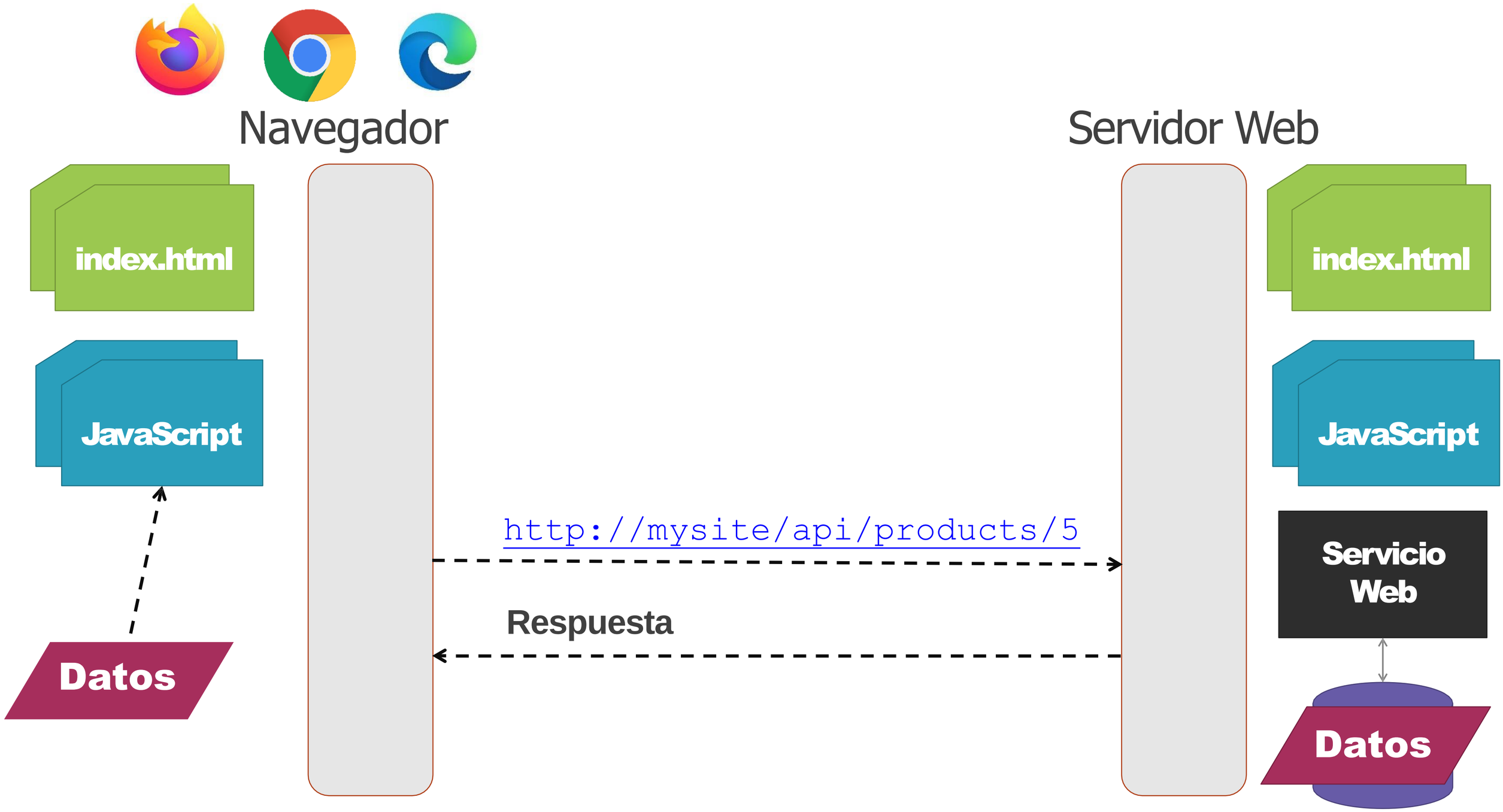
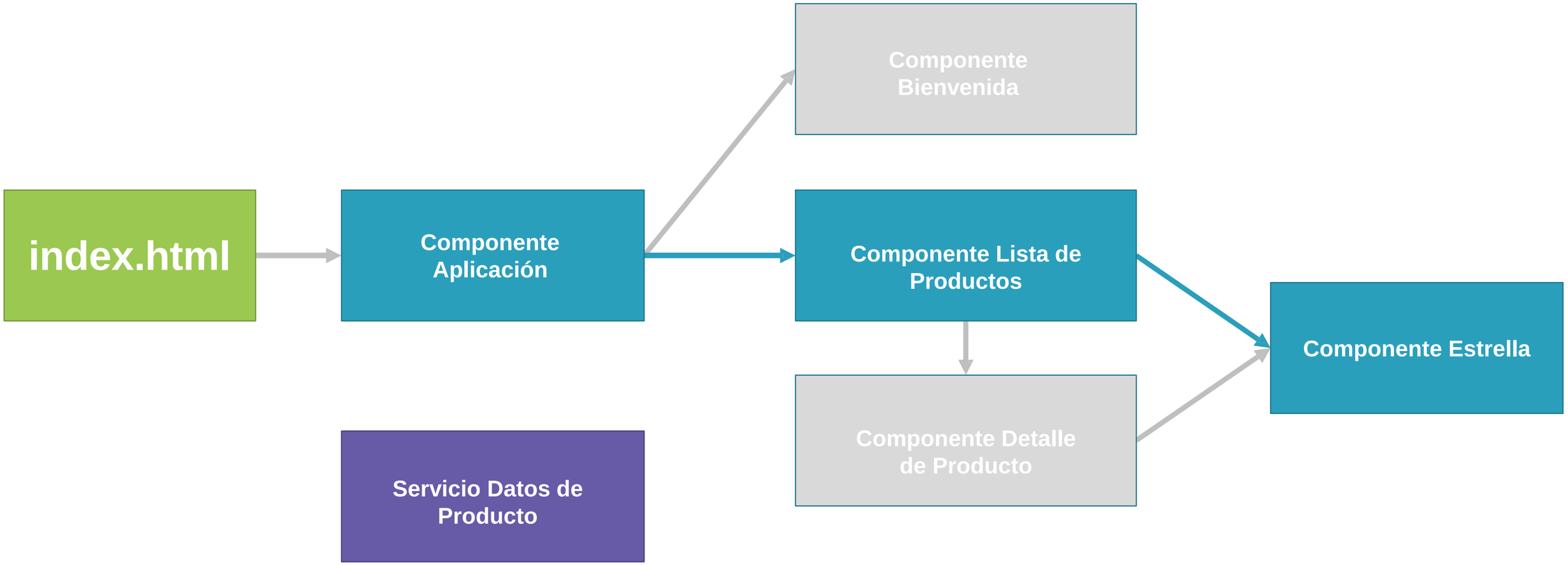
Obtener datos mediante HTTP


