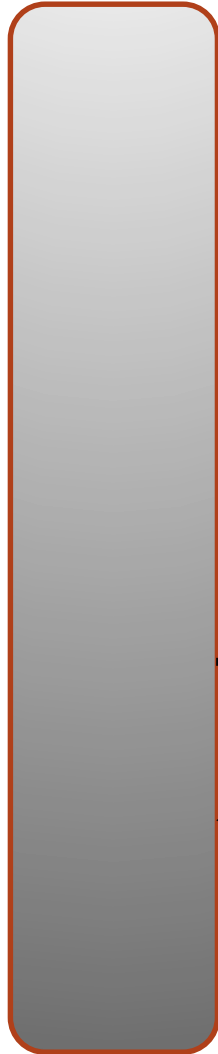


Retrieving Data Using Http

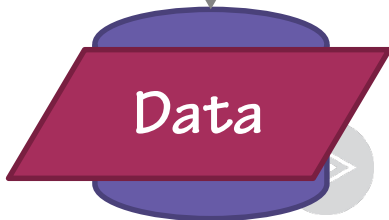
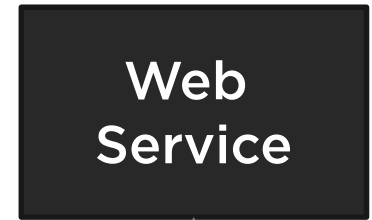




Web Browser



Web Server



(http://mysite/api/products/5)

Response

Module Overview



Observables and Reactive Extensions

Sending an Http Request

Exception Handling

Subscribing to an Observable



Observables and Reactive Extensions



Help manage asynchronous data

Treat events as a collection

- An array whose items arrive asynchronously over time

Are a proposed feature for ES 2016

Use Reactive Extensions (RxJS)

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

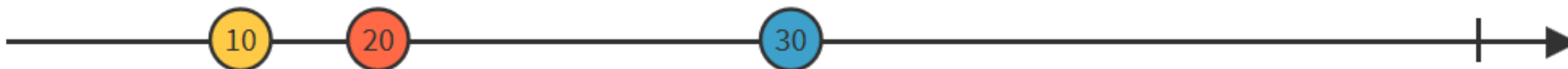
Examples: map, filter, take, merge, ...

