En este documento vamos a implementar una tabla donde se mostrarán los alumnos que nos devuelve el método del backend que hicimos en el documento anterior.

La tabla la vamos a realizar totalmente a mano y utilizando únicamente clases de bootstrap.

En la tabla tendremos las opciones para eliminar , editar y crear alumnos nuevos.

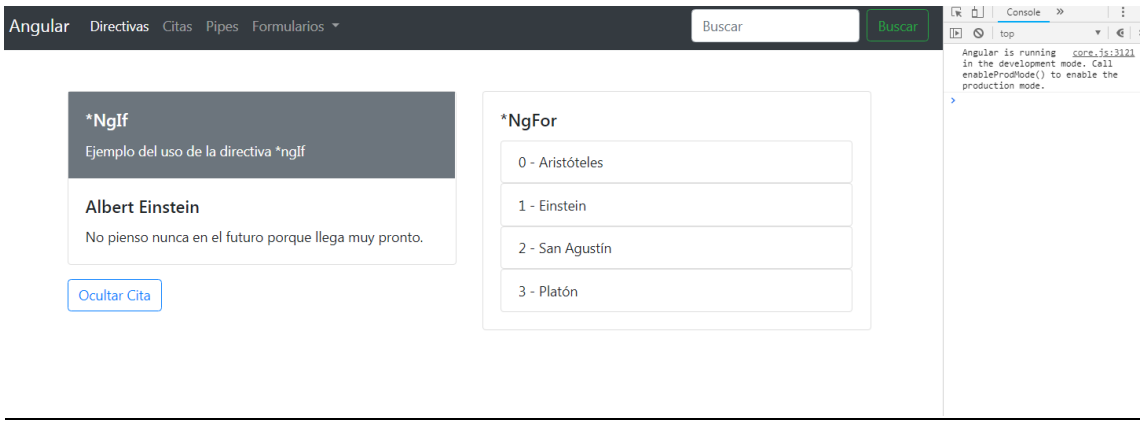
En este documento no vamos a utilizar la gestión de errores. Eso lo dejamos para otro documento, si es que estamos **animados**.

Aunque se podrían utilizar las tablas que nos ofrece PrimeNG y que son bastante fáciles de implementar en Angular.

También haré una introducción a la carga de componentes mediante Lazy Load o carga perezosa. Esto es interesante en proyectos grandes, ya que aligera mucho los javascript que se generan.

Pues vamos al lío.

Así tenemos la aplicación en este momento.



Lo primero que tenemos que hacer es crear el componente alumnos.

Este va a ser el componente que vamos a crear para que se **cargue de forma perezosa.**

Aunque es un poco lioso si no se ha hecho nunca, lo vamos a hacer de la forma tradicional y luego de la que dejaremos con Lazy Load.

Forma normal:

Creamos el componente Alumnos con ng g c components/alumnos –spec=false

Aquí le estamos diciendo que nos genere un componente llamado alumnos y que no nos genere los archivos de pruebas. (Si hay un poco de ánimo, os hago un documento de pruebas en angular)

Luego en el fichero app.routes.ts, incorporamos la ruta para cuando se pulse en la opción que ponemos en el navbar.

Creo que es fácil esto anterior, porque lo hemos hecho, pero debe quedar así:

En el navbar.component.html, la opción alumnos será:

```
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" href="#"
id="navbarDropdown" role="button" data-toggle="dropdown" aria-
haspopup="true" aria-expanded="false">
                  Formularios
                </a>
                <div class="dropdown-menu" aria-
labelledby="navbarDropdown">
                    <a class="dropdown-item"
[routerLink]="['formuhtml']">Html</a>
                    <a class="dropdown-item"
[routerLink]="['formucodigo']">Código Ts</a>
                </div>
            </li>
            <li class="nav-item">
                <a class="nav-link" routerLinkActive="active"
[routerLink]="['alumnos']">Alumnos</a>
            </li>
```

Justo debajo de la opción de los formularios.

