Going Reactive

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE
@deborahkurata | blogs.msmvps.com/deborahk/

Going Reactive

Focus on async data streams

Leverage RxJS operators

React to actions

Module Overview

Working with the async pipe

Handling errors

Improving change detection

Declarative pattern for data retrieval

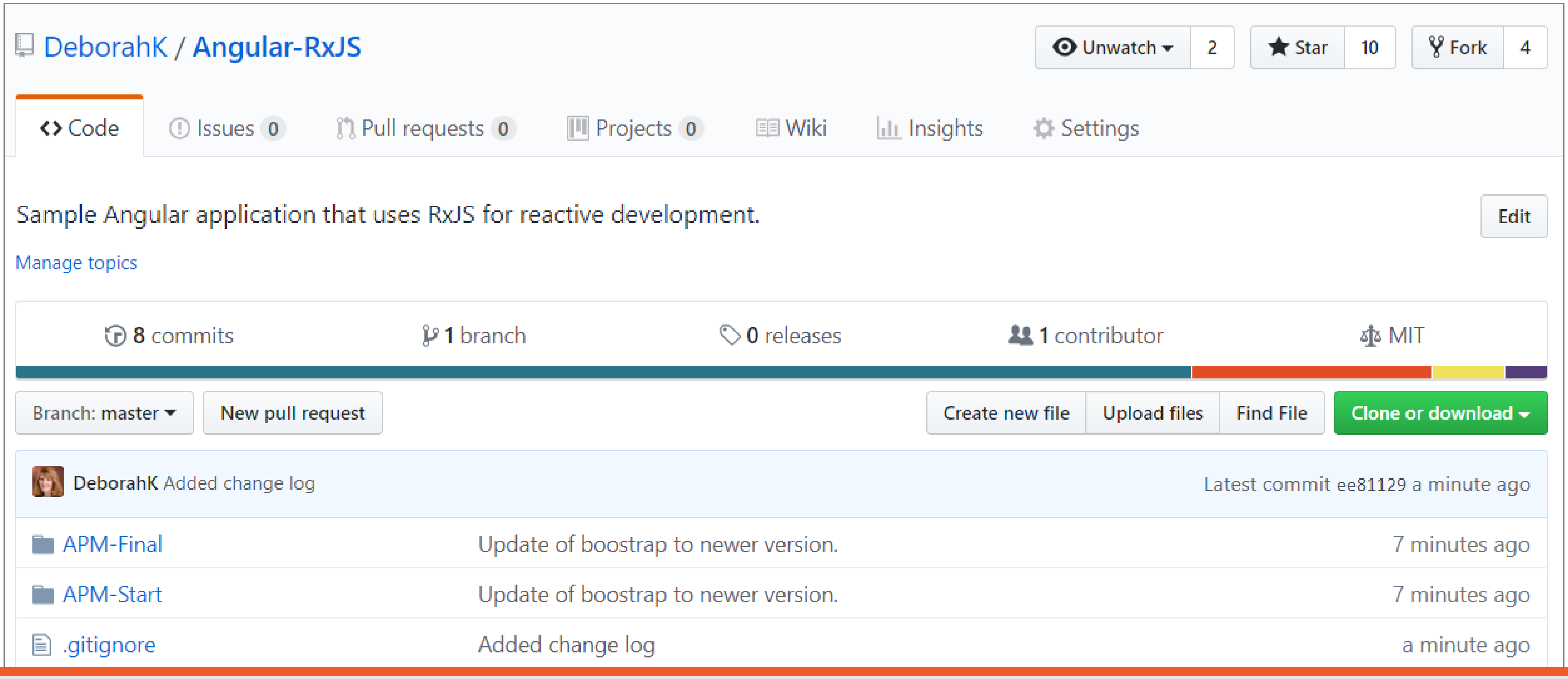
RxJS Features

`catchError`

`EMPTY`

`throwError`

GitHub Repository


DeborahK / Angular-RxJS

Unwatch 2 | Star 10 | Fork 4

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Sample Angular application that uses RxJS for reactive development.

