

Caso Agregar con Angular – parte 3



Implementación del método para agregar producto en Angular

En esta tercera parte, se implementó el método para agregar un producto desde el frontend de **Angular**. Se realizaron modificaciones en el servicio, el componente y el formulario. A continuación, se explica cada cambio y su propósito.

Paso 1: Modificación del servicio `producto.service.ts`

Código actualizado

```
import {HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Producto } from './producto';

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

```
private urlBase = "http://localhost:8080/inventario-app/productos";

private clienteHttp = inject(HttpClient);

obtenerProductosLista(): Observable<Producto[]>{
    return this.clienteHttp.get<Producto[]>(this.urlBase);
}

agregarProducto(producto: Producto): Observable<Object>{
    return this.clienteHttp.post(this.urlBase, producto);
}

}
```

Explicación de los cambios

1. Se agregó el método `agregarProducto()`:
 - Usa `HttpClient.post()` para enviar un objeto `Producto` al backend.
 - Devuelve un `Observable<Object>`, lo que permite manejar la respuesta de la API en el componente.
2. Uso de `inject(HttpClient)` en lugar de `constructor(private http: HttpClient):`
 - Esta es la nueva forma recomendada en **Angular 19** para la inyección de dependencias en **standalone components**.

Paso 2: Implementación del formulario en `agregar-producto.component.html`

Código actualizado

```
<div class="container">

    <div class="container text-center" style="margin: 30px">
        <h3>Agregar Producto</h3>
    </div>

    <form (ngSubmit)="onSubmit()">
        <div class="mb-3">
            <label for="descripcion"
                class="form-label">Descripción del Producto</label>
            <input type="text" class="form-control">
        </div>
    </form>

```

```
        id="descripcion" name="descripcion" required="true"
        [(ngModel)]="producto.descripcion">
    </div>
    <div class="mb-3">
        <label for="precio" class="form-label">Precio</label>
        <input type="number" step="any" class="form-control"
            id="precio" name="precio" [(ngModel)]="producto.precio">
    </div>
    <div class="mb-3">
        <label for="existencia" class="form-label">Existencia</label>
        <input type="number" class="form-control"
            id="existencia" name="existencia"
            [(ngModel)]="producto.existencia">
    </div>
    <div class="text-center">
        <button type="submit"
            class="btn btn-warning btn-sm me-3">Agregar</button>
        <a href="/" class="btn btn-danger btn-sm">Regresar</a>
    </div>
</form>

</div>
```

Explicación de los cambios y estilos Bootstrap

1. Uso de `ngSubmit` en el formulario

- Se agregó el evento `(ngSubmit)="onSubmit()"` en la etiqueta `<form>`.
- Evita que el navegador recargue la página y en su lugar ejecuta el método `onSubmit()` en el componente.

2. Two-Way Data Binding con `ngModel`

- `[(ngModel)]="producto.descripcion"`
- `[(ngModel)]="producto.precio"`
- `[(ngModel)]="producto.existencia"`
- Vincula cada input con la propiedad del objeto `producto` en el componente, permitiendo que los cambios en el formulario se reflejen directamente en la variable.

3. Uso de estilos de Bootstrap:

- `class="form-control"`: Aplica un estilo de campo de entrada mejorado.
- `class="mb-3"`: Agrega separación entre los campos del formulario.
- `class="btn btn-warning btn-sm me-3"`:
 - `btn-warning`: Aplica el color amarillo de Bootstrap.
 - `btn-sm`: Hace el botón más pequeño.
 - `me-3`: Agrega un margen derecho para separación.

Paso 3: Modificación del componente `agregar- producto.component.ts`

Código actualizado

```
import { Component, inject } from '@angular/core';
import { Router } from '@angular/router';
import { Producto } from '../producto';
import { ProductoService } from '../producto.service';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-agregar-producto',
  imports: [FormsModule],
  templateUrl: './agregar-producto.component.html'
})
export class AgregarProductoComponent {
  producto: Producto = new Producto();

  private productoServicio = inject(ProductoService);
  private enrutador = inject(Router);

  onSubmit() {
    this.guardarProducto();
  }

  guardarProducto() {
    this.productoServicio.agregarProducto(this.producto).subscribe(
      {
        next: (datos) => {
          this.irListaProductos();
        },
        error: (error: any) => { console.log(error) }
      }
    );
  }

  irListaProductos() {
    this.enrutador.navigate(['/productos']);
  }
}
```

Explicación de los cambios

1. Importación de `FormsModule`:

```
imports: [FormsModule]
```

- Es necesario para habilitar **Two-Way Binding** (`[ngModel]`) en Angular standalone.

2. Definición de `producto` como un objeto vacío de tipo `Producto`

```
producto: Producto = new Producto();
```

- Permite que `ngModel` vincule los valores correctamente en el formulario.

3. Método `guardarProducto()`:

- Llama a `productoServicio.agregarProducto(this.producto)`, enviando los datos al backend.
- Se usa `subscribe()` para manejar la respuesta y navegar de regreso a la lista de productos.

4. Método `irListaProductos()`:

```
irListaProductos() {
  this.enrutador.navigate(['/productos']);
}
```

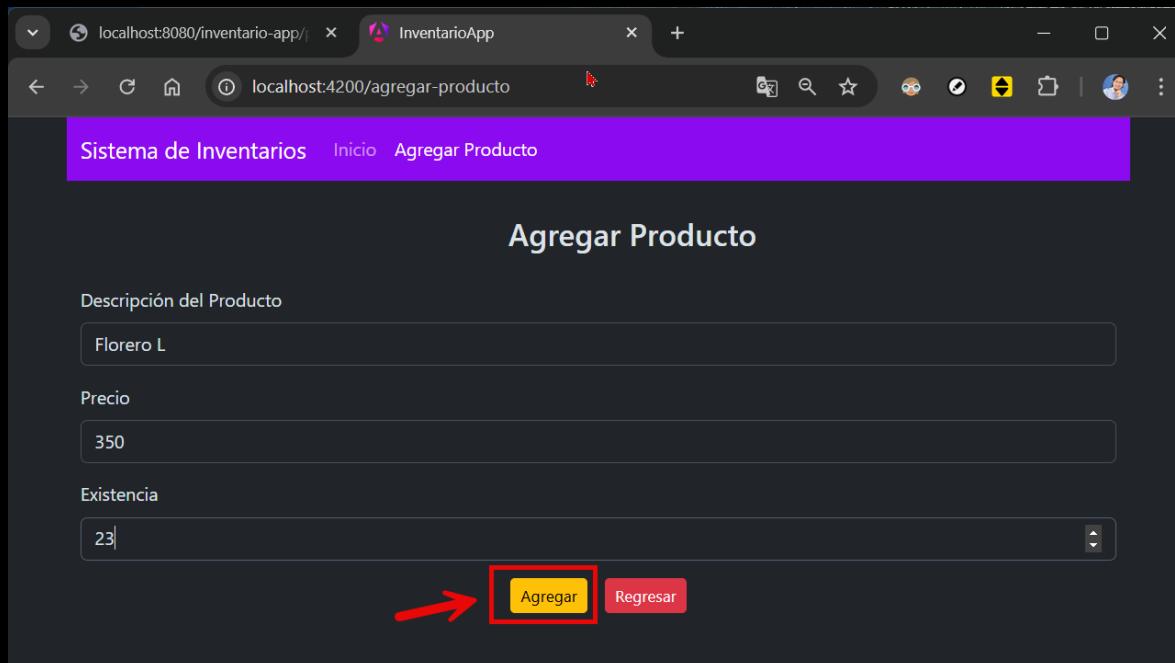
- Redirige automáticamente a la lista de productos después de agregar uno nuevo.

Paso 4: Eliminación del archivo CSS

- Se eliminó el archivo `agregar-producto.component.css` porque:
 - Se están utilizando estilos de Bootstrap en lugar de estilos personalizados.
 - Esto mantiene el código más limpio y evita la sobrecarga de archivos innecesarios.

Resultado Final

- Ejecutamos el Front End con Angular



Sistema de Inventarios Inicio Agregar Producto

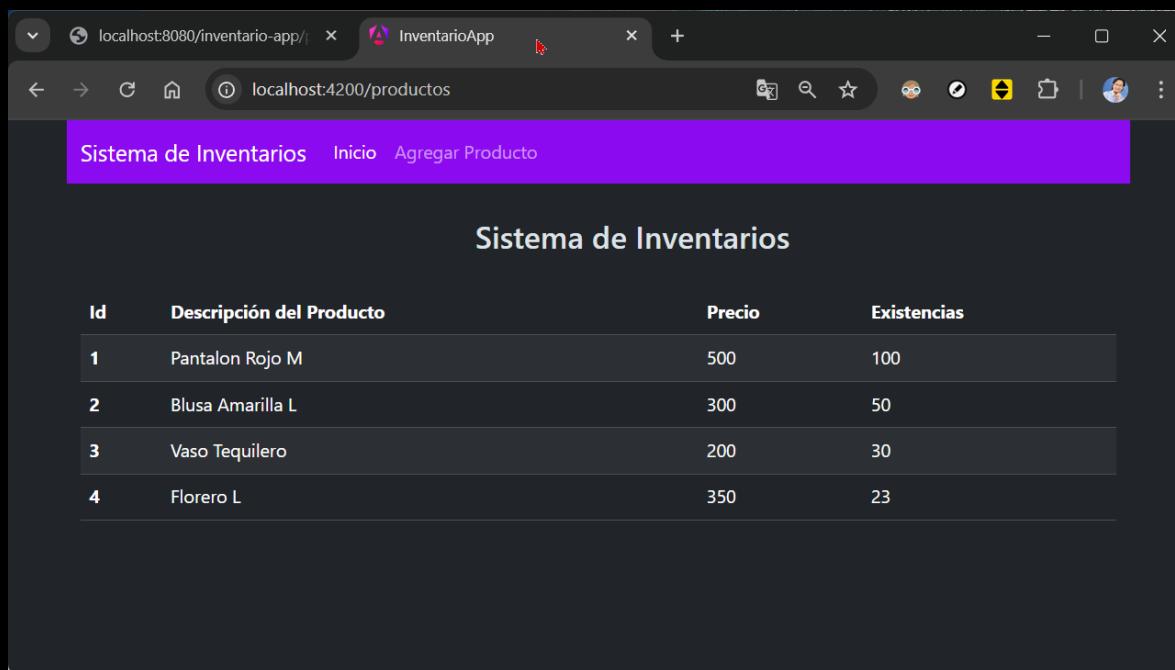
Agregar Producto

Descripción del Producto
Florero L

Precio
350

Existencia
23

Agregar Regresar

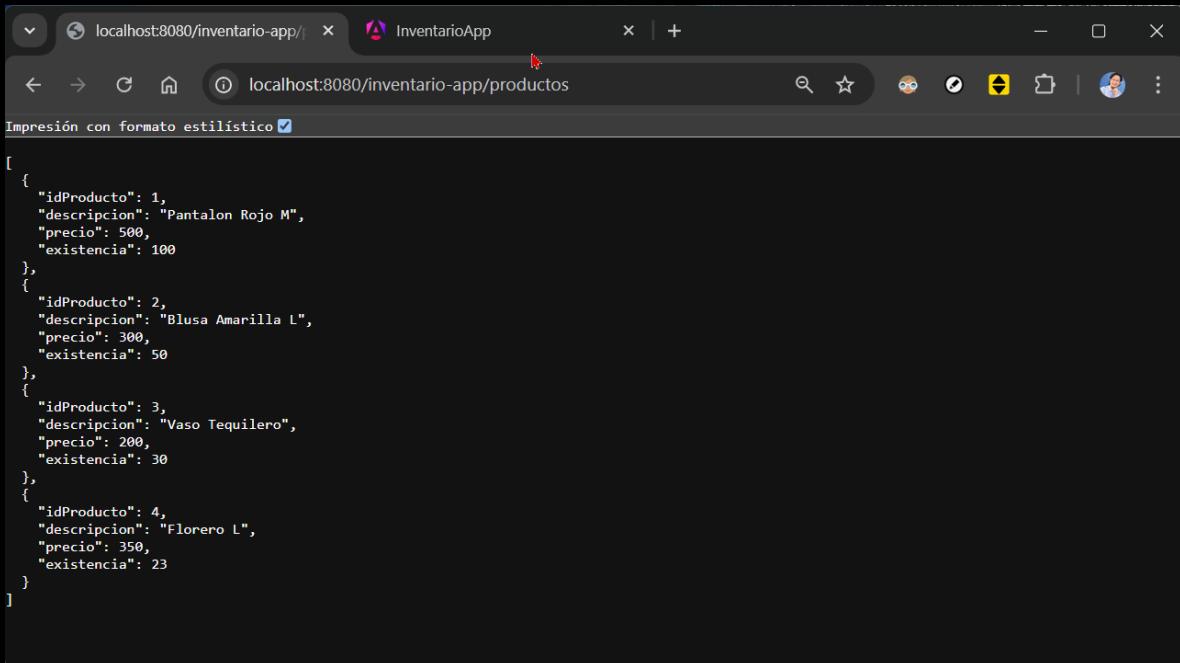


Sistema de Inventarios Inicio Agregar Producto

Sistema de Inventarios

Id	Descripción del Producto	Precio	Existencias
1	Pantalon Rojo M	500	100
2	Blusa Amarilla L	300	50
3	Vaso Tequileros	200	30
4	Florero L	350	23

- Ejecutamos el BackEnd con Spring Boot y confirmamos que se agregó el producto



```
[{"idProducto": 1, "descripcion": "Pantalon Rojo M", "precio": 500, "existencia": 100}, {"idProducto": 2, "descripcion": "Blusa Amarilla L", "precio": 300, "existencia": 50}, {"idProducto": 3, "descripcion": "Vaso Tequilero", "precio": 200, "existencia": 30}, {"idProducto": 4, "descripcion": "Florero L", "precio": 350, "existencia": 23}]
```

Conclusión

- Se agregó el método `agregarProducto()` en el servicio `ProductoService` para enviar datos al backend.
- Se implementó un formulario con Bootstrap, usando `form-control`, `mb-3`, `btn-sm`, y `btn-warning`.
- Se usó `ngModel` para Two-Way Binding, permitiendo que el formulario esté sincronizado con el objeto `producto`.
- Se importó `FormsModule` en el componente standalone para habilitar `ngModel`.
- Después de agregar un producto, se redirige automáticamente a la lista.

Con estos cambios, la aplicación ya permite **agregar productos desde Angular y enviarlos al backend de Spring Boot**. 

Saludos!

Ing. Ubaldo Acosta

Fundador de [GlobalMentoring.com.mx](https://www.globalmentoring.com.mx)