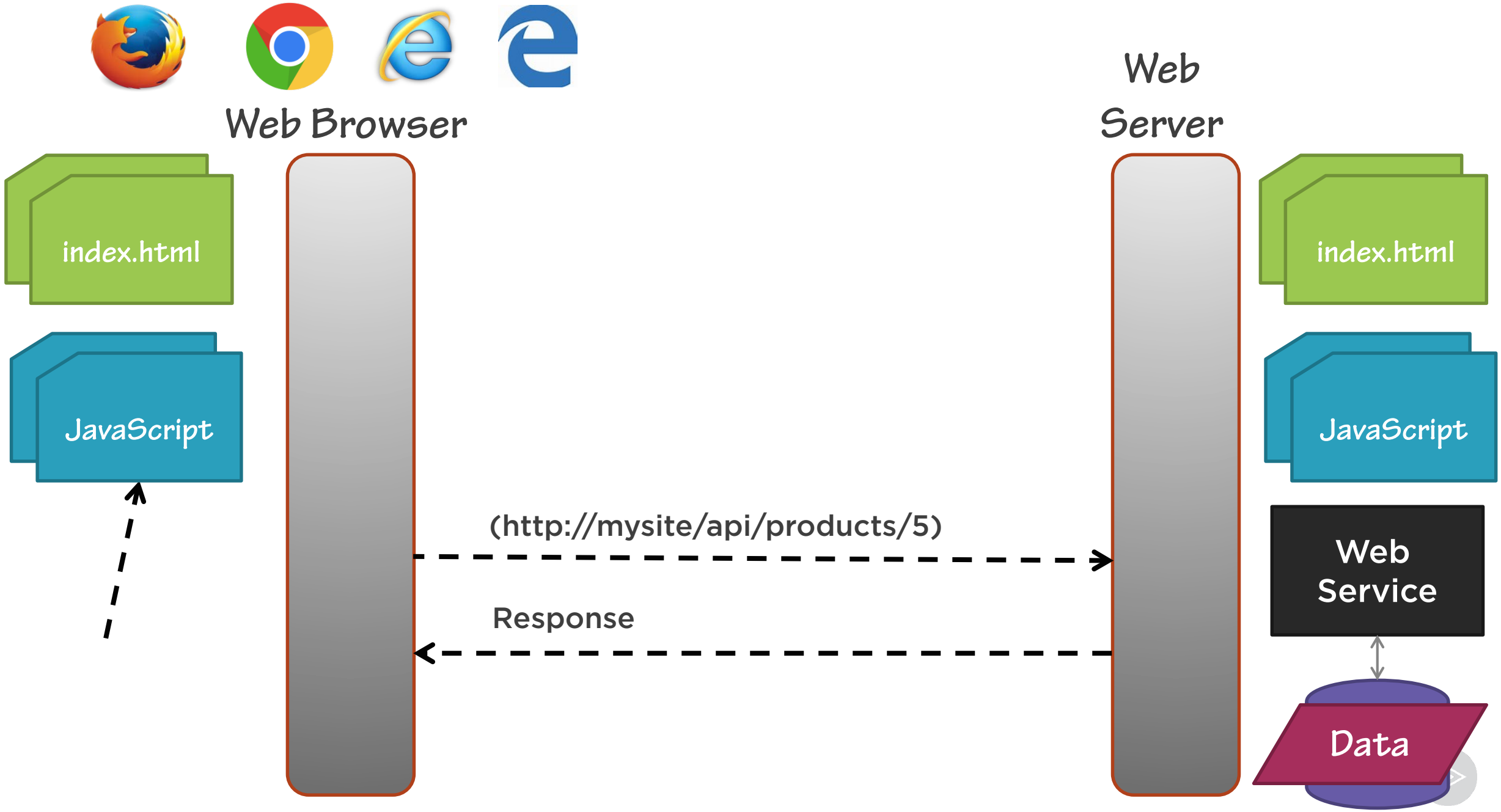
Retrieving Data Using Http

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


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

Module Overview

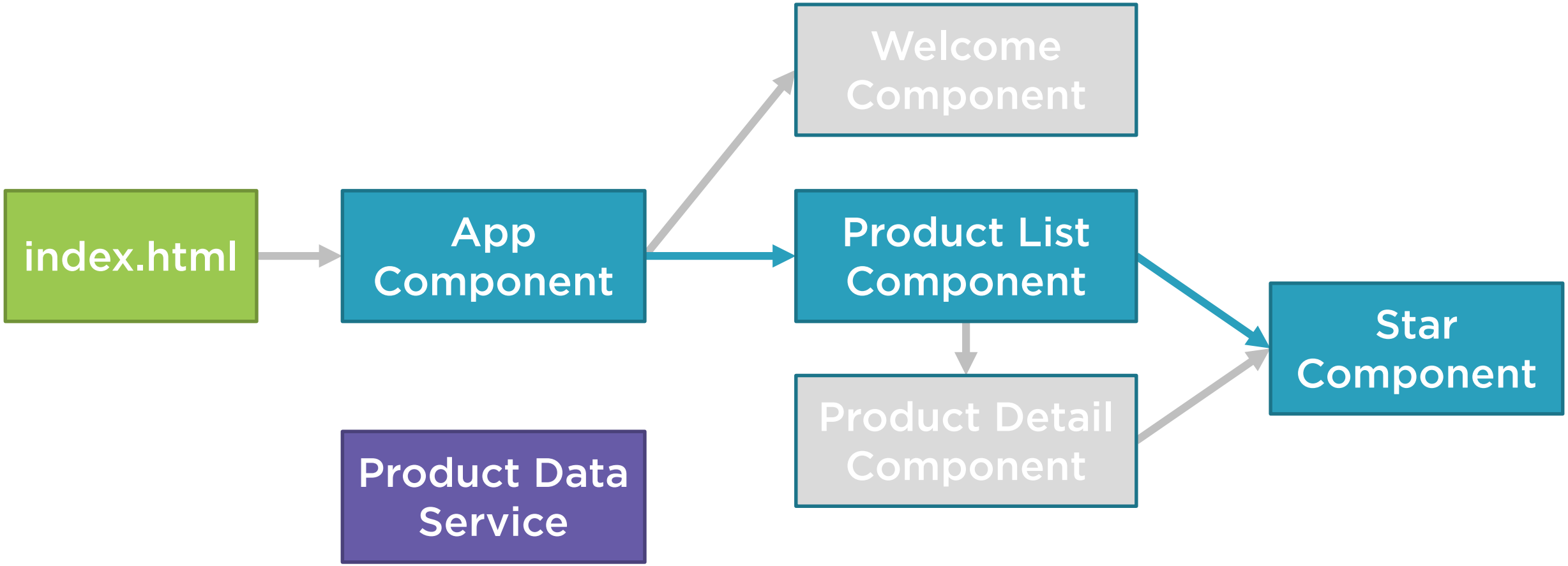
Observables and Reactive Extensions

Sending an Http Request

Exception Handling

Subscribing to an Observable

Application Architecture



Observables and Reactive Extensions

Reactive Extensions (RxJS)

