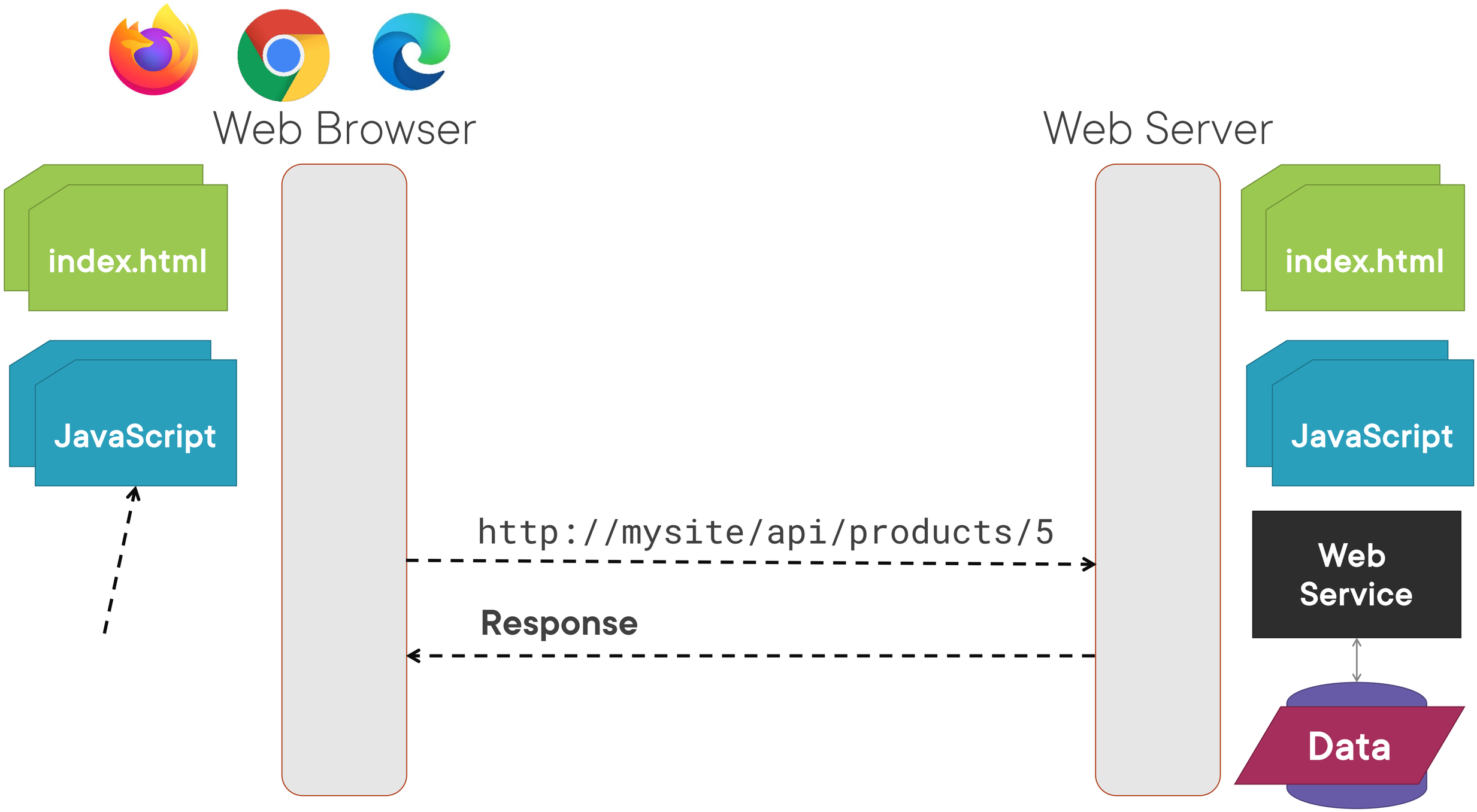
Retrieving Data Using HTTP

Deborah Kurata
Consultant | Speaker | Author | MVP | GDE

@deborahkurata


Web Browser
Web Server
index.html
JavaScript
http://mysite/api/products/5
Response
index.html
JavaScript
Web Service
Data