En el app.routes.ts, hay que incorporar la entrada:

```
{ path: 'alumnosAlum', component: AlumnosComponent },
```

Si ejecutamos el programa, nos tiene que salir ya que navega a la opción Alumnos.

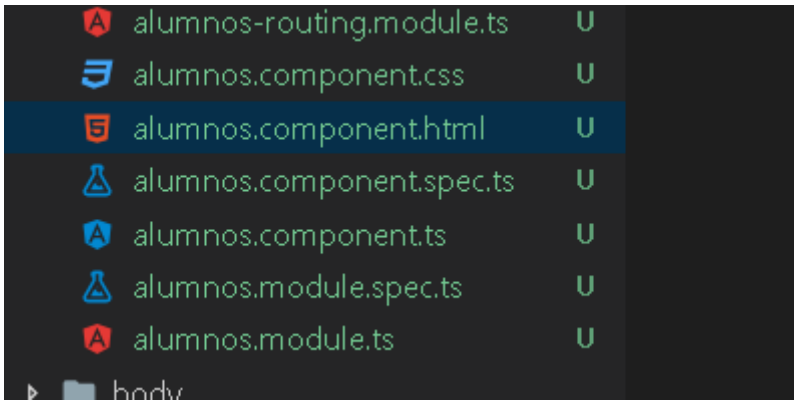
Si quieren se puede dejar así, pero la otra forma es que se cargue de forma perezosa cuando se pulsa la opción

Esto es un poco raro, pero si lo miras un par de veces se termina entendiendo.

Para ello, vamos a crear un módulo nuevo en la carpeta components/alumnos y su archivo de rutas particular.

Ejecutamos para ello el comando: Ng g m components/alumnos –routing (son dos guiones)

Nos tiene que generar los archivos siguiente.



Los importantes son el alumnos.module, donde vamos a incluir el módulo de alumnos y por lo tanto le quitaremos de app.module.

Y el otro archivo es el alumnos-routing.module, que tiene la navegación por este módulo.

En fin, que tienen que quedar así.

El fichero alumnos.module.ts tiene que tener:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { AlumnosRoutingModule } from './alumnos-routing.module';
import { AlumnosComponent } from './alumnos.component';

@NgModule({
  imports: [
    CommonModule,
    AlumnosRoutingModule
  ],
  declarations: [AlumnosComponent]
```

```
})
export class AlumnosModule { }
```

Luego el fichero de rutas alumnos-routing

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AlumnosComponent } from './alumnos.component';

const routes: Routes = [
  {
    path: '',
    component: AlumnosComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AlumnosRoutingModule { }
```

Como se puede ver, tenemos la ruta a la que hay que llamar y que vamos a configurar en el fichero de rutas de la aplicación, donde decimos que cuando se pulse en la opción Alumnos cargue este módulo.

Quedará por lo tanto así en el fichero app.routes:

```
      {
        path: 'alumnos',
        loadChildren: './components/alumnos/alumnos.module#AlumnosModule'
    },
```

En que decimos que cuando se invoque a alumnos cargue lo que hay en ese uri y en el módulo Alumnos

Ya sé que esto es un poco raro, pero ahora lo importante es que nuestra aplicación navegue a la página Alumnos.

Vamos a cear el servicio que nos devolverá la lista de los alumnos.

Para ello, desde la terminal, generamos el servicio escribiendo: ng g s services/alumnos --spec=false

Antes de usar peticiones http, tenemos que importar en el app.module, el

```
HttpClientModule
```

Por lo tanto, en los import ponemos este módulo y lo importamos arriba con:

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

Ojo que tiene que importase de angular/common/http.

Volvemos al servicio alumnos e inyectamos el constructor la clase HttpCliente.

Ya que estamos, escribimos nuestro primer método que devolverá un observable (petición asíncrona) con los datos del servidor.

