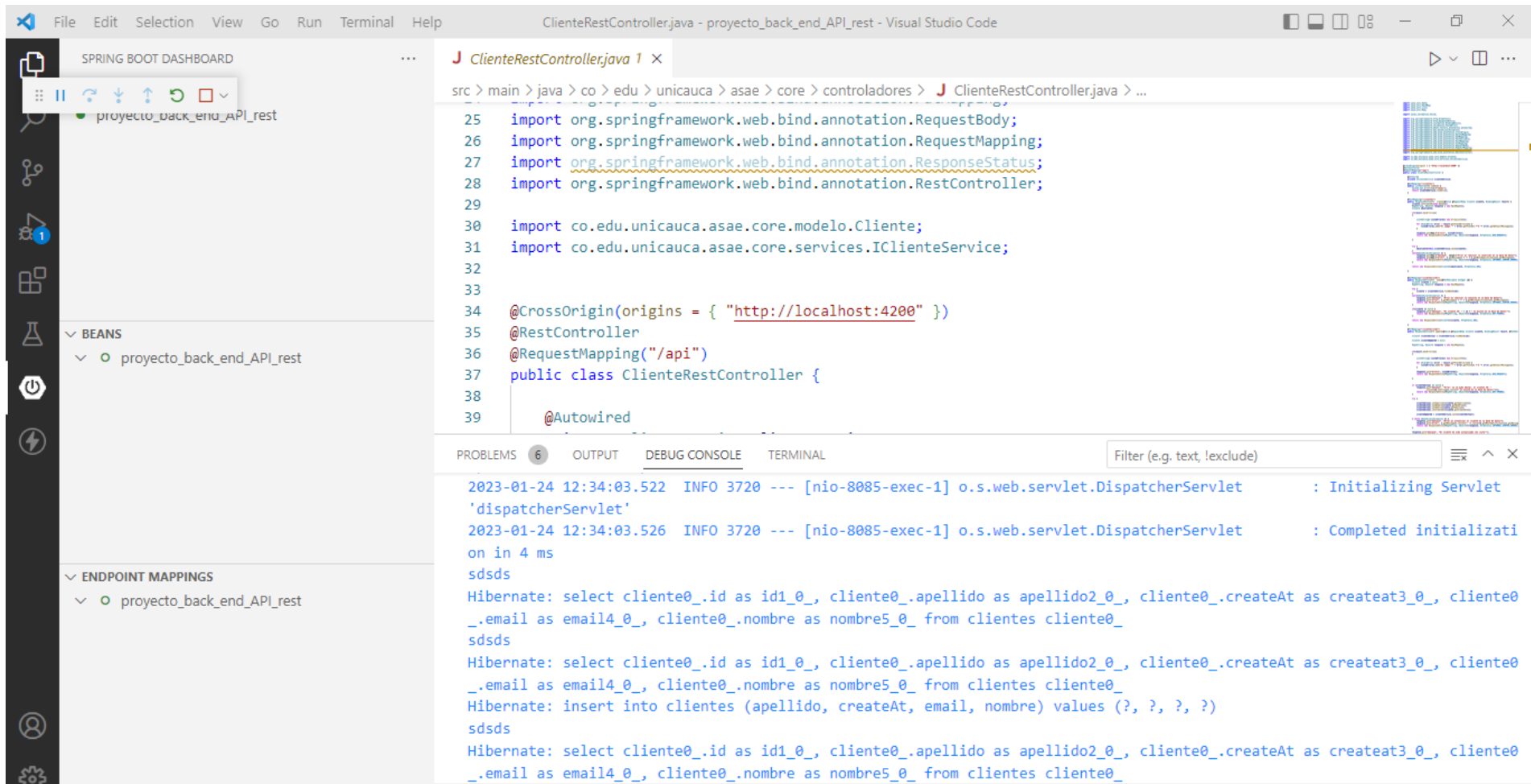
Crear Cliente

id	nombres	apellidos	email	fecha de creación
1	JUAN	PEREZ	juan@unicauca.edu.co	2023-01-24



# Aplicación funcionando

UNIVERSIDAD DEL CAUCA – FIET  
DEPARTAMENTO DE SISTEMAS



The screenshot shows the Visual Studio Code IDE with the following components:

- Spring Boot Dashboard:** Displays the project `proyecto_back_end_API_rest`.
- Beans:** Shows the `proyecto_back_end_API_rest` bean.
- Endpoint Mappings:** Shows the `proyecto_back_end_API_rest` endpoint.
- Source Code:** The file `ClienteRestController.java` is open, showing imports for Spring Web and the application's domain model and service layers.
- Debug Console:** Displays the following logs:

```
2023-01-24 12:34:03.522 INFO 3720 --- [nio-8085-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-01-24 12:34:03.526 INFO 3720 --- [nio-8085-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 4 ms
sdsds
Hibernate: select cliente0_.id as id1_0_, cliente0_.apellido as apellido2_0_, cliente0_.createAt as createat3_0_, cliente0_.email as email4_0_, cliente0_.nombre as nombre5_0_ from clientes cliente0_
sdsds
Hibernate: select cliente0_.id as id1_0_, cliente0_.apellido as apellido2_0_, cliente0_.createAt as createat3_0_, cliente0_.email as email4_0_, cliente0_.nombre as nombre5_0_ from clientes cliente0_
Hibernate: insert into clientes (apellido, createdAt, email, nombre) values (?, ?, ?, ?)
sdsds
Hibernate: select cliente0_.id as id1_0_, cliente0_.apellido as apellido2_0_, cliente0_.createAt as createat3_0_, cliente0_.email as email4_0_, cliente0_.nombre as nombre5_0_ from clientes cliente0_
```

Proyecto de Clase   Directivas   Clientes

Clientes

Listado de clientes


Crear Cliente

id	nombres	ape
1	JUAN	PER
2	DANIEL	PAZ

fecha de creación

2023-01-24

2023-01-24



**Nuevo cliente**

Cliente DANIEL creado con éxito!

OK

© 2022 Proyecto de Clase | WebApp desarrollada con Angular Spring5 | Autor: ASAE

Proyecto de Clase   Directivas   Clientes

Clientes

Listado de clientes

Crear Cliente

id	nombres	apellidos	email	fecha de creación
1	JUAN	PEREZ	juan@unicauca.edu.co	2023-01-24
2	DANIEL	PAZ	danielp@unicauca.edu.co	2023-01-24

## Considerar el código fuente del taller 1

### Requisito funcional a implementar

Yo como coordinador de la maestría quiero gestionar un nuevo estudiante para poder matricularlo en un curso, gestionar sus solicitudes y asociarlo a un proyecto de investigación. Los atributos del estudiante son los definidos en el taller 1.

### Condiciones de entrega

#### Crear un componente para el encabezado

- Crear un componente para el footer
- Crear un componente para listar los estudiantes
- Crear un componente para registrar los estudiantes
- Crear un servicio que permita listar los estudiantes y registrar los estudiantes mediante un servicio REST
- Crear una alerta cuando no existan estudiantes registrados y cuando un estudiantes se almacenó exitosamente

**Muchas gracias**  
**Preguntas**