Navegador
Servidor Web
index.html
JavaScript
Datos
http://mysite/api/products/5
Respuesta
index.html
JavaScript
Servicio Web
Datos

Resumen

Observables y Reactive Extensions
Enviar una petición HTTP
Gestión de excepciones
Suscripción a un Observable

Arquitectura de la aplicación



Para entender el código HTTP, es importante entender Reactive Extensions y los Observables

Reactive Extensions (RxJS)

- **Una biblioteca para la composición de datos mediante secuencias observables**
- **Y transformar esos datos mediante operadores**
  - **Similar a los operadores LINQ de .NET**
- **Angular utiliza Extensiones Reactivas para trabajar con datos**
  - **Especialmente datos asíncronos** vienen de una api

Síncrono vs. Asíncrono

Síncrono: tiempo real

Asíncrono: No hay respuesta inmediata

Las peticiones HTTP son asíncronas, tanto las peticiones como las respuestas

Obtención de datos

Aplicación

- **Obtener una lista de productos**
- **Notifícame cuando se reciba la respuesta**
- **Continuamos**

Obtener productos

notifícame cuando obtengas respuesta y después continuamos

Servidor Web

En algún momento posterior...

Aplicación

- **Aviso, los datos ya han llegado**
- **Los voy a procesar**

Aquí están los productos

obtenemos un observable, un stream, en vez de una cadena

Servidor Web

Observable

Una colección de elementos a lo largo del tiempo

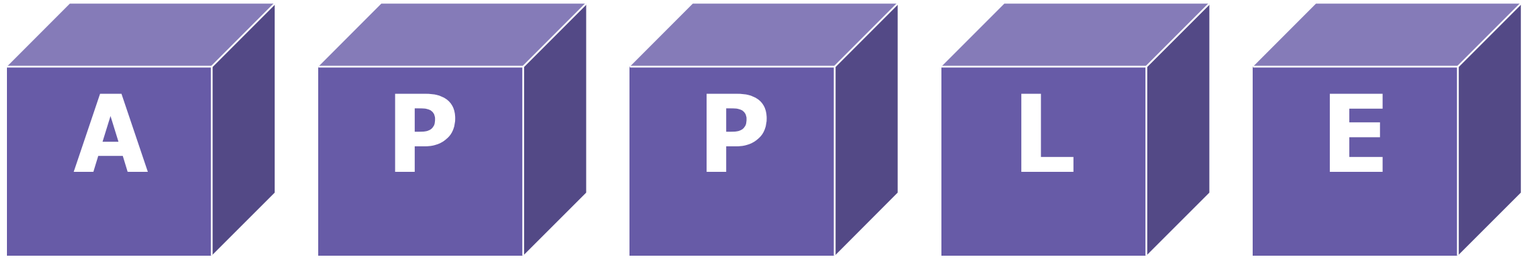
sin inicio y sin final

- **A diferencia de un array, no retiene elementos**
- **Los elementos emitidos pueden observarse a lo largo del tiempo**