Así tiene que quedar el servicio

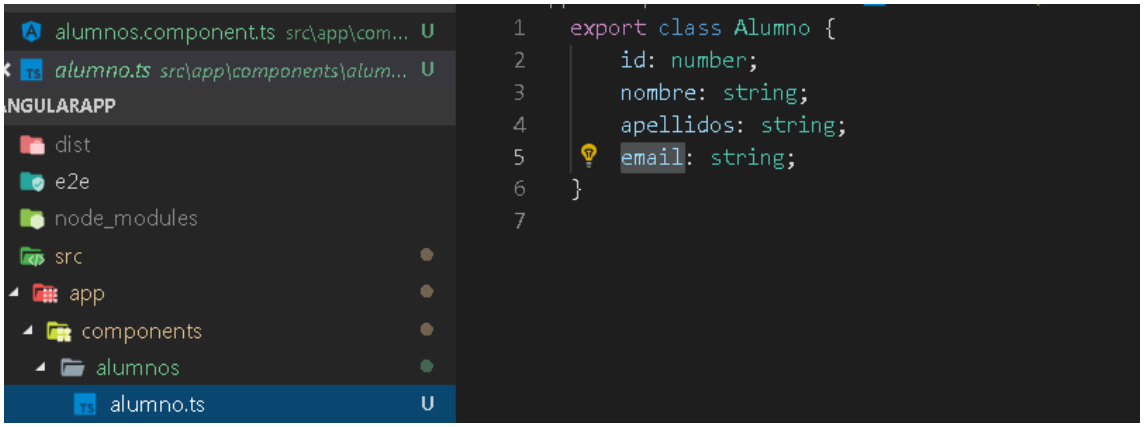
```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Alumno } from '../components/alumnos/alumno';

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

  private urlEndPoint = 'http://localhost:8080/api/alumnos';


  constructor(private http: HttpClient) { }

  getAlumnos() {
    return this.http.get( this.urlEndPoint);
  }
```

```
}
```

Creo que no es muy complejo el uso.

No sé si hace falta alguna aclaración.

Para usar la ayuda de TypeScrip, he creado una clase Alumno dentro de la carpeta alumnos y ha sido así:



Ahora vamos a la clase alumnos.component.ts

En esta clase, vamos a inyectar el servicio que hemos creado y vamos a crear un array de elementos alumnos donde almacenaremos los datos que vienen del servicio para mostrarlos luego con un ngfor en el html.

```
import { Component, OnInit } from '@angular/core';
import { Alumno } from './alumno';
import { AlumnoService } from '../../services/alumno.service';


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

  alumnos: Alumno[] = [];
```

```
  constructor(private alumnoService: AlumnoService) { }

  ngOnInit() {
    this.alumnoService.getAlumnos()
    .subscribe ( (data: Alumno[]) => {
      this.alumnos = data;
      console.log(this.alumnos);
    });

  }

}
```

El ngOnInit, se ejecuta siempre que se invoca la página. En este método invocamos el getAlumnos del servicio, pero no se dispara la petición hasta que no nos subscribimos y entonces nos devuelve un observable y cuando termina nos trae la data que como he puesto es de timo Array de alumnos y hacemos que nuestra propiedad this.alumnos se cargue con todos los alumnos. Luego hacemos un console.log para ver cómo va en consola.

ESTO ESTÁ BIEN, PERO SIEMPRE QUE ENTRA EN EL NGONINIT, HACE UNA SUBSRIPCIÓN AL OBSERVABRE Y NO LA SUSPENDE. ESTO PUEDE SER UN PROBLEMA Y VAMOS A CAMBIARLO MAS ABAJO

Escribamos el html para probar esto:

