



os para crearlos

uando creamos una aplicación en Angular es la
sus componentes y relaciones entre ellas.

ework de Angular es delegar todas las
acceso a datos (peticiones y envío de datos) y
otras clases que colaboran con las

Angular CLI nos provee la capacidad de

servicio y luego consumirlo desde una

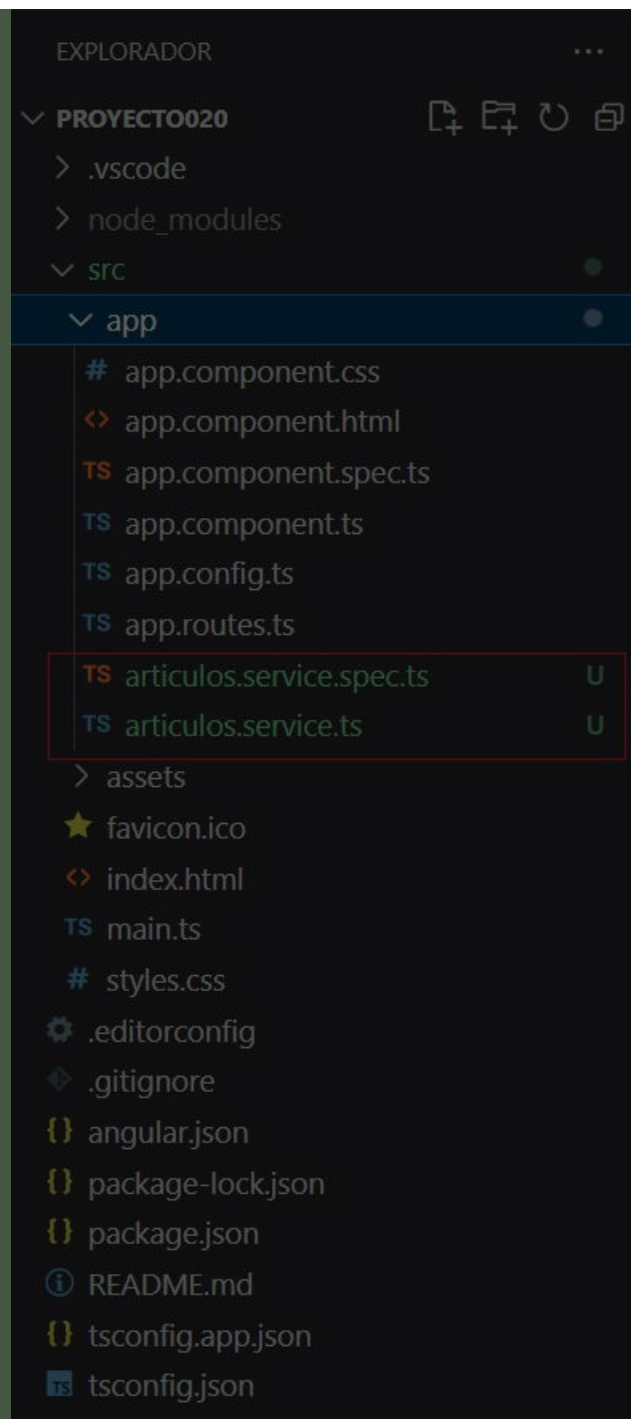
culos. Los artículos almacenarlos en un vector
para pedir los artículos a ser mostrados.

el proyecto020:

culos (en muchos casos como veremos más
datos de un servidor web):

service'.

Se crean dos archivos:



El código generado de la clase 'ArticulosService' es:

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

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

  constructor() { }

}
```

Lo modificamos por el siguiente código que permita recuperar desde la componente el vector de artículos:

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

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

  constructor() { }

  retornar() {
    return [
      {
        codigo: 1,
        descripcion: "papas",
        precio: 12.33
      },
      {
        codigo: 2,
        descripcion: "manzanas",
        precio: 54
      },
      {
        codigo: 3,
        descripcion: "sandía",
        precio: 14
      }
    ];
  }
}
```

El decorador `@Injectable()` será de suma importancia para poder acceder a esta clase desde la componente.