no puedo preguntar cuántos elementos tiene

Array: [A, P, P, L, E]

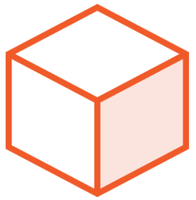
Observable:



solo alguien que está a la escucha oirá apple

¿Qué hace un observable?

Nada hasta que nos suscribimos



`next`: **Se emite el siguiente elemento**

`error`: **Se ha producido un error y no se emiten más elementos**

`complete`: **No se emiten más artículos**

3 propiedades

Obtener datos

cómo funciona en realidad, lo de pág. 8 no es lo que pasa

Aplicación

- **Realiza una llamada http get**
- **http get devuelve un Observable, que emitirá notificaciones**
- **Suscribirse para iniciar el Observable y**
- **se envía la solicitud de obtención de productos**
- **La ejecución continúa con la siguiente línea**

Obtener productos →

Servidor Web

En algún momento posterior...

Aplicación

- **Obtenemos la respuesta**
- **El Observable emite una notificación** `next`
- **Procesamos la respuesta emitida**

dentro del suscribe!! Fuera no hay nada

Aquí tenemos los productos ←

[{cart},{hammer},{saw}]

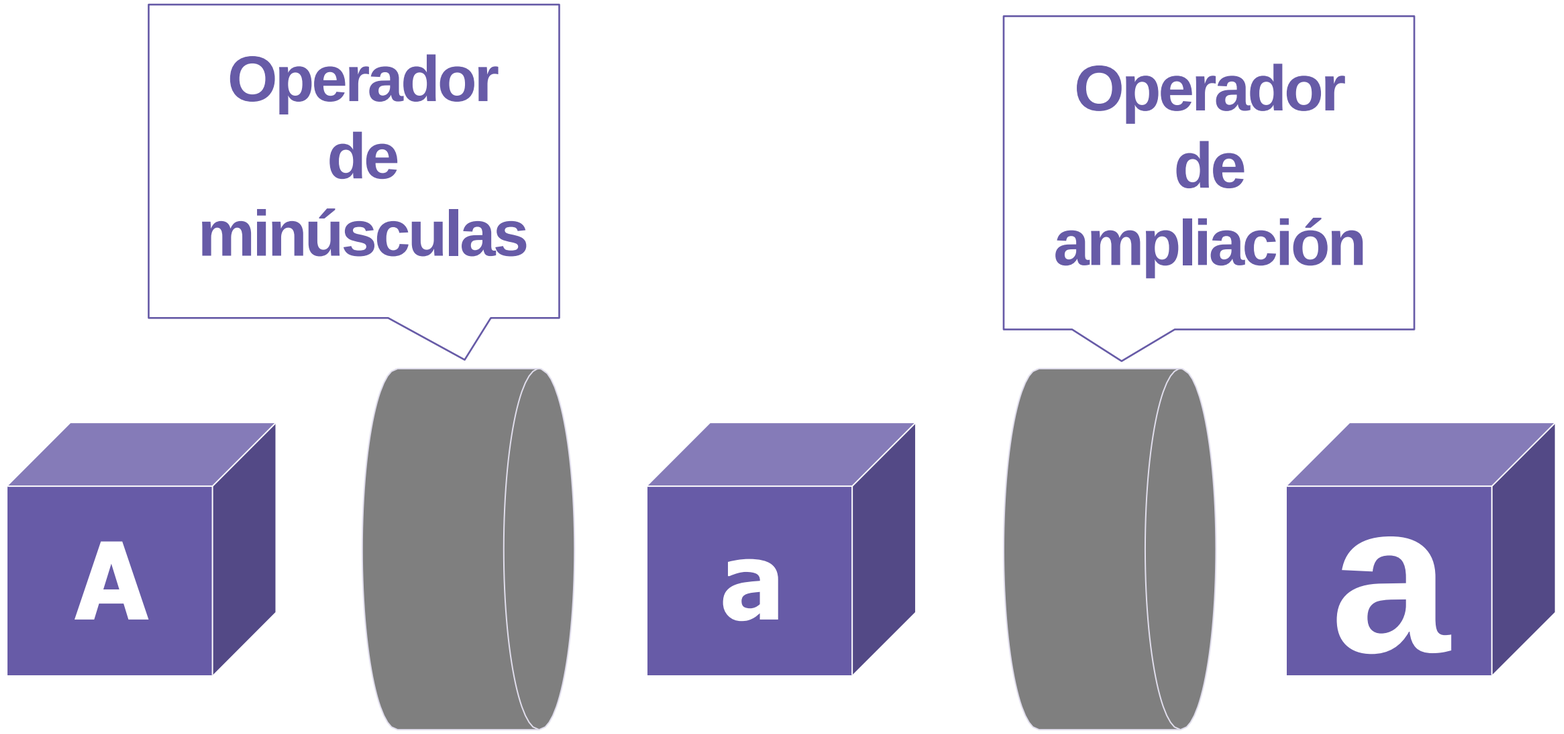
ya tenemos array con todos los productos.
DEVOLVEMOS TODO EL ARRAY (No yield return y de uno en uno)