Help manage asynchronous data

Treat events as a collection

- An array whose items arrive asynchronously over time

Subscribe to receive notifications

Are used within Angular

Observable Operators

Methods on observables that compose new observables

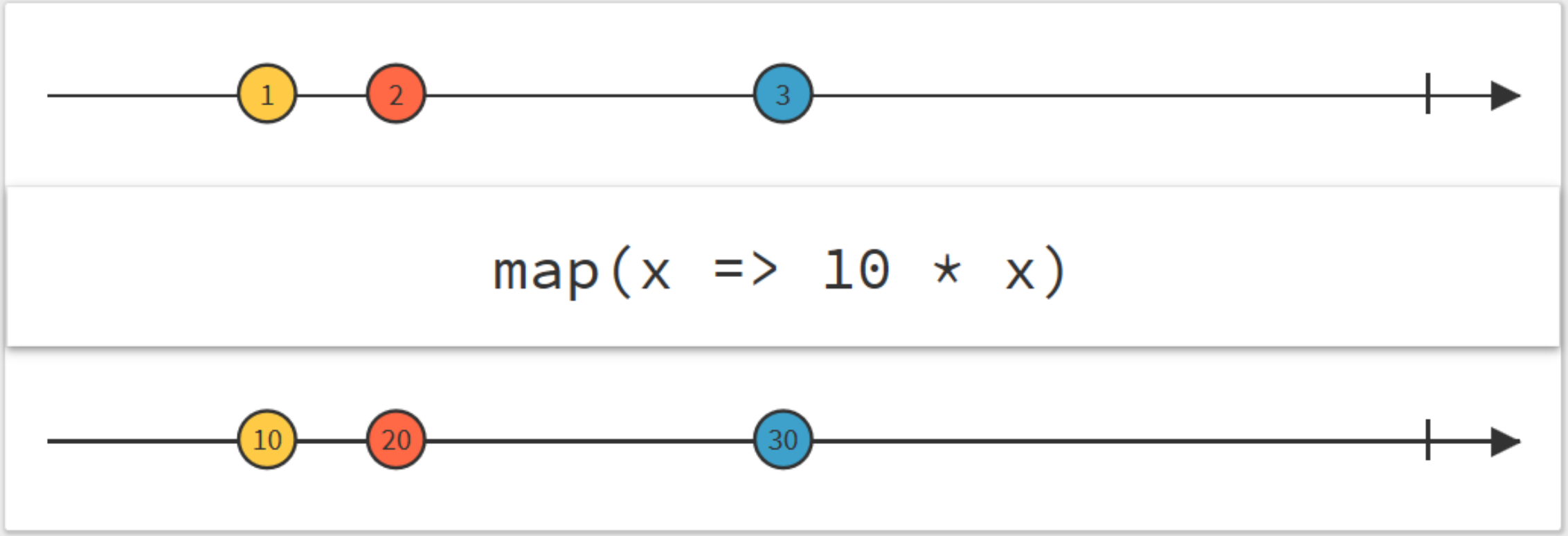
Transform the source observable in some way

Process each value as it is emitted

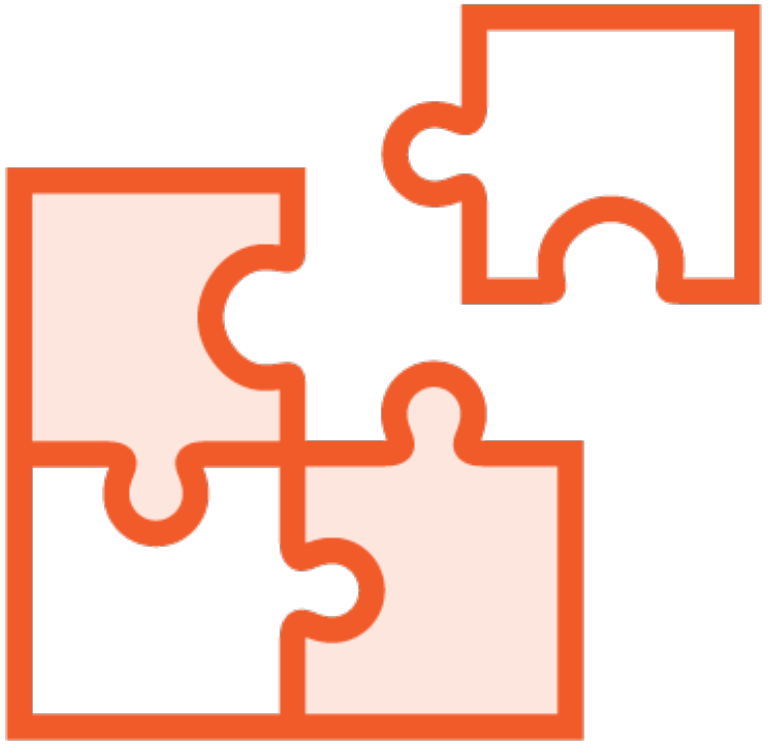
Examples: map, filter, take, merge, ...

