

Para este caso vamos a hacer un API desde angular pero las vistas van a ser con prime engine:

<https://primeng.org/>
<https://www.primefaces.org/primeng-v15-lts/installation>

Vamos a descargar un proyecto (sakai) que ya tiene todas las validaciones y ejemplos correspondientes para usar angular & primeng

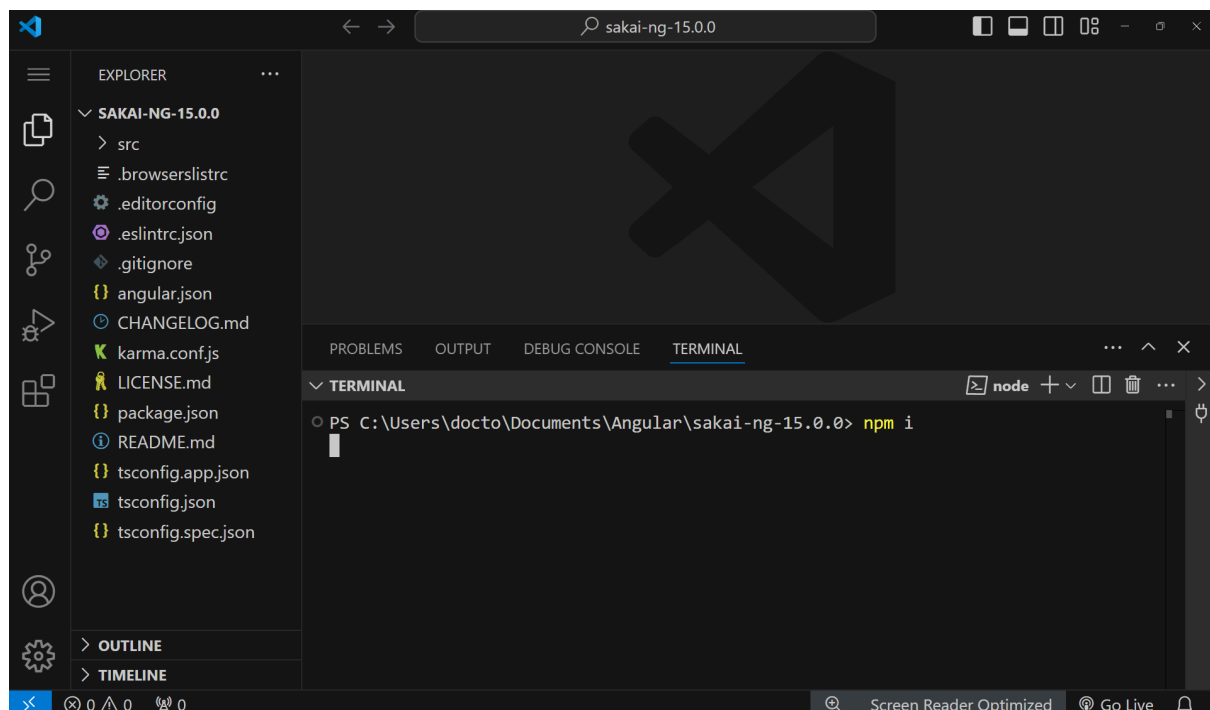
<https://github.com/primefaces/sakai-ng/tags>

Antes de esto vamos a instalar NodeJs y Luego AngularCLI y podemos verificar las versiones con los siguientes comandos:

- node -v
- ng version

Para este ejemplo vamos a descargar la versión 15

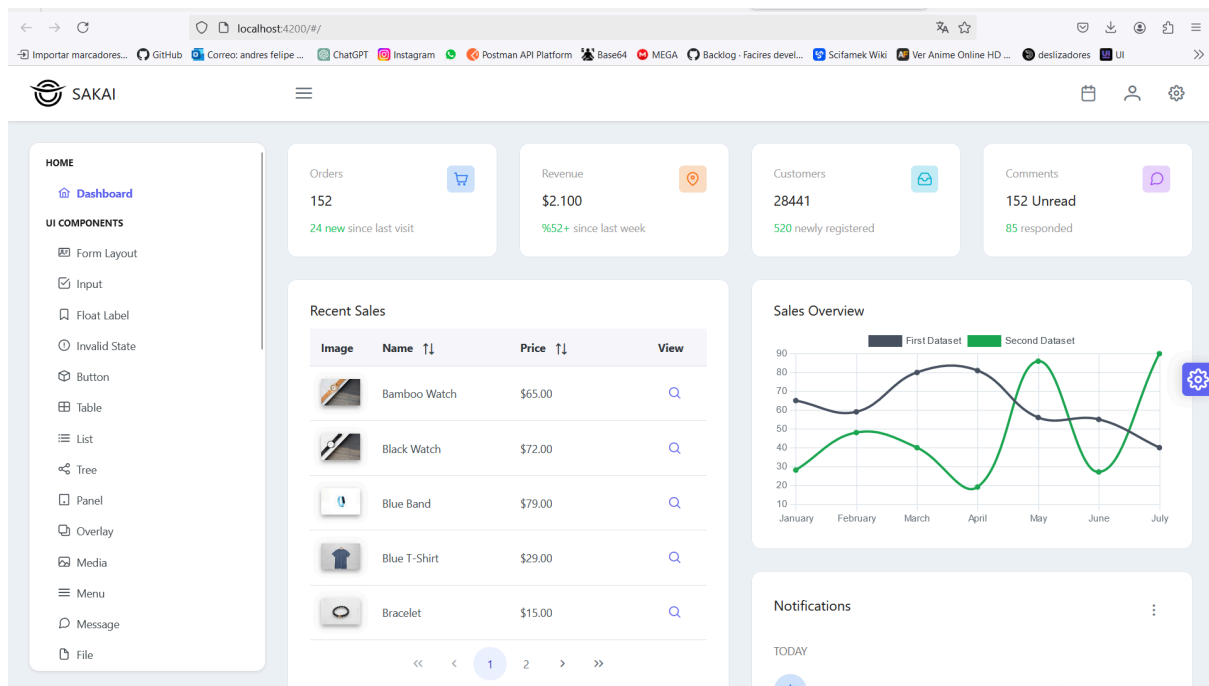
Una vez descargado y descomprimido vamos a abrir en visual y luego a hacer un “npm i”