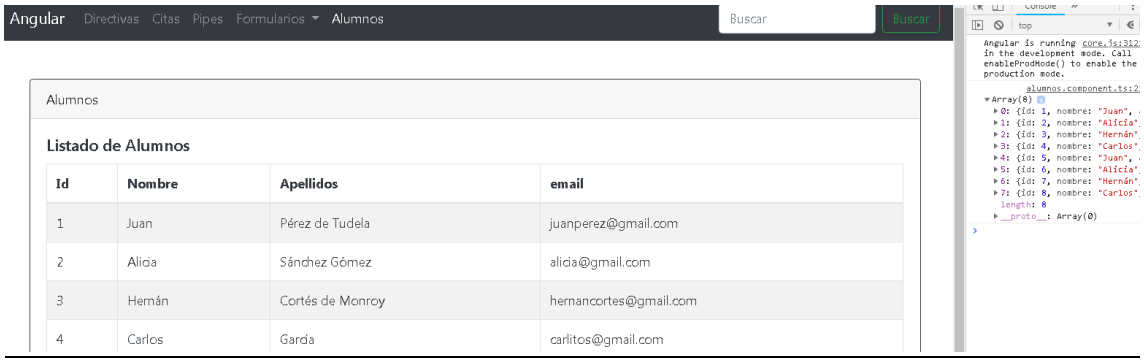
Creo que no necesita aclaración. Únicamente es un html con una tabla y algunas clases de bootstrap. Además tiene el ngfor, para recorrer el array de los alumnos.

El html debe dejarse así:

```
<div class="card border-dark mb-5">
    <div class="card-header">Alumnos</div>
    <div class="card-body text-dark">
        <h5 class="card-title">Listado de Alumnos</h5>

        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Nombre</th>
```

```
                    <th>Apellidos</th>
                    <th>email</th>

                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let alumno of alumnos ">
                    <td>{{ alumno.id }}</td>
                    <td>{{ alumno.nombre }}</td>
                    <td>{{ alumno.apellidos }}</td>
                    <td>{{ alumno.email }}</td>

                </tr>
            </tbody>
        </table>


    </div>
</div>
```

Así es como tiene que quedar la lista:



Ahora vamos a terminar escribiendo el resto de métodos en el servicio de alumnos para poderlos usar desde el componente.

Es decir, creamos los métodos update, delete , obtener un alumno y crear alumno.

El fichero alumno.service.ts tiene que quedar así, y os cuento.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Alumno } from '../components/alumnos/alumno';

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

  private urlEndPoint = 'http://localhost:8080/api/alumnos';

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


  constructor(private http: HttpClient) { }

  getAlumnos() {
    return this.http.get( this.urlEndPoint);
  }

  creaAlumno( alumno: Alumno ) {
    return this.http.post(this.urlEndPoint, alumno, { headers:
this.httpHeaders});
  }

  getAlumno ( id: number ) {
    return this.http.get( `${this.urlEndPoint}/${id}` );
  }

  updateAlumno( alumno: Alumno ) {
    return this.http.put(`${this.urlEndPoint}/${alumno.id}`, alumno,
{headers: this.httpHeaders});
  }

  deleteAlumno ( id: number ) {
    return this.http.delete ( `${this.urlEndPoint}/${id}` );
  }

}
```

Los métodos get, llaman a la url del endpoint y el el caso del getAlumno, le concatenamos el id del alumno que se recibirá por parámetro.

Todos estos métodos, devuelven un observable y no hubiese estado mal ponerlo, es decir, poner por ejemplo en el método crear lo siguiente

```
  creaAlumno( alumno: Alumno ): Observable<Alumno> {
    return this.http.post<Alumno>(this.urlEndPoint, alumno, { headers:
this.httpHeaders});
  }
```

Pero yo, no sé por qué, pero me he acostumbrado a no hacerlo, pero lo correcto es ponerlo.

En el caso del créate y el update, hay que incorporarle los headers tal cual como se hace en el postman.

Lo siguiente que vamos a hacer es cambiar la forma en la que se reciben los datos en la tabla, ahora en el ngOnInit() del componente alumnos, llenábamos un array de alumnos y comentaba que la subscripción no se cancelaba.

Una forma de cancelar la subscripción podría ser después de llenar los datos, pero hay otra forma y es crear la variable alumnos de tipo any e igualarla al observable. Luego, en el html utilizar el pipe async para que rellene los datos cuando estén.

Quedando el alumnos.component.ts de la forma:

```
import { Component, OnInit } from '@angular/core';
import { Alumno } from './alumno';
import { AlumnoService } from '../../services/alumno.service';