Module Overview

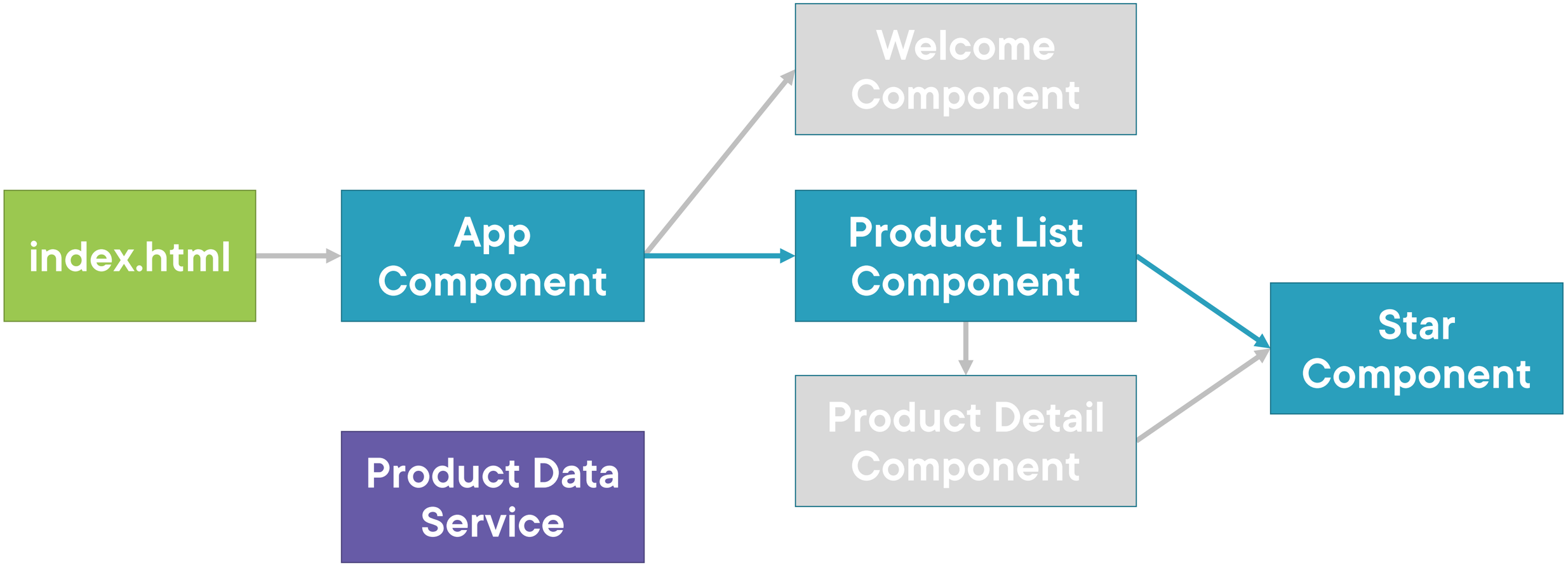
Observables and Reactive Extensions

Sending an HTTP request

Exception handling

Subscribing to an Observable

Application Architecture



To understand the HTTP code,
it's important to understand
Reactive Extensions and
Observables

Reactive Extensions (RxJS)

A library for composing data using observable sequences

And transforming that data using operators

- **Similar to .NET LINQ operators**

Angular uses Reactive Extensions for working with data

- **Especially asynchronous data**

Synchronous vs. Asynchronous

Synchronous: real time

Asynchronous: No immediate response

HTTP requests are asynchronous: request and response

Getting Data

Application

- **Get me a list of products**
- **Notify me when the response arrives**
- **I'll continue along**

Get me products →

Web Server

At some later point in time...