Esto va a tardar... todo depende de la velocidad del internet y de la máquina.

Para correr el proyecto usamos:

ng serve -o



Antes de iniciar el proyecto vamos a borrar todo lo que no necesitamos.

src\app\app-routing.module.ts

```
src > app > TS app-routing.module.ts > AppRoutingModule
1 import { RouterModule } from '@angular/router';
2 import { NgModule } from '@angular/core';
3 import { NotFoundComponent } from './demo/components/notfound/notfound.component';
4 import { AppLayoutComponent } from './layout/app.layout.component';
5
6 @NgModule({
7   imports: [
8     RouterModule.forRoot([
9       {
10        path: '', component: AppLayoutComponent,
11        children: [
12          { path: '', loadChildren: () => import('./demo/components/dashboard/dashboard.module').then(m => m.DashboardModule) },
13          { path: 'uikit', loadChildren: () => import('./demo/components/uikit/uikit.module').then(m => m.UikitModule) },
14          { path: 'utilities', loadChildren: () => import('./demo/components/utilities/utilities.module').then(m => m.UtilitiesModule) },
15          { path: 'documentation', loadChildren: () => import('./demo/components/documentation/documentation.module').then(m => m.DocumentationModule) },
16          { path: 'blocks', loadChildren: () => import('./demo/components/primeblocks/primeblocks.module').then(m => m.PrimeBlocksModule) },
17          { path: 'pages', loadChildren: () => import('./demo/components/pages/pages.module').then(m => m.PagesModule) },
18        ],
19      },
20      { path: 'auth', loadChildren: () => import('./demo/components/auth/auth.module').then(m => m.AuthModule) },
21      { path: 'landing', loadChildren: () => import('./demo/components/landing/landing.module').then(m => m.LandingModule) },
22      { path: 'notfound', component: NotFoundComponent },
23      { path: '**', redirectTo: '/notfound' },
24    ], { scrollPositionRestoration: 'enabled', anchorScrolling: 'enabled', onSameUrlNavigation: 'reload' })
25   ],
26   exports: [RouterModule]
27 })
28 export class AppRoutingModule {
29 }
30
```

Y al quitar todo lo que no necesitamos nos debe de quedar de la siguiente manera:

src\app\app-routing.module.ts

```

import { RouterModule } from '@angular/router';
import { NgModule } from '@angular/core';
import { NotFoundComponent } from '../demo/components/notfound/notfound.component';
import { AppLayoutComponent } from '../layout/app.layout.component';

@NgModule({
  imports: [
    RouterModule.forRoot([
      { path: 'notfound', component: NotFoundComponent },
      { path: '**', redirectTo: '/notfound' },
    ]), { scrollPositionRestoration: 'enabled', anchorScrolling: 'enabled', onSameUrlNavigation: 'reload' })
  ],
  exports: [RouterModule]
})
export class AppRoutingModule {
}