Servidor Web

(imagina) suponer que tenemos unas lentes que cambian lo que vemos

Observable Pipe

Los pipes transforman. Otros se enlazan con el observable y lo muestran por console log. En el ejemplo que tenemos ahora se ve esto.

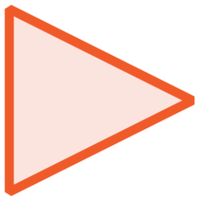


original

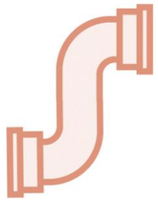
Reactive Extensions (RxJS)

- **Una biblioteca para la composición de datos mediante secuencias observables**
- **Y transformar esos datos mediante operadores**
  - **Similar a los operadores LINQ de .NET**
- **Angular utiliza Extensiones Reactivas para trabajar con datos**
  - **Especialmente datos asíncronos**

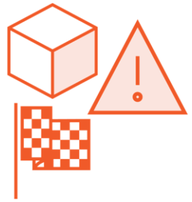
Uso común de observables



Iniciar el Observable (subscribe)



Emitir elementos a través de un conjunto de operadores



Procesar notificaciones: next, error, complete

Parar el observable (unsubscribe)

HAY QUE DESUSCRIBIRSE!!!!!

Ejemplo sin apis

Ejemplo

Ejemplo

```
import { Observable, range } from 'rxjs';

import { map, filter } from 'rxjs/operators';



const source$: Observable<number> = range(0, 10);

source$.pipe(
    map(x => x * 3),
    filter(x => x % 2 === 0)
).subscribe(x => console.log(x));
```

observable, como list

0, 1, 2 ... 10

está ahí, pero no se ha ejecutado

funciones arrow/expresiones lambda

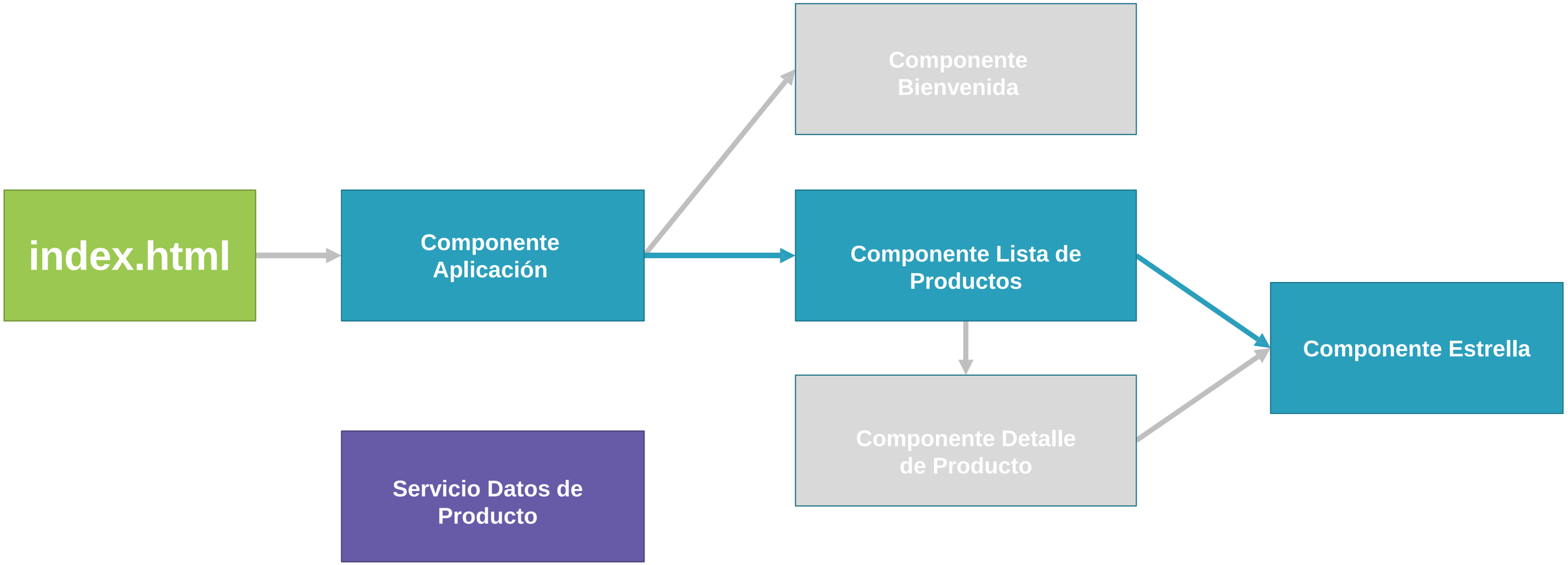
resultado pipe se pasa al siguiente, de uno en uno