Application

- **"Hey, your data arrived"**
- **OK, I'll process it. Thanks!**

← Here are the products

Web Server

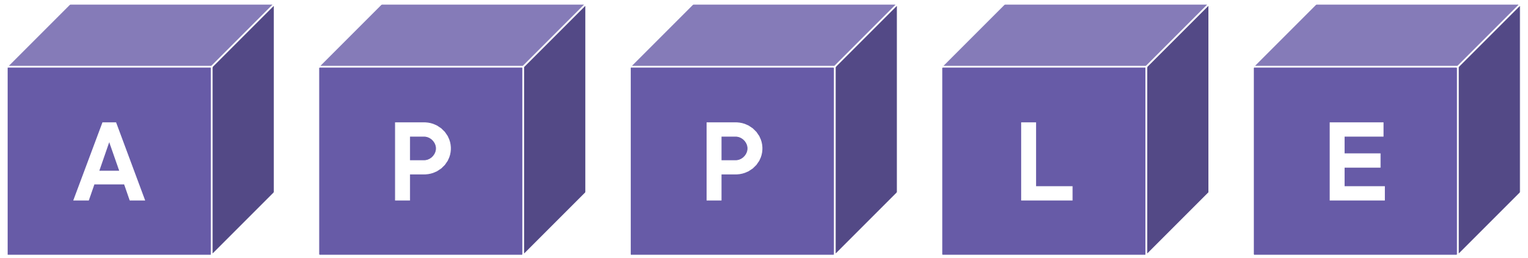
Observable

A collection of items over time

- **Unlike an array, it doesn't retain items**
- **Emitted items can be observed over time**

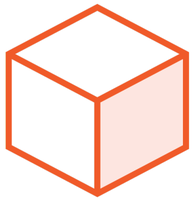
Array: [A, P, P, L, E]

Observable:



What Does an Observable Do?

Nothing until we subscribe



`next`: **Next item is emitted**

`error`: **An error occurred and no more items are emitted**

`complete`: **No more items are emitted**

Getting Data

Application

- Call http get
- http get returns an Observable, which will emit notifications
- Subscribe to start the Observable and the get request is sent
- Code continues along

Get me products →

Web Server

At some later point in time...