```

Y cuando compilamos “ng s” nos debe de salir lo siguiente:

```

TERMINAL
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> ng s
* Generating browser application bundles (phase: setup)... TypeScript compiler options "target" and "useDefineForClassFields" are set to "ES2022" and "false" respectively by the Angular CLI. To control ECMA version and features use the Browserslist configuration. For more information, see https://angular.io/guide/build#configuring-browser-compatibility
NOTE: You can set the "target" to "ES2022" in the project's tsconfig to remove this warning.
✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Raw Size
vendor.js           | vendor | 2.80 MB
styles.css, styles.js | styles | 678.15 kB
polyfills.js        | polyfills | 316.43 kB
main.js             | main | 120.59 kB
runtime.js          | runtime | 6.51 kB
| Initial Total | 3.90 MB

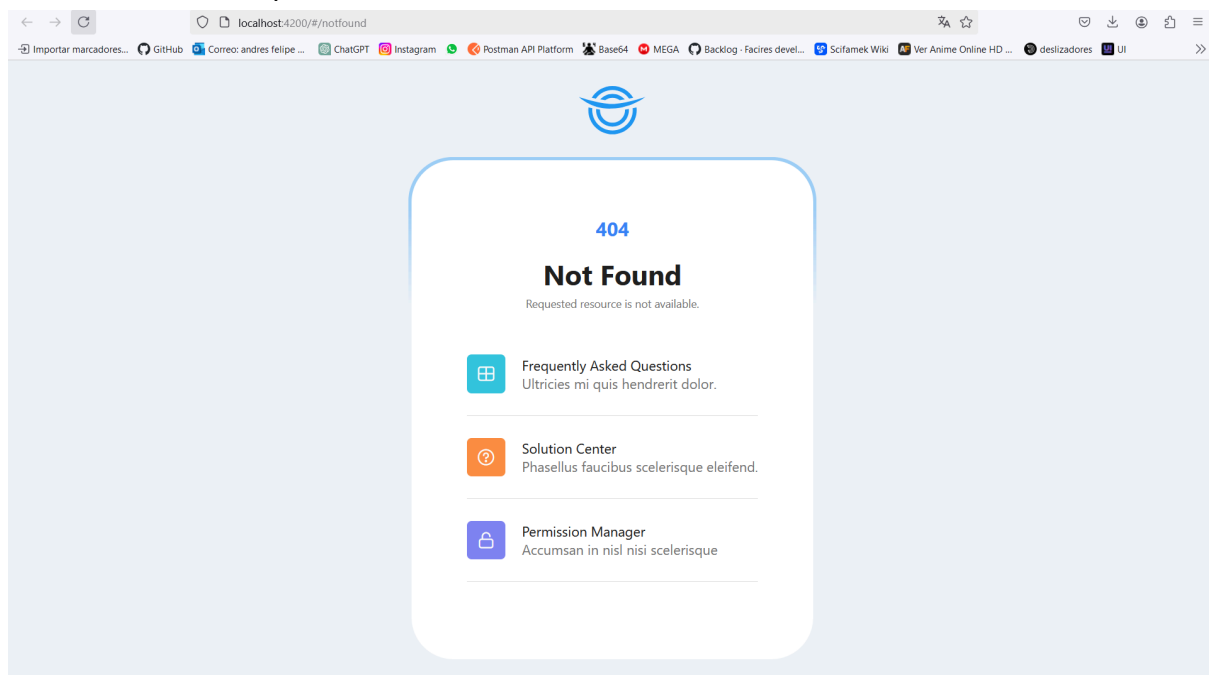
Build at: 2024-12-01T22:00:36.940Z - Hash: 05397ad2d215f0e0 - Time: 3471ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.

```

Y nos debe de salir que no se encontró la ruta.

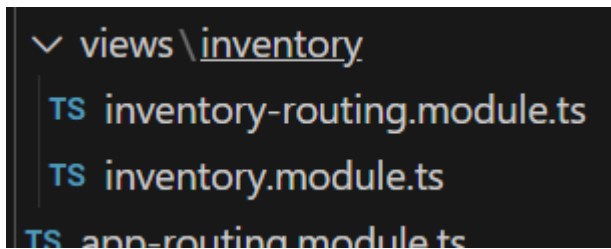


Ahora vamos a crear un módulo de inventarios para poder visualizar productos y realizar las operaciones CRUD.

Por ello vamos a crear una carpeta que se llame views en la ruta "src\app\views"

Y luego vamos a crear un módulo de inventario:

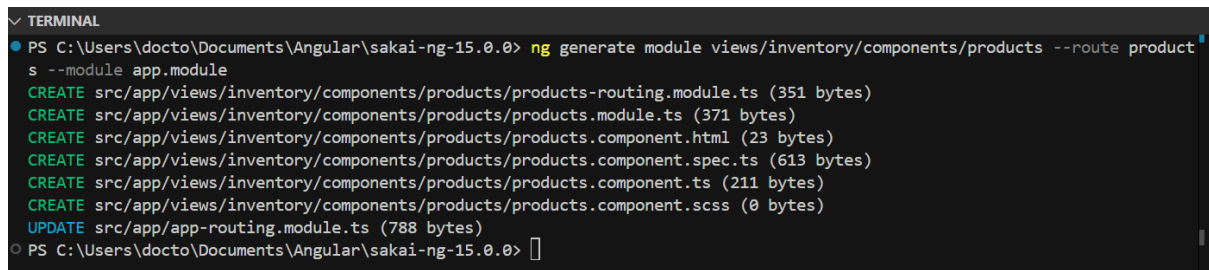
ng generate module views/inventory --routing



Y luego vamos a crear una carpeta de componentes "src\app\views\inventory\components" y allí vamos a poner nuestras vistas para hacer el CRUD.