Edit

Manage topics

8 commits | 1 branch | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

DeborahK Added change log — Latest commit ee81129 a minute ago

APM-Final	Update of boostrap to newer version.	7 minutes ago
APM-Start	Update of boostrap to newer version.	7 minutes ago
.gitignore	Added change log	a minute ago

https://github.com/DeborahK/Angular-RxJS

Async Pipe

```
"products$ | async"
```

Subscribes to the Observable when component is initialized

Returns each emitted value

When a new item is emitted, component is marked to be checked for changes

Unsubscribes when component is destroyed

Common Pattern with an Async Pipe

Product List Component

```
products: Product[] = [];

constructor(private productService: ProductService) { }

ngOnInit() {
 this.productService.getProducts()
  .subscribe(products => this.products = products);
}
```

Product List Component

```
products$: Observable<Product[]>;

constructor(private productService: ProductService) { }

ngOnInit() {
 this.products$ = this.productService.getProducts();
}
```

Template with an Async Pipe

Product List Template

```
<div *ngIf="products">

<table>
  <tr *ngFor="let product of products">
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>
  </tr>
</table>
```

Product List Template

```
<div *ngIf="products$ | async as products">

<table>
  <tr *ngFor="let product of products">
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>
  </tr>
</table>
```

Handling Errors

Catch Observable errors

Error stops the Observable

It won't emit any more items

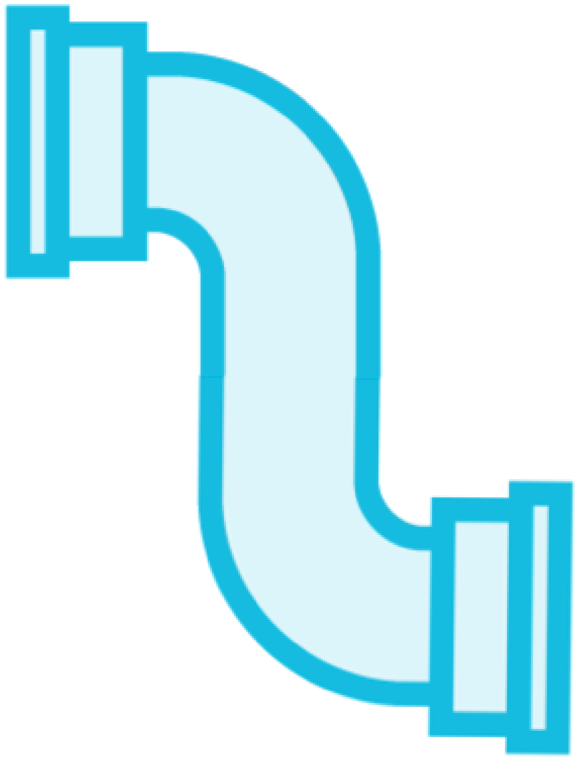
We can't use it anymore

Handling Errors

Catch and Replace

Catch and Rethrow

RxJS Operator: `catchError`



Catches any errors that occur on an Observable

```
catchError(this.handleError)
```

Used for catching errors and

- Rethrowing an error
- Or replacing the errored Observable to continue after an error occurs