Application

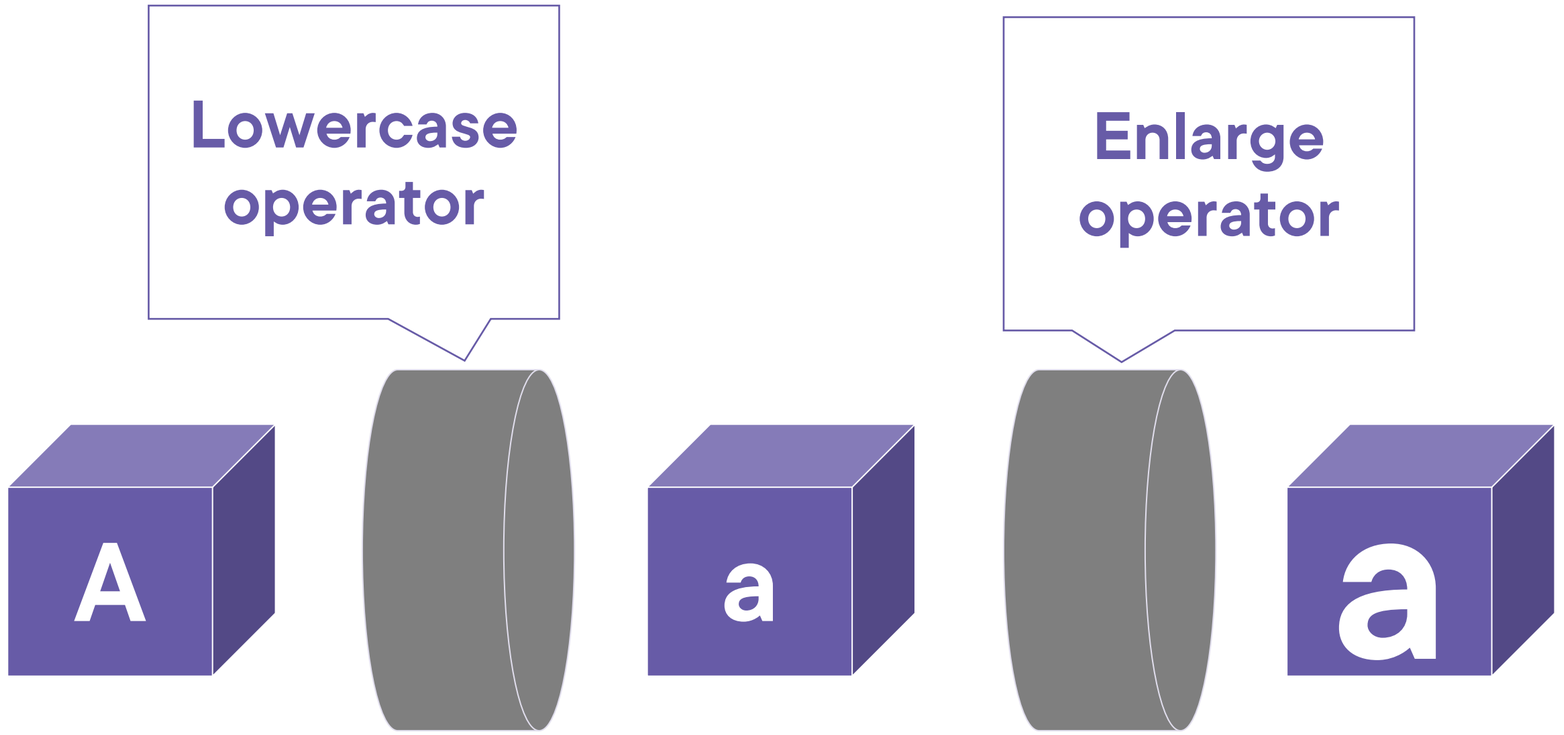
- The response is returned
- The Observable emits a `next` notification
- We process the emitted response

← Here are the products

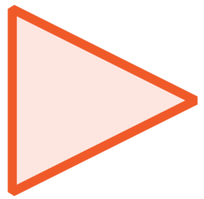
[{cart},{hammer},{saw}]

Web Server

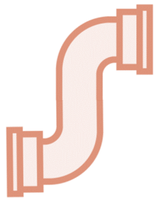
Observable Pipe



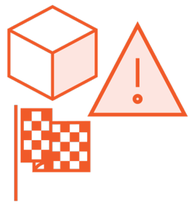
Common Observable Usage



Start the Observable (subscribe)



Pipe emitted items through a set of operators



Process notifications: next, error, complete

Stop the Observable (unsubscribe)