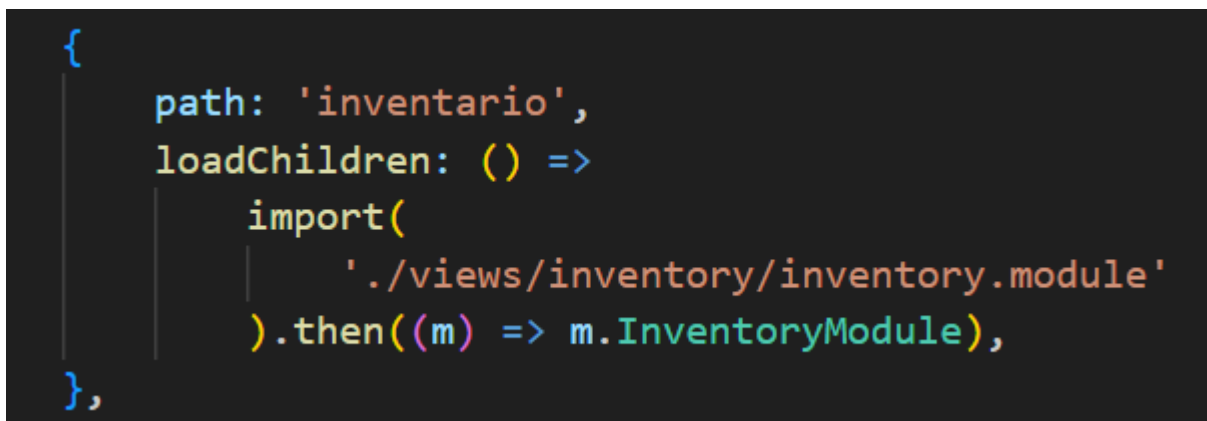
1 - Crear el módulo de productos para carga perezosa:

ng generate module views/inventory/components/products --route products --module app.module



Y si vamos a nuestro archivo de las rutas del proyecto de angular

"src\app\app-routing.module.ts" podremos ver que el anterior comando nos agregó la ruta pero hay que modificarla por que la gracia es que se pueda acceder desde inventarios osea <http://localhost:4200/#/inventario/productos>



Y ahora tenemos que ir a inventario para agregar la ruta de productos:

src\app\views\inventory\inventory-routing.module.ts

```
const routes: Routes = [
  {
    path: 'productos',
    loadChildren: () =>
      import(
        './components/products/products.module'
      ).then((m) => m.ProductsModule),
  },
];
```

Esto lo hicimos así para que tanto productos como inventario puedan tener hijos.

SHARED MODULE

Ahora tenemos que pensar en que nuestra aplicación tiene que empezar a importar materiales desde primeng para construir las vistas y no es buena práctica importar muchas cosas en diferentes .ts por ello lo que vamos a hacer es crear una carpeta que se va a llamar “shared” y luego crear allí un shared.module.ts para usar todos nuestros import:

- 1 - Crear la carpeta “src\app\shared”
- 2 - Crear el módulo ng generate module shared

```
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> ng generate module shared
CREATE src/app/shared/shared.module.ts (192 bytes)
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> █
```

CREACIÓN DE LA VISTA PRODUCTOS

ahora vamos a empezar a crear nuestra tabla para renderizar los productos

Vamos a usar las plantillas de primeng

<https://primeng.org/table#accessibility> y yo escogí la table module.

Lo primero que tenemos que hacer es agregarla en el shared module y dejarla lista para exportar.

```
app > shared > ts shared.module.ts > ...
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { TableModule } from 'primeng/table';
💡

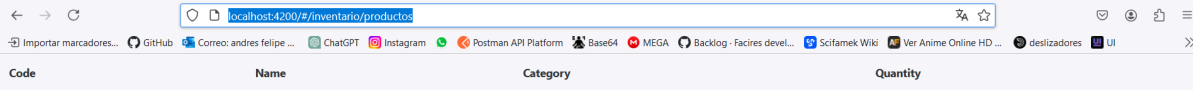
@NgModule({
  declarations: [],
  imports: [
    CommonModule
  ],
  exports: [
    TableModule
  ]
})
export class SharedModule { }
```

Lo segundo que tenemos que hacer es ir a nuestro módulo de productos “src\app\views\inventory\components\products\products.module.ts” e importar el shared module

```
import { ProductsComponent } from '../products.component';
import { SharedModule } from '../../shared/shared.module';
💡

@NgModule({
  > declarations: [ ...
  ],
  imports: [
    CommonModule,
    ProductsRoutingModule,
    SharedModule
  ]
})
```

Y luego vamos a copiar y pegar el código de la tabla (Advertencia: cambiar el value a un arreglo vacío) y si ejecutamos nuestro proyecto con el comando: “ng s” podremos ver nuestra tabla vacía:

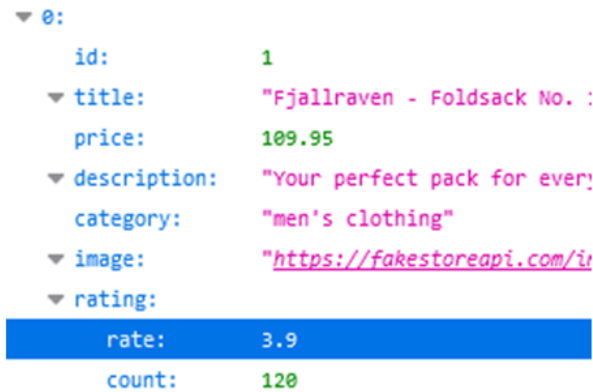


| Code | Name | Category | Quantity |
|------|------|----------|----------|
|------|------|----------|----------|