Replacing an Errored Observable

An Observable created from hard-coded or local data

An Observable that emits an empty value or empty array

The EMPTY RxJS constant

Catch and Replace

Product Service

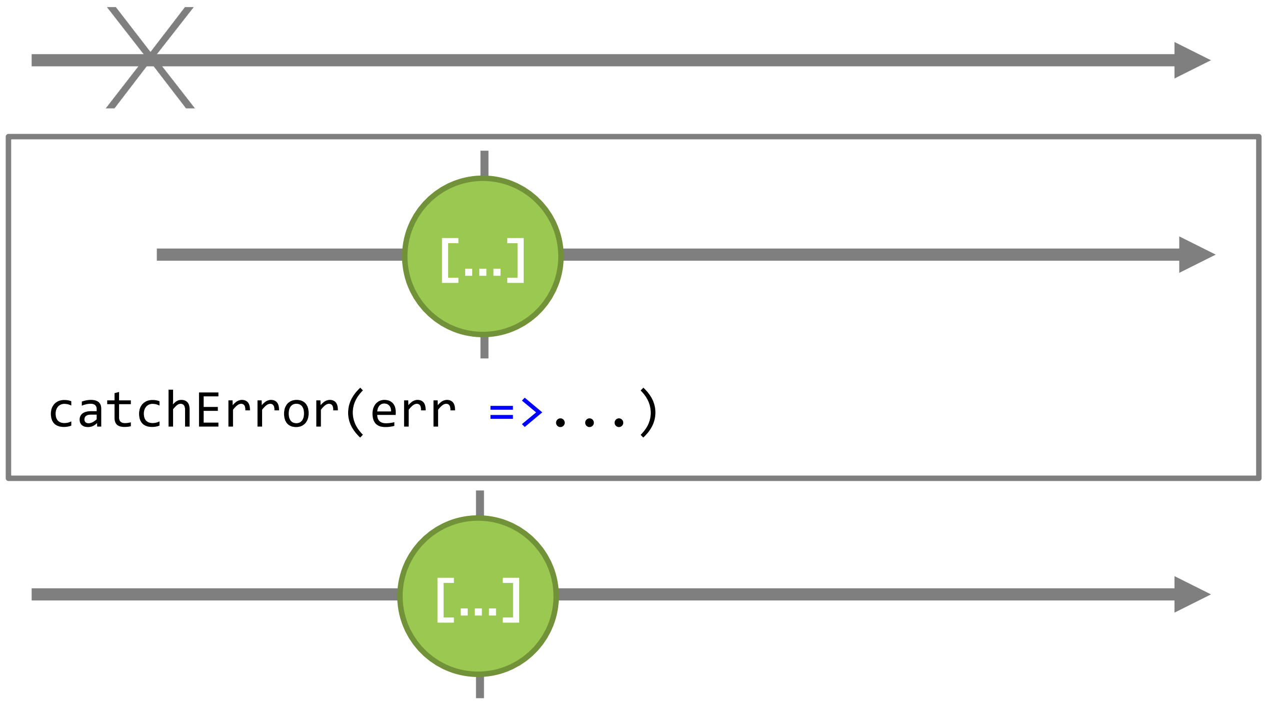
```
return this.http.get<Product[]>(this.productsUrl)
  .pipe(
    catchError(err => {
      console.error(err);
      return of([{ id: 1, productName: 'cart'},
                 { id: 2, productName: 'hammer'}]);
  });
```

Product List Component

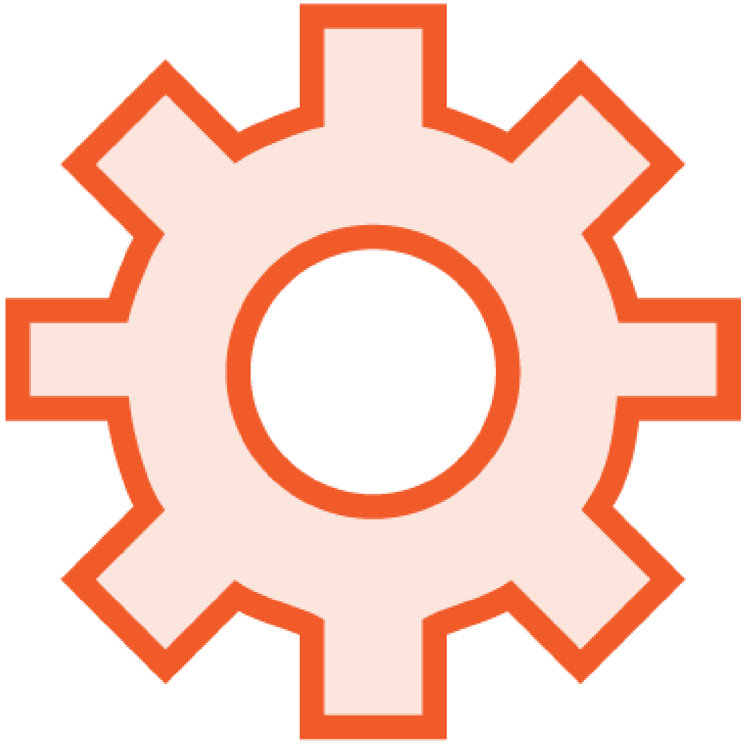
```
ngOnInit() {
 this.productService.getProducts()
  .subscribe(
    products => this.products = products,
    err => this.errorMessage = err
  );
}
```