Example

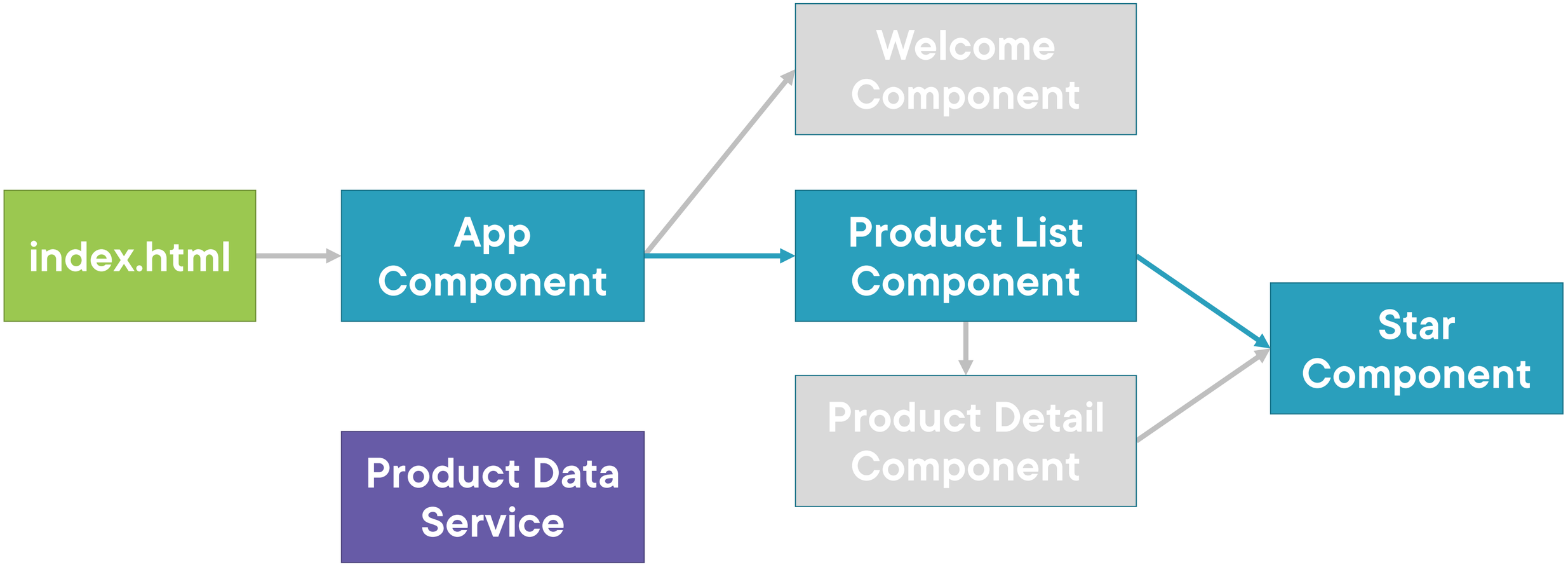
Example

```
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));
```