3. Ahora veremos como consumimos el servicio desde nuestra componente. Procedemos a modificar la componente que se crea por defecto 'AppComponent' que tiene por responsabilidad mostrar en la página el listado de artículos:

```

import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ArticulosService } from '../articulos.service';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  articulos :any;

  constructor(private articulosServicio: ArticulosService) {
    this.articulos=this.articulosServicio.retornar();
  }
}

```

Primero importamos el servicio llamado ArticulosService que se almacena en el archivo 'articulos.service.ts':

```
import { ArticulosService } from '../articulos.service';
```

Para inyectar el objeto de la clase 'ArticulosService' que crea Angular en forma automática lo hacemos en el parámetro del constructor::

```

constructor(private articulosServicio: ArticulosService) {
  this.articulos=this.articulosServicio.retornar();
}

```

Esta asignación dispara la actualización de la página HTML.

4. Falta que codifiquemos la vista con los datos recuperados:

app.component.html

```

<table>
  @for(articulo of articulos;track $index) {
    <tr>
      <td>{{articulo.codigo}}</td>
      <td>{{articulo.descripcion}}</td>
      <td>{{articulo.precio}}</td>
    </tr>
  }
</table>
<router-outlet />

```

5. Por último la hoja de estilo de la componente:

app.component.css

```

/* Estilo para la tabla */
table {
  border-collapse: collapse;
  width: 100%;
}

/* Estilo para las celdas de la tabla */
td {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
}

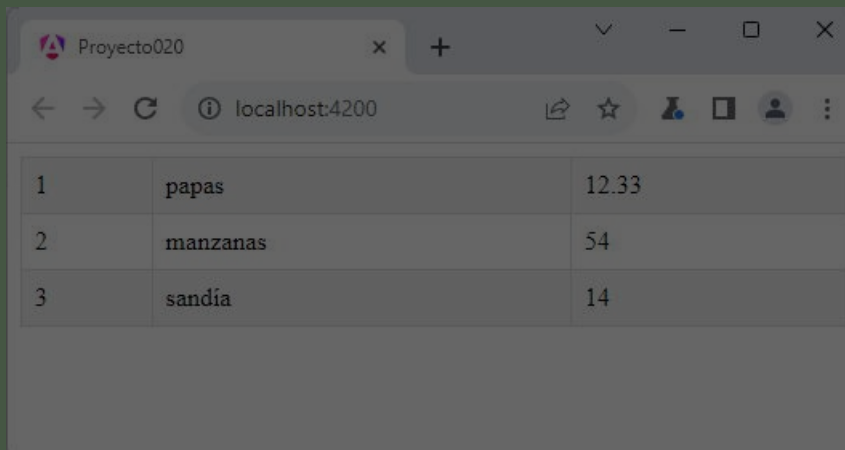
/* Estilo para las filas impares */
tr:nth-child(odd) {
  background-color: #f2f2f2;
}

/* Estilo para las filas al pasar el ratón sobre ellas */
tr:hover {
  background-color: #e6e6e6;
}

```

6. Si ejecutamos ahora el proyecto020 veremos en el navegador el listado de artículos:

```
ng server -o
```



1	papas	12.33
2	manzanas	54
3	sandía	14

Podemos probar esta aplicación en la web [aquí](#).

Al principio y con problemas muy sencillos parece que solo agregamos complejidad a nuestra aplicación.

Veremos que esta forma de desacoplar el acceso a datos de las componentes y delegarla en otras clases llamadas servicios facilita el mantenimiento de nuestras aplicaciones.

También hay que hacer notar que la forma de consumir dichas clases se hace por medio del patrón de inyección de dependencias.

Retornar