Observables

Interactive diagrams of Rx Observables



Composing Operators



Compose operators with the pipe method

Often called "pipeable operators"

Composing Operators

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));
```

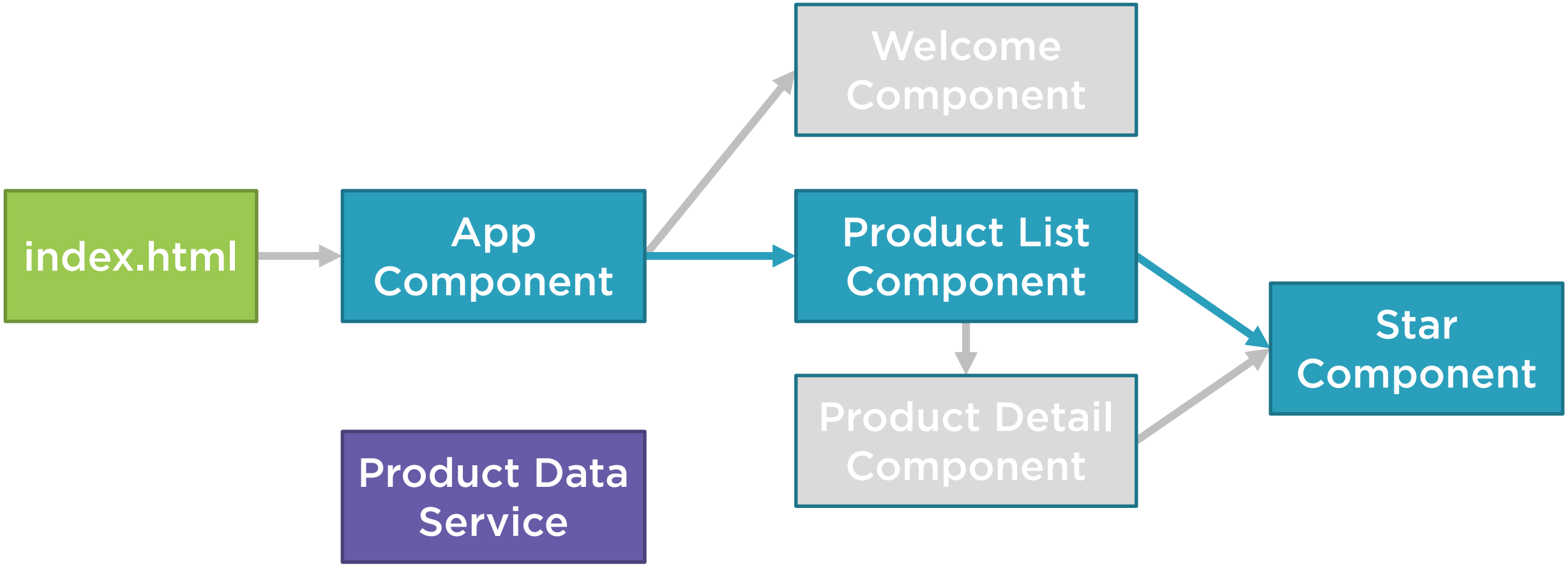
Result