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

//  alumnos: Alumno[] = [];
  alumnos: any;

  constructor(private alumnoService: AlumnoService) { }
```

```
  ngOnInit() {
    // this.alumnoService.getAlumnos()
    // .subscribe ( (data: Alumno[]) => {
    //   this.alumnos = data;
    //   console.log(this.alumnos);
    // });

    this.alumnos = this.alumnoService.getAlumnos();


  }




}
```

Y en el html, el tr que hace la iteración quedando así:

```
<tbody>
                <tr *ngFor="let alumno of alumnos | async">
                    <td>{{ alumno.id }}</td>
                    <td>{{ alumno.nombre }}</td>
                    <td>{{ alumno.apellidos }}</td>
                    <td>{{ alumno.email }}</td>

                </tr>
            </tbody>
```

Antes de seguir, vamos a crear el componente del formulario, para poderlo utilizar para crear nuevos alumnos y para editarlos.

Para ello, ng g c components/alumnos/alumnosForm –spec=false –flat (para que no haga otra carpeta)

Una vez hecho, en el fichero de rutas incorporamos la llamada a ese formulario así:

```
    { path: 'alumnos/form', component: AlumnosFormComponent },
    { path: 'alumnos/form/:id', component: AlumnosFormComponent },
```

Una forma es el formulario vacío y otra enviando el id del alumno para la edición del elemento que se envíe.

Así tiene que quedar el formulario de alumnos:

```
<div class="card bg-white text-dark ">
    <div class="card-header">{{ titulo }} </div>
    <div class="card-body">
        <div class="mb-3">
            <button class="btn btn-outline-success"
routerLink="/alumnos">Volver</button>
        </div>

        <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)]="alumno.nombre" name="nombre">
                </div>
            </div>

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

```
            <div class="form-group row">
                <div class="col-sm-8">
                    <button *ngIf="!alumno.id" (click)="guardarAlumno()"
class="btn btn-outline-primary btn-block">Crear</button>
                    <button *ngIf="alumno.id"
(click)="actualizarAlumno()" class="btn btn-outline-primary btn-
block">Actualizar</button>
                </div>
            </div>
        </form>


    </div>

</div>
```

He puesto un botón volver y los botones de crear o actualizar se muestran si hay o no un id del alumno, para detectar si lo que se va a hacer es actualizar o crear.

Por lo demás, creo que todo lo que contiene es entendible, si algo no se entiende, me pueden mandar una pregunta y la contesto gustosamente.

Y el form-alumnos.ts tiene que tener esto

```
import { Component, OnInit } from '@angular/core';
import { Alumno } from './alumno';
import swal from 'sweetalert';
import { AlumnoService } from '../../services/alumno.service';
import { Router, ActivatedRoute } from '@angular/router';

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

  titulo = 'Alumnos';

  alumno: Alumno = new Alumno();
```

```
constructor(
  private alumnoService: AlumnoService,
  private router: Router,
  private activatedRoute: ActivatedRoute) { }

ngOnInit() {
  this.editarAlumno();
}

guardarAlumno() {
  this.alumnoService.creaAlumno ( this.alumno )
  .subscribe ( data => {
    console.log(data);
    this.router.navigate(['/alumnos']);
    swal(
      'Guardado',
      `El alumno ${this.alumno.nombre} ha sido guardado`,
      'success'
    );
  });

}

editarAlumno() {
  this.activatedRoute.params.subscribe( params => {
    const id = params['id'];
    if ( id ) {
      this.alumnoService.getAlumno(id).subscribe ( (data: Alumno) => {
        this.alumno = data;
        console.log(this.alumno);
      });
    }
  });
}

actualizarAlumno() {
  this.alumnoService.updateAlumno( this.alumno )
  .subscribe( (data: Alumno) => {
    this.router.navigate(['/alumnos']);
    swal(
      'Actualizado',
```

```
        `El alumno ${data.nombre} ha sido actualizado`,
        'success'
      );
    });
  }

}
```

En la que como algo nuevo, hemos hecho:

Hemos incorporado sweetalert para que queden más chulos los mensajes .