Creación de la interface producto

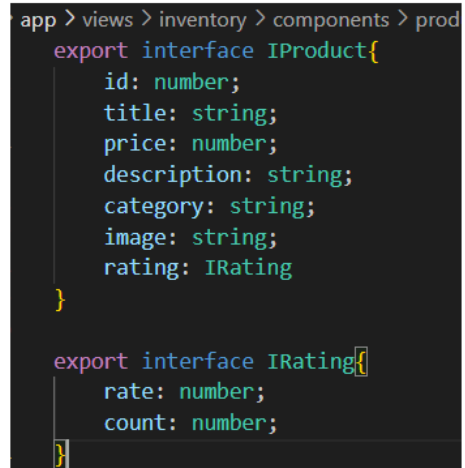
Una interfaz es lo que nos ayudará a definir los atributos y métodos de un Objeto. Es simplemente un contrato de atributos y métodos.

para ello tenemos que crear una carpeta que se llame models en “src\app\views\inventory\components\products\models” y allí dentro vamos a crear nuestra interfaz de producto src\app\views\inventory\components\products\models\product.ts



A screenshot of a JSON object representing a product. The object has the following structure:

| | |
|--------------|--|
| 0: | |
| id: | 1 |
| title: | "Fjallraven - Foldsack No. 1" |
| price: | 109.95 |
| description: | "Your perfect pack for everyday use and travel." |
| category: | "men's clothing" |
| image: | "https://fakestoreapi.com/img/61mp5g64UOL.jpg" |
| rating: | |
| rate: | 3.9 |
| count: | 120 |



```
app > views > inventory > components > prod
export interface IProduct{
  id: number;
  title: string;
  price: number;
  description: string;
  category: string;
  image: string;
  rating: IRating;
}

export interface IRating{
  rate: number;
  count: number;
}
```

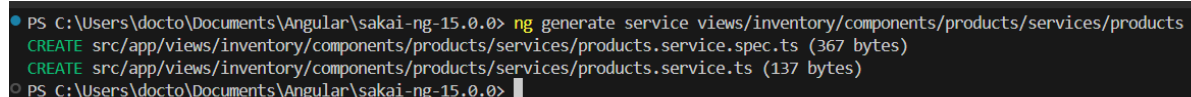
Y luego para usar esa interfaz basta solo con importarla.

Consumo de API

Ahora lo que vamos a hacer para rellenar nuestros productos es usar la pagina <https://fakestoreapi.com/> exactamente en: <https://fakestoreapi.com/products>

Para poder consumir una API tenemos que respetar los patrones arquitectónicos por ello tenemos que crear un carpeta que se llame services en la ruta: “src\app\views\inventory\components\products\services” y eso lo hacemos con el siguiente comando:

ng generate service views/inventory/components/products/services/products



```
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> ng generate service views/inventory/components/products/services/products
CREATE src/app/views/inventory/components/products/services/products.service.spec.ts (367 bytes)
CREATE src/app/views/inventory/components/products/services/products.service.ts (137 bytes)
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0>
```

Y nos crea un servicio inyectable:

```

src > app > views > inventory > components > products > services > TS products.ser
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class ProductsService {
7
8    constructor() { }
9  }
10

```

Aquí es donde vamos a generar toda la lógica para consumir el API

1 -> Angular ya trae una librería inyectable que sirve para comunicarse con API se llama "HttpClient" y lo único que tenemos que hacer es importar e inyectar en el constructor.

```

import { IProduct } from '../models/Iproduct';
import { HttpClient } from '@angular/common/http';
💡
@Injectable({
  providedIn: 'root'
})
export class ProductsService {

  constructor(private httpClient: HttpClient) { }

```

Y ahora vamos a crear un método para consumir el API y luego visualizarla:

1 - Declarar el URL de la API y luego un arreglo para guardar los elementos de la API

```

private readonly urlAPI: string = "https://fakestoreapi.com/products";
public arrProduct: Array<IProduct> = [];

```

2 - Crear un método con un get y una suscripción para consumir el API


```

getAll(): Array<IProduct>{
  this.httpClient.get(this.urlAPI).subscribe(
    (data: any) => {
      this.arrProduct = data;
    }
  );

  return [];
}

```

Para poder utilizar ese servicio tenemos que inyectarla en el constructor del componente de los productos:

```

import { IProduct } from '../models/Iproduct';
import { ProductsService } from '../services/products.service';

@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.scss']
})
export class ProductsComponent {
  arrProduct: Array<IProduct> = [];

  constructor(private productsService: ProductsService){}

```

Luego tenemos que hacer que el servicio de consumo se llame en el método onInit y crear un getData para traer la lista desde el servicio:

```

getData(){
  return this.productsService.arrProduct;
}

viewINFO(product: IProduct){
  alert(product.description);
}