```
0
6
12
18
24
```

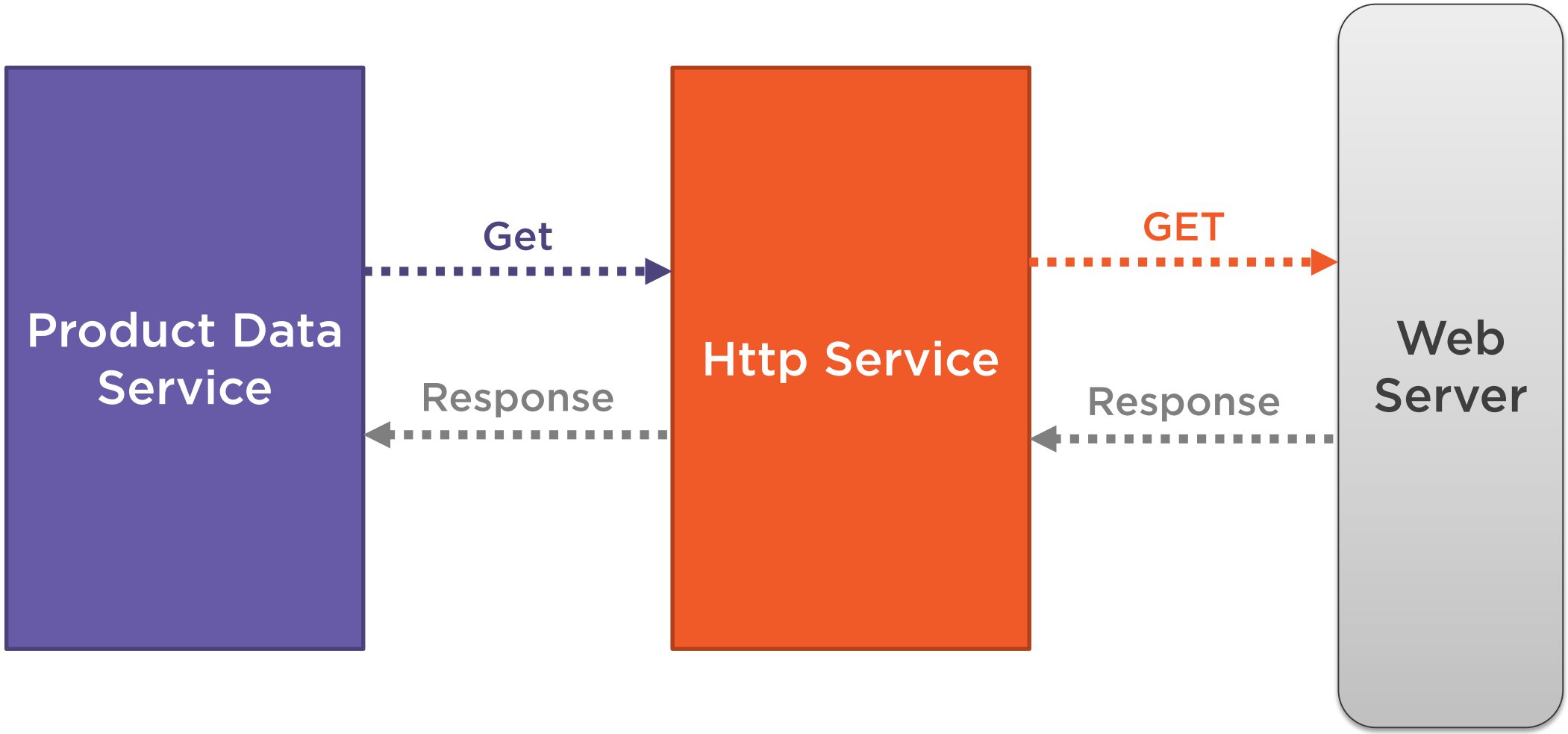
Promise vs Observable

Promise	Observable
Provides a single future value	Emits multiple values over time
Not lazy	Lazy
Not cancellable	Cancellable