Marble Diagram: catchError

```
return this.http.get<Product[]>(this.url)
 .pipe(
  catchError(err => {
   console.error(err);
   return of(
    [{ id: 1, productName: 'cart'},
     { id: 2, productName: 'hammer'}
    ]);
  })
 );
```



RxJS Operator: `catchError`



`catchError` is an error handling operator

- Takes in an input stream, subscribes
- Creates an output stream

When a source item is emitted

- Item is emitted to the output stream

If an error occurs

- Catches the error
- Unsubscribes from the input stream
- Returns a replacement Observable
- Optionally rethrows the error

Handling Errors

Catch and Replace

Catch and Rethrow

Catch and Rethrow

Product Service

```
return this.http.get<Product[]>(this.productsUrl)
  .pipe(
    catchError(err => {
      console.error(err);
      return throwError(err);
  });
```

RxJS Creation Function: `throwError`

Creates an Observable that emits no items

And immediately emits an error notification

```
throwError(err)
```

Used for

- Propagating an error

RxJS Creation Function: `throwError`

throwError is a creation function

Creates an Observable that emits no items

- Observable<never>

Immediately emits an error notification

Common Pattern with Error Handling

Product Service

```
private productsUrl = 'api/products';

getProducts(): Observable<Product[]> {
  return this.http.get<Product[]>(this.productsUrl)
    .pipe(
      catchError(this.handleError)
    );
}

private handleError(err) {
  // ...
  return throwError(errorMessage);
}
```

Error Handling

Product List Component

```
this.productService.getProducts()
  .subscribe(
    products => this.products = products,
    err => this.errorMessage = err
  );
```

Product List Component

```
this.products$ = this.productService.getProducts()
  .pipe(
    catchError(err => {
     this.errorMessage = err;
     return ???;
    })
  );
```

RxJS Constant: **EMPTY**

Returns an Observable that emits no items

And immediately emits a complete notification

```
return EMPTY;
```

Used for

- Returning an empty Observable

Error Handling

Product List Component

```
this.products$ = this.productService.getProducts()
  .pipe(
    catchError(err => {
     this.errorMessage = err;
     return EMPTY;
    })
  );
```

Demo

Handling errors

Benefits of an Async Pipe

No need to subscribe

No need to unsubscribe

Improve change detection

Change Detection Strategies

Angular uses change detection to track changes to application data structures …

… So it knows when to update the UI with changed data

Uses the default chec

Every component is checked when:

- Any change is detected

mance by minimizing change detection cycles

Component is only checked when:

- @Input properties change
- Event emits
- A bound Observable emits

```
@Component({
 templateUrl: './product-list.component.html',
 changeDetection: ChangeDetectionStrategy.OnPush
})
```

Common Pattern

Product Service

```
private productsUrl = 'api/products';
getProducts(): Observable<Product[]> {
  return this.http.get<Product[]>(this.productsUrl)
    .pipe(
      catchError(this.handleError)
    );
}
```

Product List Component

```
ngOnInit() {
 this.products$ = this.productService.getProducts()
 .pipe(
    catchError(err => {
     this.errorMessage = err;
     return EMPTY;
    })
  );
}
```

Declarative Pattern

Product Service

```
private productsUrl = 'api/products';

products$ = this.http.get<Product[]>(this.productsUrl);
```

Product List Component

```
products$ = this.productService.products$;
```

Declarative Pattern with Error Handling

Product Service

```
private productsUrl = 'api/products';

products$ = this.http.get<Product[]>(this.productsUrl)
  .pipe(
     catchError(this.handleError)
  );
```