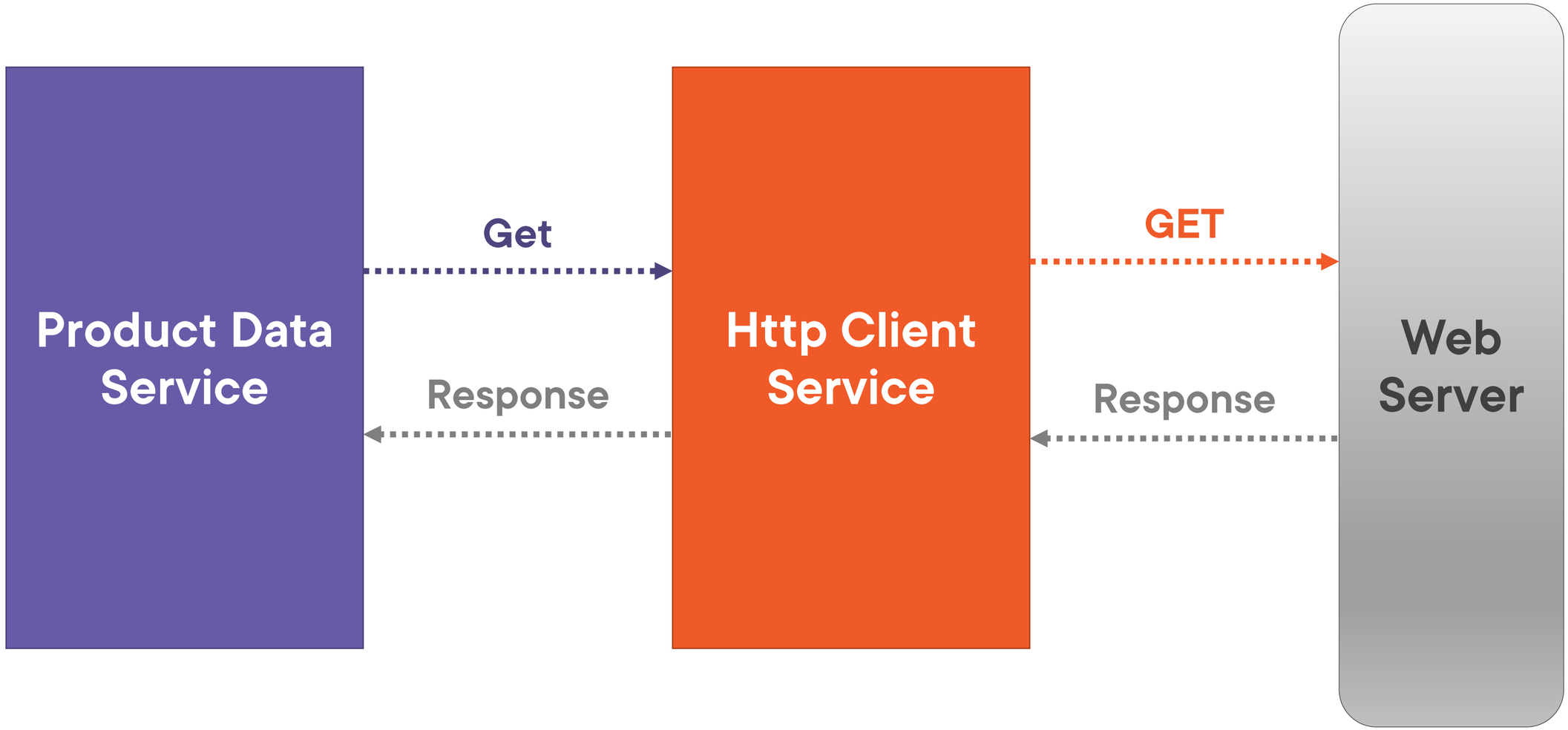
Result

```
0
6
12
18
24
```

Application Architecture



Sending an HTTP Request



Setting up an HTTP Request

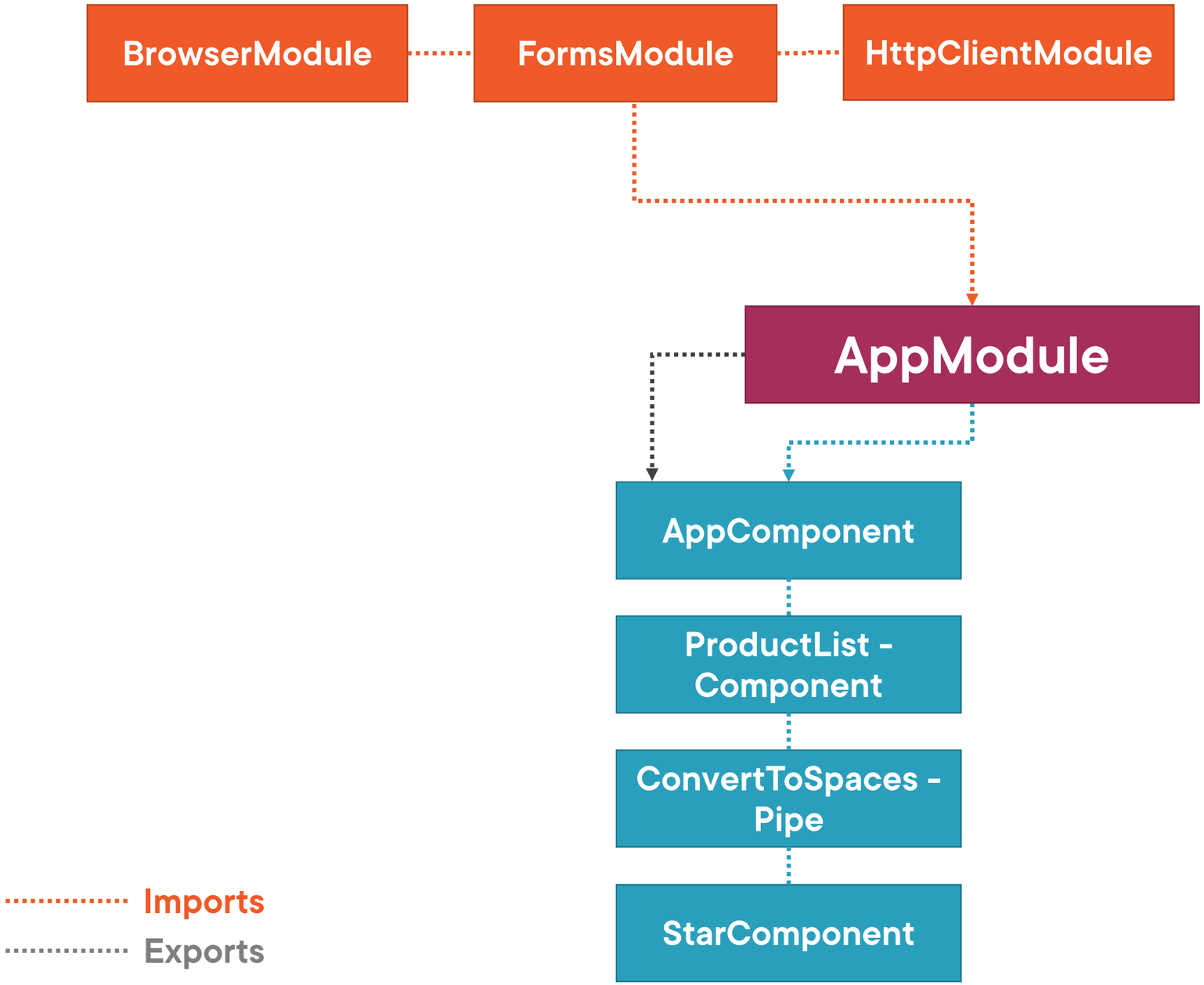
product.service.ts

```
...
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);

  }
}
```