	Supports map, filter, reduce and similar operators

Application Architecture



Sending an Http Request



Sending an Http Request

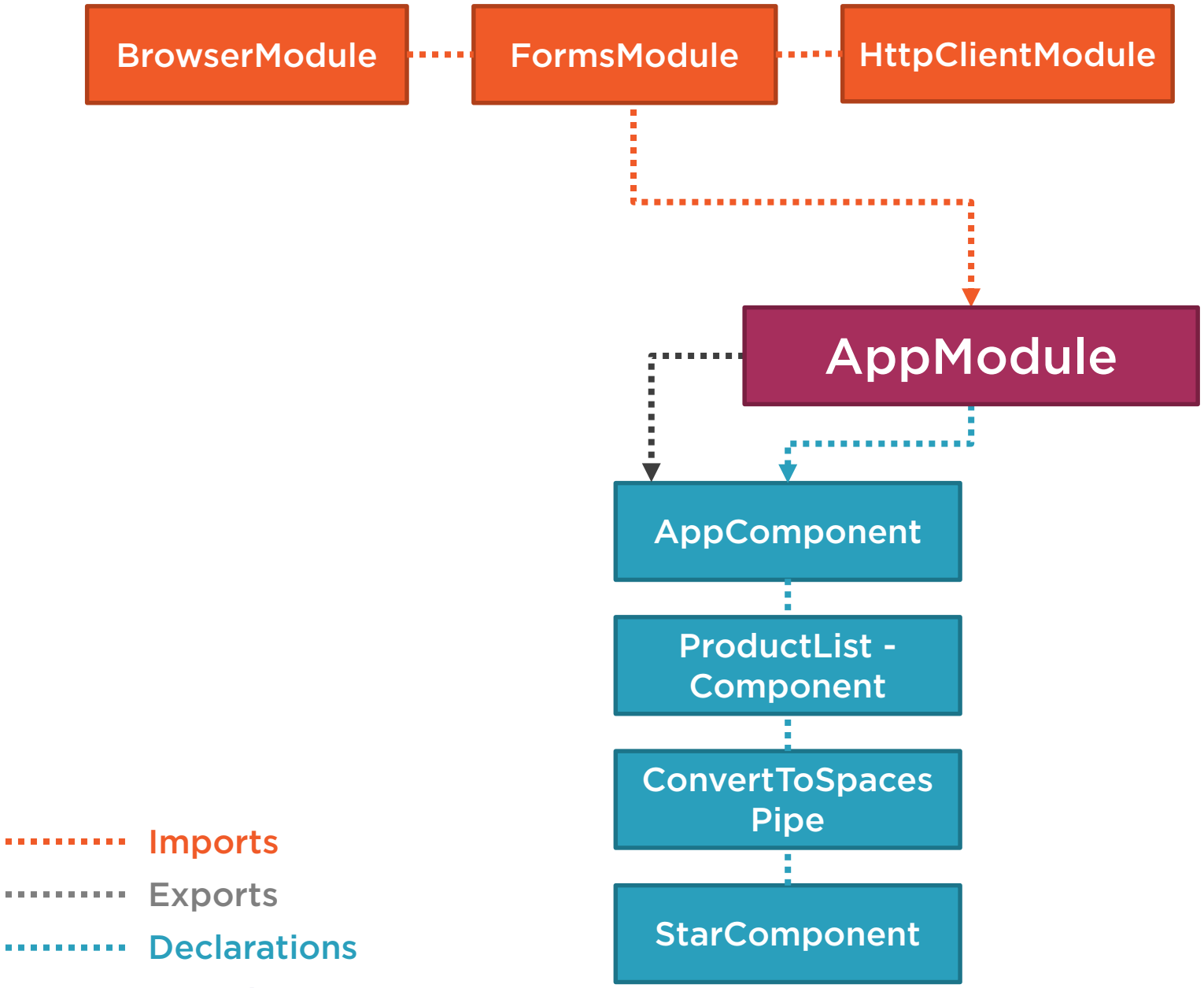
product.service.ts