Hemos utilizado Router para navegar a otra página y ActivatedRoute para capturar los parámetros que vienen en la url

Además en el html, lo hemos mejorado incluyendo un alert para cuando no hay elementos en la tabla.

```
<div class="card border-dark mb-5">
    <div class="card-header">Alumnos</div>
    <div class="card-body text-dark">
        <h5 class="card-title">Listado de Alumnos</h5>
        <div class="mb-3">
            <button class="btn btn-outline-primary"
routerLink="/alumnos/form">Nuevo Alumno</button>
        </div>

        <ng-template #sindatos>
            <div class="alert alert-success" role="alert">
                No hay alumnos para listar
            </div>
        </ng-template>

        <table *ngIf="(alumnos | async)?.length > 0 else sindatos"
class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Nombre</th>
                    <th>Apellidos</th>
                    <th>email</th>
```

```
                    <th>
                        editar
                    </th>
                    <th>
                        borrar
                    </th>
                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let alumno of alumnos | async">
                    <td>{{ alumno.id }}</td>
                    <td>{{ alumno.nombre }}</td>
                    <td>{{ alumno.apellidos }}</td>
                    <td>{{ alumno.email }}</td>
                    <td>
                        <button type="button" class="btn btn-warning btn-
sm" name="button" [routerLink]="['/alumnos/form', alumno.id
]">editar</button>
                    </td>
                    <td>
                        <button type="button" class="btn btn-danger btn-
sm" name="buttondelete" (click)="borrarAlumno(alumno)">borrar</button>
                    </td>
                </tr>
            </tbody>
        </table>


    </div>
</div>
```

Este html, lo único que tiene de especial es que hemos incorporado al principio un alert para informar que no hay alumnos y el botón crear que nos lleva al formulario para crear un nuevo alumno.

Luego hemos puesto los botones para actualizar y borrar.

Para el caso de la opción borrar, hemos llamado a un método que está en alumnos.component.ts y que presentamos con sweetalert2 (que me acabo de dar cuenta que existe) para pedir confirmación .

Este método borrar es copiado / pegado de la página de sweetalert2, cambiando los textos y cuando se pulsa borrar, llamando al correspondiente método del servicio alumnos que hemos inyectado previamente.

Hago un inciso para contaros que para utilizar sweetalert, hay que seguir la instalación que está en

https://sweetalert.js.org/ (Versión 1)

y https://sweetalert2.github.io/ (versión 2)

Queda así:

```
import { Component, OnInit } from '@angular/core';
import { Alumno } from './alumno';
import { AlumnoService } from '../../services/alumno.service';
import Swal from 'sweetalert2';

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

//  alumnos: Alumno[] = [];
  alumnos: any;

  constructor(private alumnoService: AlumnoService) { }

  ngOnInit() {
    // this.alumnoService.getAlumnos()
    // .subscribe ( (data: Alumno[]) => {
    //   this.alumnos = data;
    //   console.log(this.alumnos);
    // });
```

```
    this.alumnos = this.alumnoService.getAlumnos();


  }

  borrarAlumno( alumno: Alumno ) {
    Swal.fire({
      title: 'Estás seguro',
      text: `Se va a proceder a borrar a ${alumno.nombre} `,
      type: 'warning',
      showCancelButton: true,
      confirmButtonColor: '#d33',
      cancelButtonColor: '#3085d6',
      confirmButtonText: 'Sí, bórralo'
    }).then((result) => {
      if (result.value) {
        this.alumnoService.deleteAlumno( alumno.id )
        .subscribe ( data => {
          this.alumnos = this.alumnoService.getAlumnos();
          Swal.fire(
            'Borrado!',
            'El alumno ha sido borrado.',
            'success'
          );

        });
      }
    });


  }

}
```

Creo que como punto de partida para consumir unos datos de un backend, es suficiente.

Nos queda para otro capítulo, incluir gestión de errores en el backend y presentarlos en el front end.

Nos queda algo importante, como es subir los cambios al repositorio:

Git status

Git add .

Git status

Git commit –m “Terminado el frontend sin control de errores”

Git push origin master