Registering the HTTP Service Provider

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
AppModule
AppComponent
ProductList - Component
ConvertToSpaces - Pipe
StarComponent
Imports
Exports
Declarations
Bootstrap

Setting up an HTTP Request

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);

  }
}
```

Setting up an HTTP Request

product.service.ts

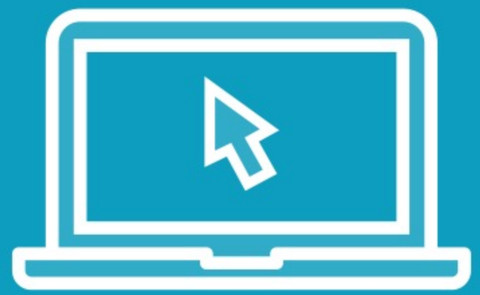
```
...
import { HttpClient } from '@angular/common/http';

@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);

  }
}
```

Setting up an HTTP Request

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);

  }
}
```

Demo



Setting up an HTTP request

Exception Handling

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) {
  }
```

Subscribing to an Observable

```
x.subscribe()

x.subscribe(Observer)

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

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

Subscribing to an 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
   });
}
```

Unsubscribing from an Observable

Store the subscription in a variable

Implement the OnDestroy lifecycle hook

Use the subscription variable to unsubscribe

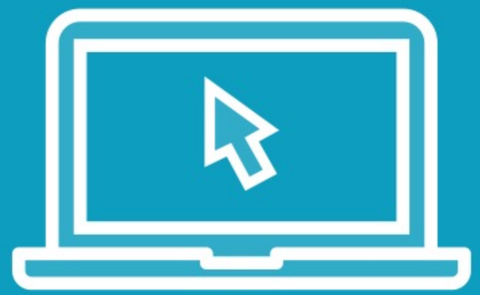
Unsubscribing from an Observable

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



Subscribing to an Observable

Unsubscribing from an Observable

HTTP Checklist: Setup

Add HttpClientModule to the imports array of one of the application's Angular Modules

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

HTTP Checklist: Calling HTTP Get

Define a dependency for the Http Client Service in the constructor

Create a method for each HTTP request

Call the desired HTTP method, such as get

Use generics to specify the returned type

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

  constructor(private http: HttpClient) { }

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

HTTP Checklist: Exception Handling

Add error handling