```
...
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
Providers
Bootstrap

Sending 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);

  }
}
```

Sending an Http Request

product.service.ts

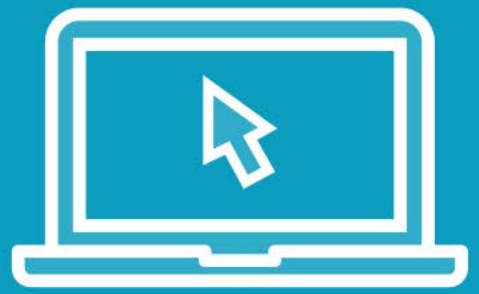
```
...
import { HttpClient } from '@angular/common/http';


@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);

  }
}
```

Sending 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



Sending 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) {
  }
```

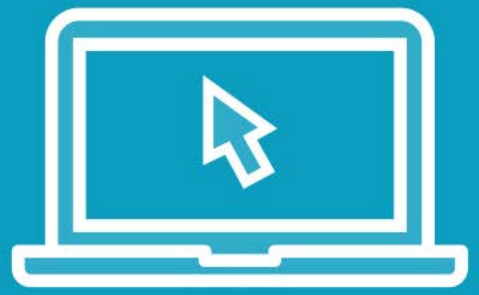
Subscribing to an Observable

```
x.subscribe(nextFn)

x.subscribe(nextFn, errorFn)

x.subscribe(nextFn, errorFn, completeFn)

let sub = x.subscribe(nextFn, errorFn, completeFn)
```

product-list.component.ts

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

Demo



Subscribing to an Observable

Http Checklist: Setup

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

Http Checklist: Service

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

Use generics to specify the returned type

Add error handling

Http Checklist: Subscribing

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

- Normally assigns a property to the returned JSON object

Provide an error function to handle any returned errors

Learning More

Pluralsight Courses

- "Angular: Reactive Forms"
  - HTTP and CRUD
- "Play by Play: Angular 2/RxJS/HTTP and RESTful Services with John Papa and Dan Wahlin"
  - RxJS and Observables
- "Angular HTTP Communication"
  - Intermediate HTTP Techniques

Summary

- Observables and Reactive Extensions
- Sending an Http Request
- Exception Handling
- Subscribing to an Observable

Application Architecture