debemos cumplir esto

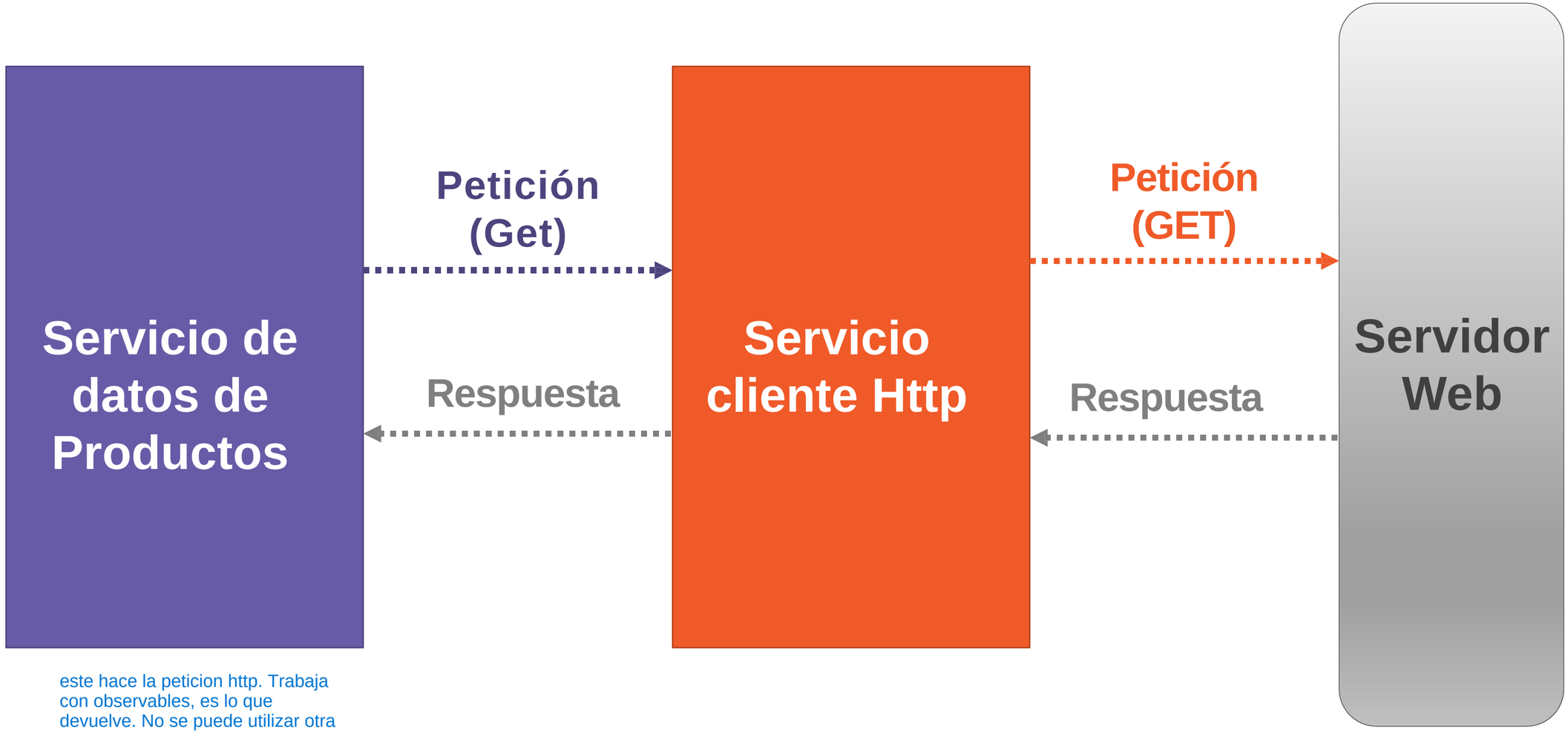
Resultado

```
0
6
12
18
24
```

Application Architecture



Enviar una petición HTTP



este hace la peticion http. Trabaja con observables, es lo que devuelve. No se puede utilizar otra cosa

En el mismo sitio, pero diferentes cosas

Configurando una petición HTTP

product.service.ts

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

@Injectable({
   providedIn: 'root'
})
export class ProductService {
  private productUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts() {
    return this.http.get(this.productUrl);

  }
}
```

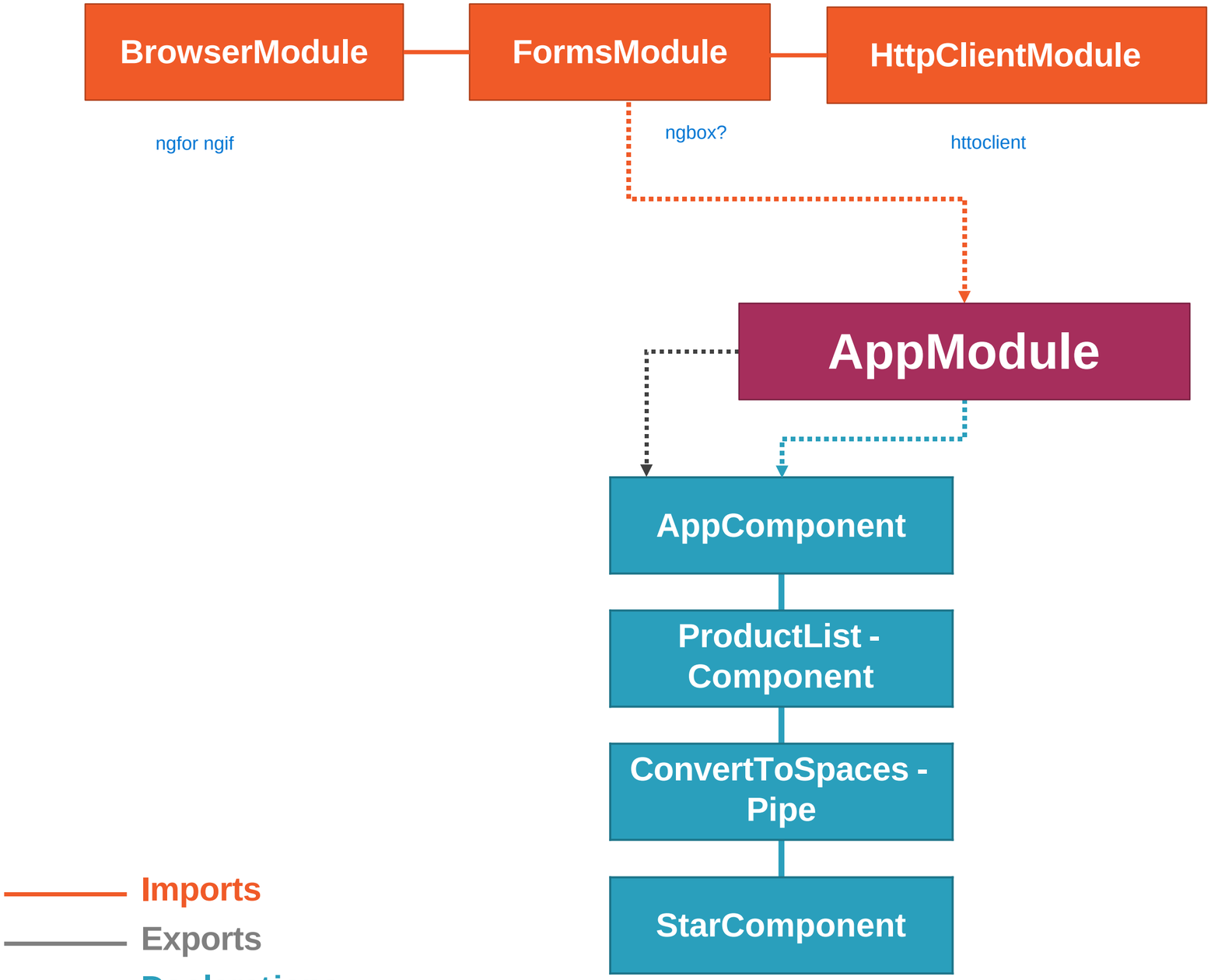
injectable porque el servicio de productos lo vamos a utilizar en múltiples componentes

devuelve un any. Falta algo para que sea la versión final

Registrando el proveedor del servicio HTTP

app.module.ts

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

@NgModule({
  imports: [
      BrowserModule,
      FormsModule,
      HttpClientModule ],
  declarations: [
       AppComponent,
       ProductListComponent,
       ConvertToSpacesPipe,
       StarComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```


BrowserModule
FormsModule
HttpClientModule
ngfor ngif
ngbox?
httoclient
AppModule
AppComponent
ProductList - Component
ConvertToSpaces - Pipe
StarComponent
Imports
Exports
Declarations
Bootstrap

Configurando una petición HTTP

product.service.ts

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

@Injectable({
   providedIn: 'root'
})
export class ProductService {
  private productUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts() {
   return this.http.get(this.productUrl);

  }
}
```