```
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) {
}
```

HTTP Checklist: Subscribing

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

Provide a function to handle any returned errors

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

HTTP Checklist: Unsubscribing

Store the subscription in a variable

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

Implement the OnDestroy lifecycle hook

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

Use the subscription variable to unsubscribe

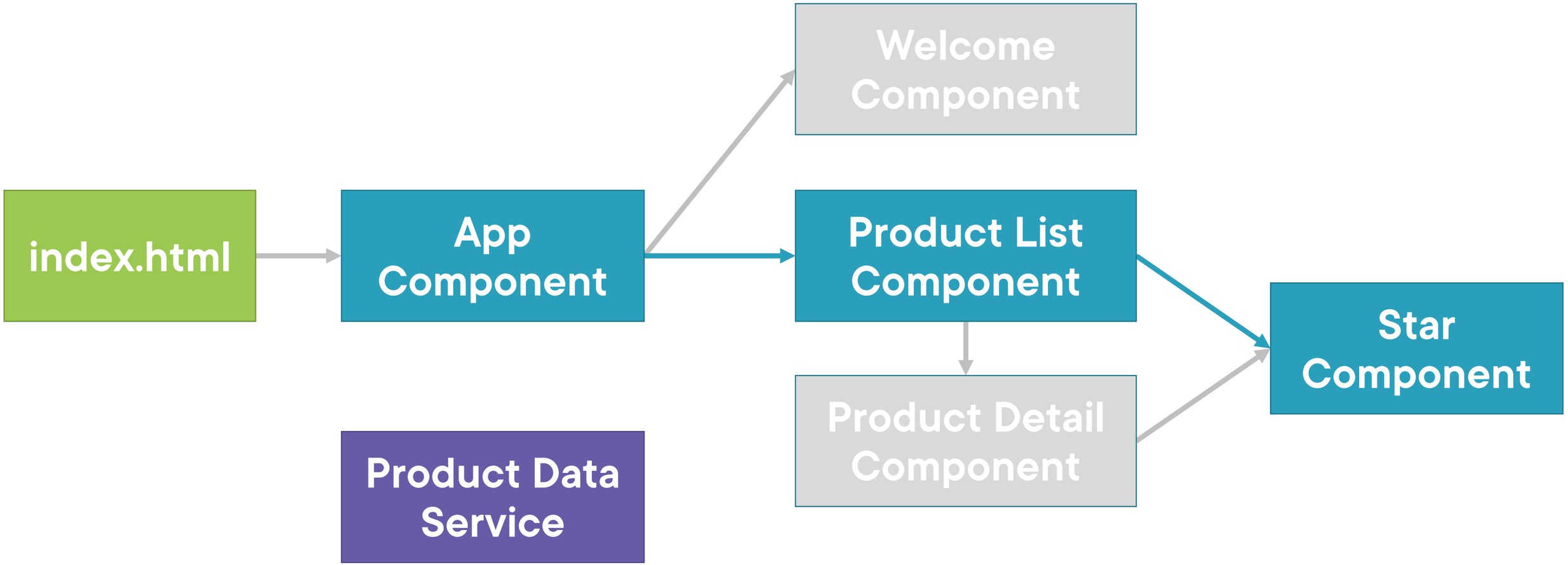
```
ngOnDestroy(): void {
    this.sub.unsubscribe();
}
```

Learning More

Pluralsight Courses

- **"Angular: Reactive Forms"**
  - **HTTP and CRUD**
- **"RxJS in Angular: Reactive Development"**
  - **RxJS and Observables**
- **"Angular HTTP Communication"**
  - **Intermediate HTTP Techniques**

Application Architecture



Coming up next ...

Navigation and Routing Basics