Product List Component

```
products$ = this.productService.products$
 .pipe(
    catchError(err => {
     this.errorMessage = err;
     return EMPTY;
    })
 );
```

Benefits of a Declarative Approach

Leverages the power of RxJS Observables and operators

Effectively combine streams

Easily share Observables

Readily react to actions

Checklist: Data Retrieval

Define the shape of the data

- Interface or Class

```
export interface Product {
  id: number;
  productName: string;
  productCode: string;
  categoryId: number;
  description: string;
}
```

Checklist: Data Retrieval

Build a service

- Set a property to the Observable returned from http.get
- Use the type argument to map the response to the desired shape
- When the response is received, it's emitted and the Observable completes
- Pipe through desired operators

```
private productsUrl = 'api/products';

products$ = this.http.get<Product[]>(this.productsUrl)
  .pipe(
     catchError(this.handleError)
  );
```

Checklist: Data Retrieval

In a component, assign the service property to a local property

```
products$ = this.productService.products$
  .pipe(
     catchError(err => {
       this.errorMessage = err;
       return EMPTY;
     })
  );
```

Use OnPush change detection

```
@Component({
 templateUrl: './product-list.component.html',
 changeDetection: ChangeDetectionStrategy.OnPush
})
```

Checklist: Data Retrieval

In the template, use an async pipe

```
<div *ngIf="products$ | async as products">

<table>
  <tr *ngFor="let product of products">
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>
  </tr>
</table>
```

Checklist: Handling Errors

Catch and replace

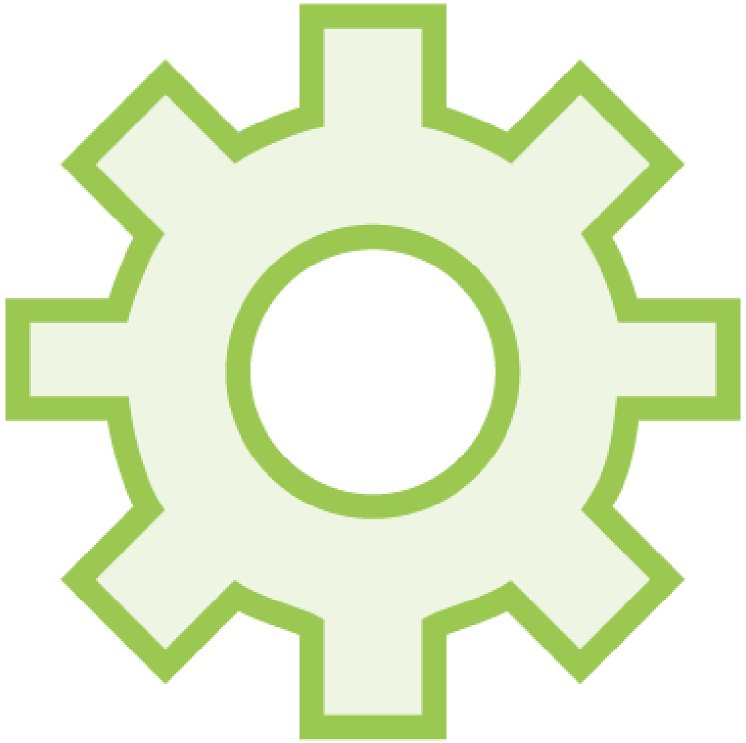
- An Observable that emits an alternate set of data
- An Observable that emits an empty set
- EMPTY

```
catchError(err => {
   this.errorMessage = err;
   return EMPTY;
})
```

Catch and rethrow

```
catchError(err => {
   console.error(err);
   return throwError(err);
})
```

RxJS Features



catchError: Catches any error and replaces the error Observable with a new Observable

throwError: Creates an Observable that emits no items and immediately emits an error notification

```
products$ = this.http.get<Product[]>(this.productsUrl)
  .pipe(
    catchError(err => {
      console.error(err);
      return throwError(err);
  });
```