Observables

Interactive diagrams of Rx Observables



```
map(x => 10 * x)
```



Promise vs Observable

Promise

Provides a single future value

Not lazy

Not cancellable

Observable

Emits multiple values over time

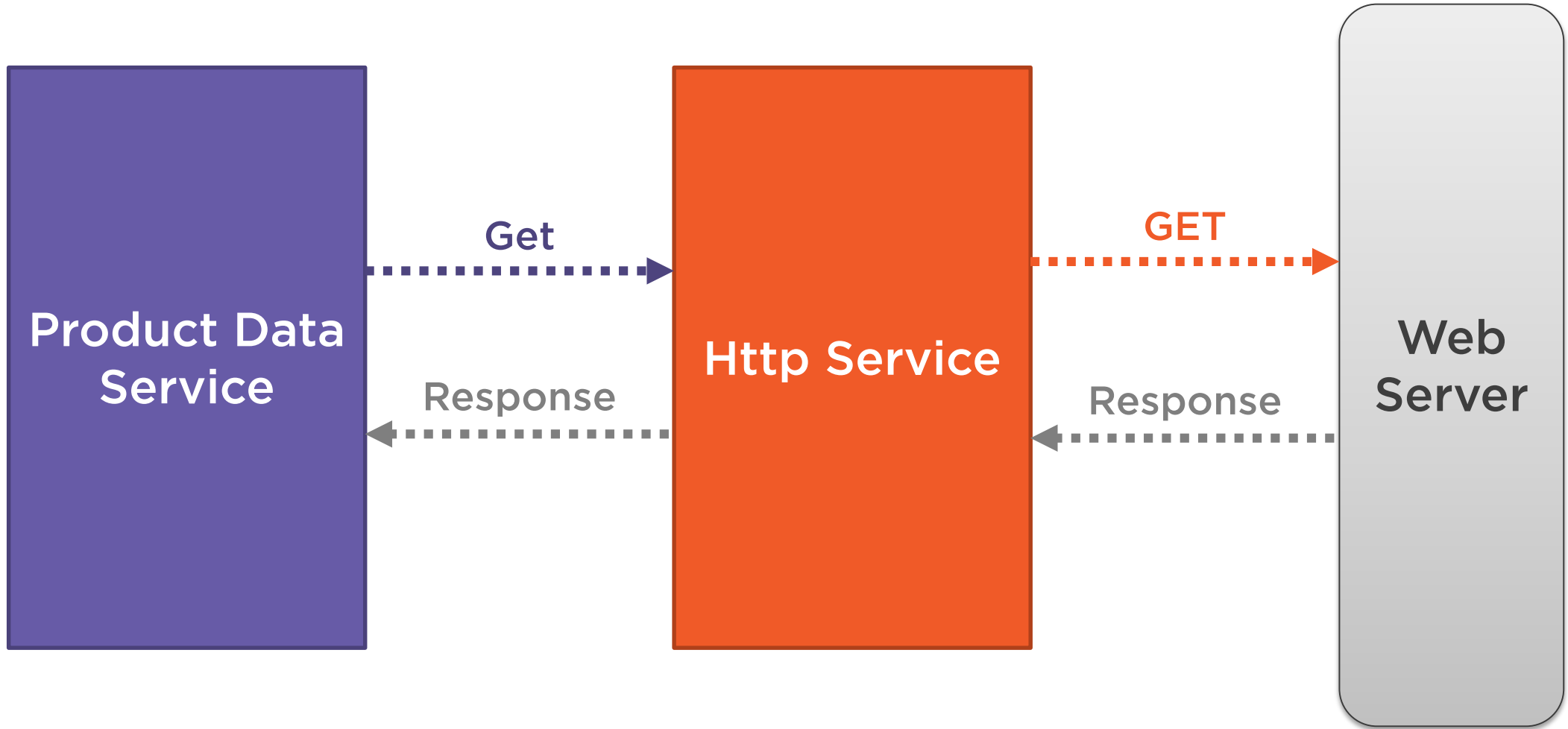
Lazy

Cancellable

Supports map, filter, reduce and similar operators



Sending an Http Request



Sending an Http Request

product.service.ts

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

@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: HttpClient) { }

  getProducts() {
    return this._http.get(this._productUrl);
  }
}
```

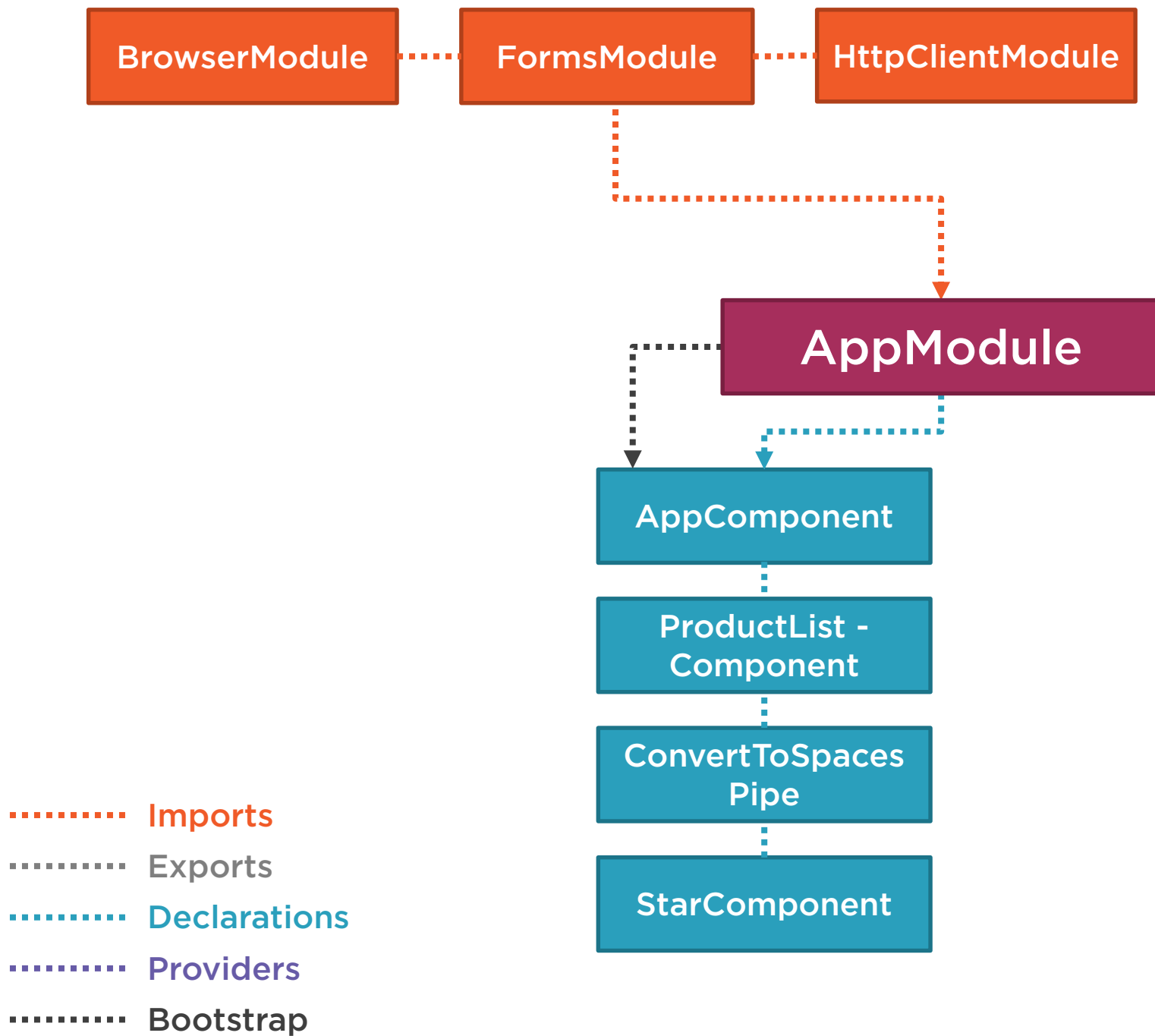
Registering the Http Service Provider

app.module.ts

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

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





Sending an Http Request

product.service.ts

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

@Injectable()
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()
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/Observable';

@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: HttpClient) { }

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

Exception Handling

product.service.ts

```
...
import { HttpClient, HttpResponseError } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/do';
...

getProducts(): Observable<IProduct[]> {
  return this._http.get<IProduct[]>(this._productUrl)
    .do(data => console.log('All: ' + JSON.stringify(data)))
    .catch(this.handleError);
}

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



Subscribing to an Observable

```
x.then(valueFn, errorFn)           //Promise
x.subscribe(valueFn, errorFn)       //Observable
x.subscribe(valueFn, errorFn, completeFn) //Observable
let sub = x.subscribe(valueFn, errorFn, completeFn)
```

product-list.component.ts

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



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

Summary



Observables and Reactive Extensions

Sending an Http Request

Exception Handling

Subscribing to an Observable



Application Architecture