```

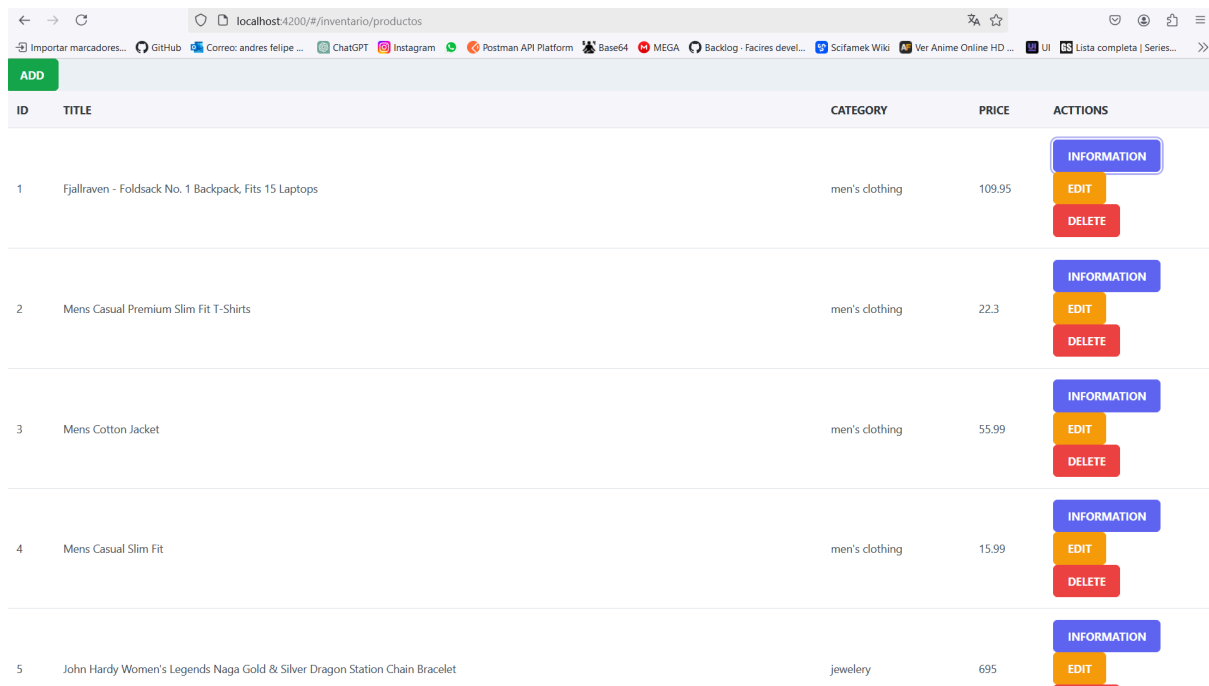
Y por último en la lista de primeng ponemos e getData

```

<p-table [value]="getData()"

```

Ahora vamos a darle estilo y poner botones para lograr la siguiente vista:



| ID | TITLE | CATEGORY | PRICE | ACTIONS |
|----|---|----------------|--------|--|
| 1 | Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops | men's clothing | 109.95 | <div>INFORMATION</div> <div>EDIT</div> <div>DELETE</div> |
| 2 | Mens Casual Premium Slim Fit T-Shirts | men's clothing | 22.3 | <div>INFORMATION</div> <div>EDIT</div> <div>DELETE</div> |
| 3 | Mens Cotton Jacket | men's clothing | 55.99 | <div>INFORMATION</div> <div>EDIT</div> <div>DELETE</div> |
| 4 | Mens Casual Slim Fit | men's clothing | 15.99 | <div>INFORMATION</div> <div>EDIT</div> <div>DELETE</div> |
| 5 | John Hardy Women's Legends Naga Gold & Silver Dragon Station Chain Bracelet | jewelery | 695 | <div>INFORMATION</div> <div>EDIT</div> |

BORRAR UN PRODUCTO

Solo tenemos que hacer un `httpClient.delete` con un `subscribe`:

```
delete(product: IProduct): void{
  const delURL = `https://fakestoreapi.com/products/${product.id}`;

  this.httpClient.delete(delURL).subscribe((data:any)=>{
    this.arrProduct = data;
  });

  alert(`Delete product: ${product.id}`);
}
```

Y pues obviamente el servidor no puede modificar una API pública, lo único que podemos hacer es mirar las peticiones de red y ver un delete con código de respuesta sobre 200.

CREAR

Vamos a crear una ventana modal para que se muestre si queremos editar o modificar.

Vamos a crear un componente para agregar/modificar

Paso 1 crear una carpeta components dentro de productos en la ruta
“src\app\views\inventory\components\products\components”

y lo hacemos con el siguiente comando:

ng generate component views/inventory/components/products/components/product

```
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> ng generate component views/inventory/components/products/components/product
CREATE src/app/views/inventory/components/products/components/product/product.component.html (22 bytes)
CREATE src/app/views/inventory/components/products/components/product/product.component.spec.ts (606 bytes)
CREATE src/app/views/inventory/components/products/components/product/product.component.ts (207 bytes)
CREATE src/app/views/inventory/components/products/components/product/product.component.scss (0 bytes)
UPDATE src/app/views/inventory/components/products/products.module.ts (550 bytes)
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0>
```