no es lo correcto

Configurando una petición HTTP

product.service.ts

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

@Injectable({
   providedIn: 'root'
})
export class ProductService {
  private productUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts() {
   return this.http.get<IProduct[]>(this.productUrl);

  }
}
```

mejor, pero aún falta algo

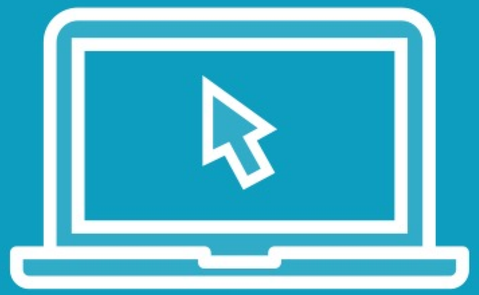
Configurando una petición HTTP

product.service.ts

```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
   providedIn: 'root'
})
export class ProductService {
  private productUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<IProduct[]> {
   return this.http.get<IProduct[]>(this.productUrl);

  }
}
```

Así, debemos devolver un observable

Demo



Configurando una petición HTTP

Manejo de excepciones

product.service.ts

```
...
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';
...

  getProducts(): Observable<IProduct[]> {
   return this.http.get<IProduct[]>(this.productUrl).pipe(
     tap(data => console.log('All: ', JSON.stringify(data))),
     catchError(this.handleError)
   );
  }

  private handleError(err: HttpErrorResponse) {
  }
```

Suscripción a un observable

```
x.subscribe()

x.subscribe(Observer)

x.subscribe({
    nextFn,
    errorFn,
    completeFn
})

const sub = x.subscribe({
     nextFn,
     errorFn,
     completeFn
})
```

Suscripción a un observable

product.service.ts

```
getProducts(): Observable<IProduct[]> {
 return this.http.get<IProduct[]>(this.productUrl).pipe(
   tap(data => console.log('All: ', JSON.stringify(data))),
   catchError(this.handleError)
 );
}
```

product-list.component.ts

```
ngOnInit(): void {
   this.productService.getProducts().subscribe({
      next: products => this.products = products,
      error: err => this.errorMessage = err
   });
}
```

Cancelar la suscripción a un Observable

Almacenar la suscripción en una variable

Implementar el hook del ciclo de vida OnDestroy

Utilizar la variable de suscripción para cancelar la suscripción

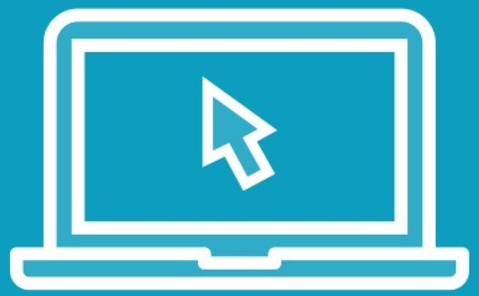
Cancelar la suscripción a un Observable

product-list.component.ts

```
ngOnInit(): void {
    this.sub = this.productService.getProducts().subscribe({
        next: products => this.products = products,
        error: err => this.errorMessage = err
    });
}
```

```
ngOnDestroy(): void {
    this.sub.unsubscribe();
}
```

Demo



Suscripción a un Observable
Cancelar la suscripción a un Observable

Lista de comprobación de HTTP: Configuración

Añade HttpClientModule a la matriz de importaciones de uno de los módulos Angular de la aplicación

```
@NgModule({
  imports: [
      BrowserModule,
      FormsModule,
      HttpClientModule ],
  declarations: [...],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Lista de comprobación de HTTP: Llamar a HTTP Get