Para poder llamarlo y abrirlo como una ventana emergente usamos el servicio de primeng y para poder inyectar tenemos que declararlo como proveedor en:

src\app\views\inventory\components\products\products.module.ts

```
import { DialogModule } from 'primeng/dialog';
import { DialogService, DynamicDialogModule } from 'primeng/dynamicdialog';

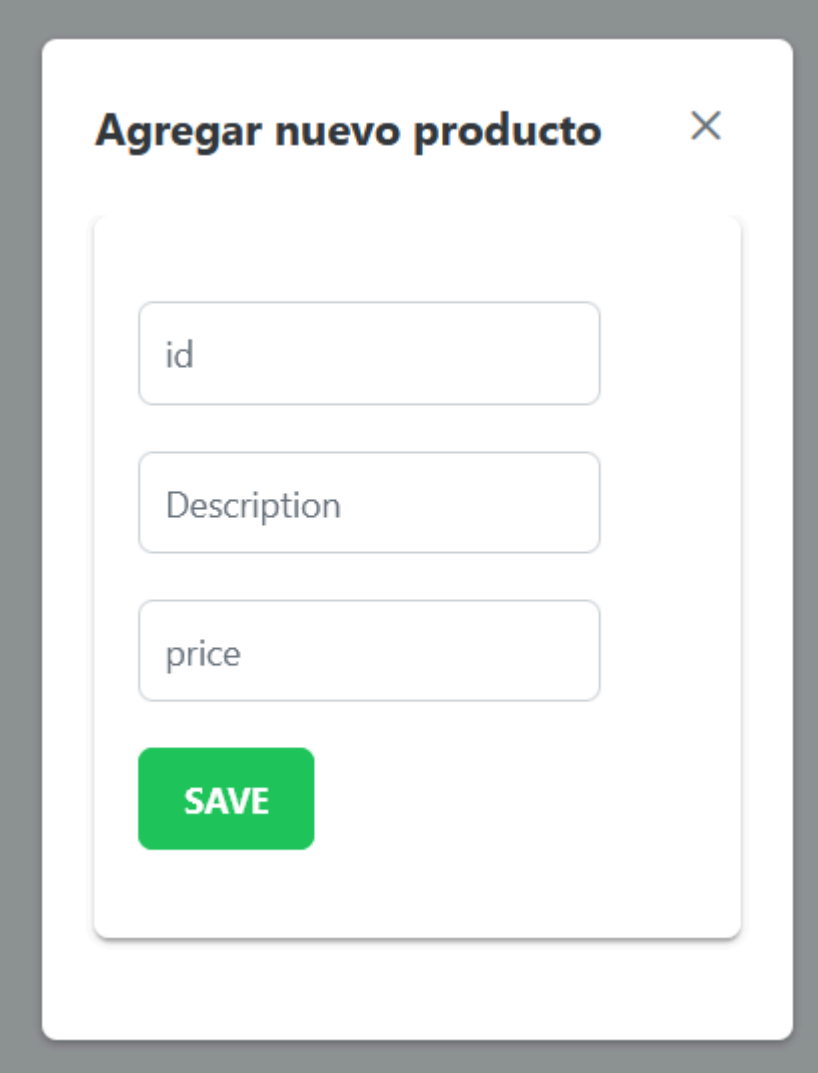
@NgModule({
  declarations: [
    ProductsComponent,
    ProductComponent
  ],
  imports: [
    CommonModule,
    ProductsRoutingModule,
    SharedModule,
    DialogModule,
    DynamicDialogModule
  ],
  providers: [
    DialogService
  ]
})
```

Y con esto ya podemos inyectar para tener ventanas emergentes:

```
export class ProductsComponent {
  constructor(private productService: ProductService, private dialogService: DialogService){}
```

```
this.dialogService.open(ProductComponent, {  
  header: displayedText,  
  height: '400px',  
  width: '300px'  
});
```

Y luego vamos a programar la siguiente ventanita



The image shows a modal dialog box with a white background and a gray border. At the top, it has the title "Agregar nuevo producto" in bold black text, followed by a close button (X). Below the title, there is a large white rounded rectangle containing three input fields. The first field is labeled "id", the second "Description", and the third "price". At the bottom of this rounded rectangle is a green button with the text "SAVE" in white capital letters.

Ahora bien para poder guardar estos elementos necesitamos usar los formularios de angular

Y para esto necesitamos declarar e importar en 2 archivos:

src\app\views\inventory\components\products\services\products.service.ts

src/app/views/inventory/components/products/components/product/product.component.ts

```
1 import { Component } from '@angular/core';
2 import { FormGroup } from '@angular/forms';
3
4 @Component({
5   selector: 'app-product',
6   templateUrl: './product.component.html',
7   styleUrls: ['./product.component.scss']
8 })
9 export class ProductComponent {
10   ⚡
11   public form: FormGroup = {};
12 }
```

Ahora bien nosotros tenemos que especificar mediante una clase cuáles son los atributos del formulario por ello creamos una carpeta llamada class en la ruta:

“src/app/views/inventory/components/products/class”

con el siguiente comando:

ng generate class views/inventory/components/products/class/product-repository

```
▼ TERMINAL
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> ng generate class views/inventory/components/products/class/product-repository
CREATE src/app/views/inventory/components/products/class/product-repository.spec.ts (199 bytes)
CREATE src/app/views/inventory/components/products/class/product-repository.ts (35 bytes)
PS C:\Users\docto\Documents\Angular\sakai-ng-15.0.0> █
```

En el repositorio vamos a crear los 2 métodos para si queremos crear un formulario y editar producto o crear producto:

```

c > app > views > inventory > components > products > class > TS product-repository.ts > ...
1  import { FormBuilder, FormControl, FormGroup } from "@angular/forms";
2  import { IProduct } from "../models/Iproduct";
3
4  export class ProductRepository {
5
6      constructor(){}
7
8
9      new(): FormGroup{
10         return new FormBuilder().group(
11             {
12                 title: new FormControl(),
13                 description: new FormControl(),
14                 price: new FormControl()
15             }
16         );
17     }
18
19
20     edit(product: IProduct){
21         return new FormBuilder().group(
22             {
23                 title: new FormControl(product.title),
24                 description: new FormControl(product.description),
25                 price: new FormControl(product.price)
26             }
27         );
28     }
29
30 }
31

```

Ahora tenemos que ir a nuestro product Component para realizar 3 inyecciones de dependencia:

- ProductoServicio:
src\app\views\inventory\components\products\services\products.service.ts
- Y los módulos de prime engine dynamicDialogConfig y dynamicDialogRef para poder cerrar la ventana:

```

    })
    export class ProductComponent {

        constructor(
            private productService: ProductService,
            private dynamicDialogConfig: DynamicDialogConfig,
            private dynamicDialogRef: DynamicDialogRef,
        ){

        }

    }

```

Ahora necesitamos asociar la vista con un formulario y un método para guardar:

src\app\views\inventory\components\products\components\product\product.component.html

```

> app > views > inventory > components > products > componen
1  <p-card>
2  >   <form [formGroup]="getForm()"> ...
9   </form>
0   </p-card>
1   |

```

src\app\views\inventory\components\products\components\product\product.component.ts

```

getForm(){
    return this.productService.form;
}

```

Y dentro del formulario creamos un save:

```

save(){
  console.log(this.productsService.form.value);

  const form: FormGroup = this.productsService.form;
  if (form.valid){
    const product: IProduct = this.productsService.form.value as IProduct;
    this.productsService.create(product);
    alert("Producto guardado con éxito.");
  }else{
    alert("Error al guardar el producto.");
  }

  this.closeModalWindow();
}

```

Y luego como nosotros tenemos el servicio inyectado podemos usarlo para guardar en el API

src\app\views\inventory\components\products\services\products.service.ts

```

create(product: IProduct): void {
  const postURL = "https://fakestoreapi.com/products";

  this.httpClient.post(postURL, product).subscribe(
    (data) => {
      alert("SAVE IN API");
      console.log(data);
    }
  );
}

```

Y obviamente el servidor no va a almacenar el nuevo producto lo único que nos queda hacer es ver la consola para saber si hay una solicitud en red.

EDITAR PRODUCTO

Paso 1: creamos un onclick para editar el producto en la siguiente plantilla:

src\app\views\inventory\components\products\products.component.html


```
<br>
<p-button label="EDIT" styleClass="p-button-warning" (onClick)="edit(product)"/>
<br>
```

Creamos este método en el componente para llamar a la ventana emergente y le mandamos el producto:

src\app\views\inventory\components\products\products.component.ts

```
edit(product: IProduct):void{
  this.dialogService.open(ProductComponent, {
    header: "Modificar producto",
    data: product,
    height: '400px',
    width: '300px'
  });
}
```

Ahora bien dentro del componente “producto” usamos el método ngOnInit() y la inyección de dependencias para asociar el producto que le dimos editar con el formulario:

src\app\views\inventory\components\products\components\product\product.component.ts

```
ngOnInit(){
  const data = this.dynamicDialogConfig.data!;

  if(data){
    const product : IProduct = data;
    this.productsService.form = this.productsService.edit(product);
    this.isEditableProduct = true;
  }
}
```

Y también creamos un booleano que nos servirá de bandera para saber si un producto se está creando o editando.

Ahora lo que tenemos que hacer es que cuando se presione el save usar la bandera que está seteada en true para poder editar el producto:

src\app\views\inventory\components\products\components\product\product.component.ts

```

save(){
  console.log(this.productsService.form.value);

  const form: FormGroup = this.productsService.form;
  if (form.valid){
    const product: IProduct = this.productsService.form.value as IProduct;

    if(this.isEditableProduct){
      this.productsService.editProd(product);
    }else{
      this.productsService.create(product);
    }
  }else{
    alert("Error al guardar el producto.");
  }

  this.closeModalWindow();
}

```

Y ahora solo basta poner el método PUT en el servicio:

src\app\views\inventory\components\products\services\products.service.ts

```

editProd(product: IProduct){
  const editURL = `https://fakestoreapi.com/products/${product.id}`;

  this.httpClient.put(editURL, product).subscribe((data: any) => {
    this.arrProduct = data;
  });

  alert(`Edit product: ${product.id}`);
}

```

Advertencia: no se llama edit por qué ese nombre ya está ocupado en el repositorio cuando seteamos el formulario.