Definir una dependencia para el servicio de cliente Http en el constructor

Crear un método para cada petición HTTP

Llamar al método HTTP deseado, como por ejemplo get

Usar genéricos para especificar el tipo devuelto

```
export class ProductService {
  private productUrl = 'www.myService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<IProduct[]> {
   return this.http.get<IProduct[]>(this.productUrl);
  }
}
```

Lista de comprobación de HTTP: Manejo de excepciones

Añadir gestión de errores

```
getProducts(): Observable<IProduct[]> {
  return this.http.get<IProduct[]>(this.productUrl).pipe(
    tap(data => console.log(JSON.stringify(data))),
    catchError(this.handleError)
  );
}

private handleError(err: HttpErrorResponse) {
}
```

Lista de comprobación de HTTP: Suscripción

Llamar al método de suscripción del observable devuelto

Proporcionar una función para manejar un elemento emitido

Proporcionar una función para manejar cualquier error devuelto

```
ngOnInit(): void {
    this.productService.getProducts().subscribe({
        next: products => this.products = products,
        error: err => this.errorMessage = err
    });
}
```

Lista de comprobación de HTTP: Cancelar Suscripción

Guardar la suscripción en una variable

```
this.sub = this.ps.getProducts().subscribe(…)
```

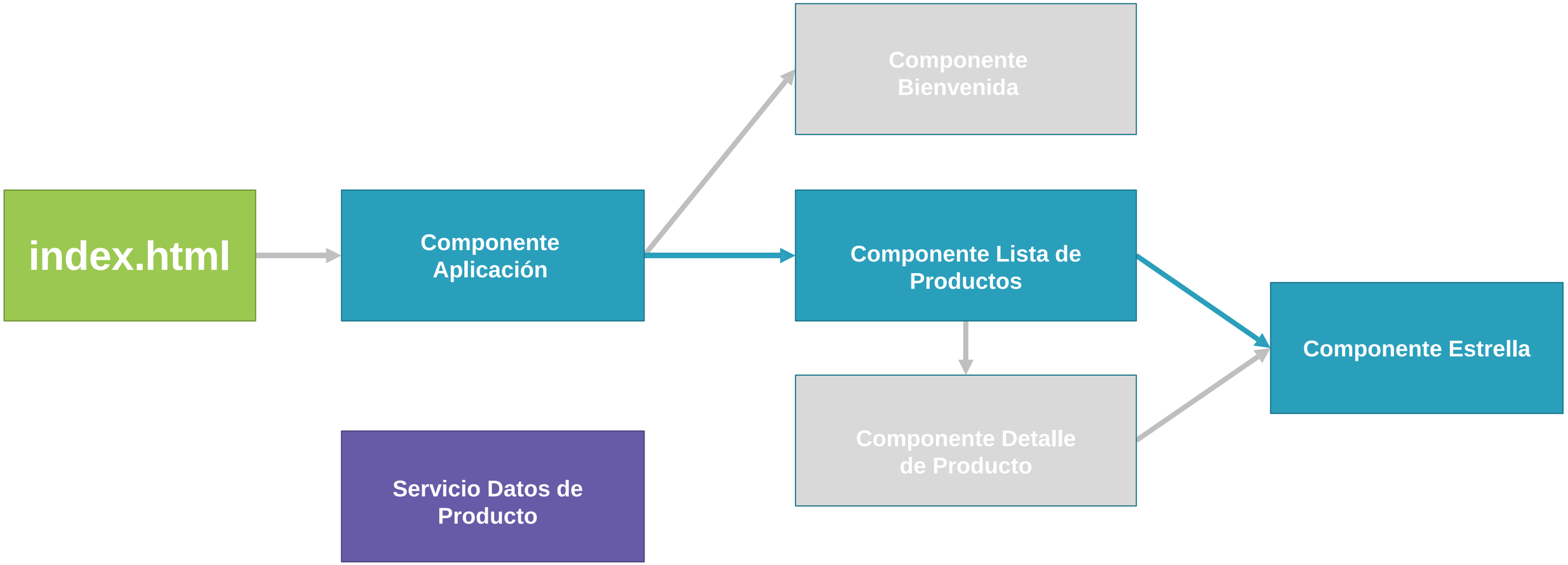
Implementar el hook del ciclo de vida OnDestroy

```
export class PLComponent implements OnInit, OnDestroy
```

Utilice la variable de suscripción para cancelar la suscripción

```
ngOnDestroy(): void {
    this.sub.unsubscribe();
}
```

Application Architecture



A continuación …

Fundamentos de navegación y enrutamiento