Angular Hands-on Workshop

Ervin Suhanko
Front End Software Engineer,
Team Lead & Mentor

Prerequisite

- Basic understanding of HTML, CSS, and JavaScript
- Basic understanding of Programming
- Familiarity with ES6 and Typescript is helpful

5th – 6th – 7th – 8th – 9th Sep'22 | 9:30 – 10:30 AM EST

Agenda

Day 1	Day 2	Day 3	Day 4	Day 5
Angular Introduction and Framework Overview	Modules	Routing	RxJS Essentials	Tips, Tricks and Best Practices
Angular CLI (Command-line Interface tool)	Components and Data flow	Pipes and Directives	Services and HTTP	
Angular Project Structure		Reactive Forms		
TypeScript Essentials				

RxJS

- A library for composing data using observable sequences
- And transforming that data using operators
- Angular uses Reactive Extensions for working with data - Especially asynchronous data
- RxJS provides an implementation of the Observable type

RxJS

Synchronous vs Asynchronous

- **Synchronous** means that you call a web service (or function or whatever) and wait until it returns - all other code execution and user interaction is stopped until the call returns.

- **Asynchronous** means that you do not halt all other operations while waiting for the web service call to return. Other code executes and/or the user can continue to interact with the page (or program UI).

- HTTP requests are asynchronous: request and response

RxJS

Observables vs Promise

Observables	Promises
Emit multiple values over **a period of time.**	Emit a single value at a time.
Are lazy: they're not executed **until we subscribe** to them using the subscribe() method.	Are not lazy: execute immediately after creation.
Have subscriptions that are **cancellable** using the unsubscribe() method, which stops the listener from receiving further values.	Are not cancellable.
Provide the map for forEach, filter, reduce, retry, and retryWhen **operators**.	Don't provide any operations.

RxJS

How to create new Observable

From a promise

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

// Create an Observable out of a promise
const data = from(fetch('/api/endpoint'));
// Subscribe to begin listening for async result
data.subscribe({
  next(response) { console.log(response); },
  error(err) { console.error('Error: ' + err); },
  complete() { console.log('Completed'); }
});
```

RxJS

How to create new Observable

From an event

```
import { fromEvent } from 'rxjs';

const el = document.getElementById('my-element')!;

// Create an Observable that will publish mouse movements
const mouseMoves = fromEvent<MouseEvent>(el, 'mousemove');

// Subscribe to start listening for mouse-move events
const subscription = mouseMoves.subscribe(evt => {
  // Log coords of mouse movements
  console.log(`Coords: ${evt.clientX} X ${evt.clientY}`);

  // When the mouse is over the upper-left of the screen,
  // unsubscribe to stop listening for mouse movements
  if (evt.clientX < 40 && evt.clientY < 40) {
    subscription.unsubscribe();
  }
});
```

RxJS

How to create new Observable

From a counter

```
import { interval } from 'rxjs';

// Create an Observable that will publish a value on an interval
const secondsCounter = interval(1000);
// Subscribe to begin publishing values
const subscription = secondsCounter.subscribe(n =>
  console.log(`It's been ${n + 1} seconds since subscribing!`));
```

RxJS

Operators

- Operators are functions that build on the observables foundation to enable sophisticated manipulation of collections.
- RxJS defines operators such as map(), filter(), concat(), and flatMap().
- You can use *pipes* to link operators together. Pipes let you combine multiple functions into a single function

RxJS

Operators

```
// Create a function that accepts an Observable.
const squareOddVals = pipe(
  filter((n: number) => n % 2 !== 0),
  map(n => n * n)
);
```

RxJS

Common Operators

AREA	OPERATORS
Creation	**from**, fromEvent, **of**
Combination	**combineLatest**, concat, **merge**, startWith , withLatestFrom, zip
Filtering	**debounceTime**, distinctUntilChanged, filter, take, takeUntil
Transformation	bufferTime, **concatMap**, **map**, **mergeMap**, scan, switchMap
Utility	**tap**
Multicasting	share

RxJS

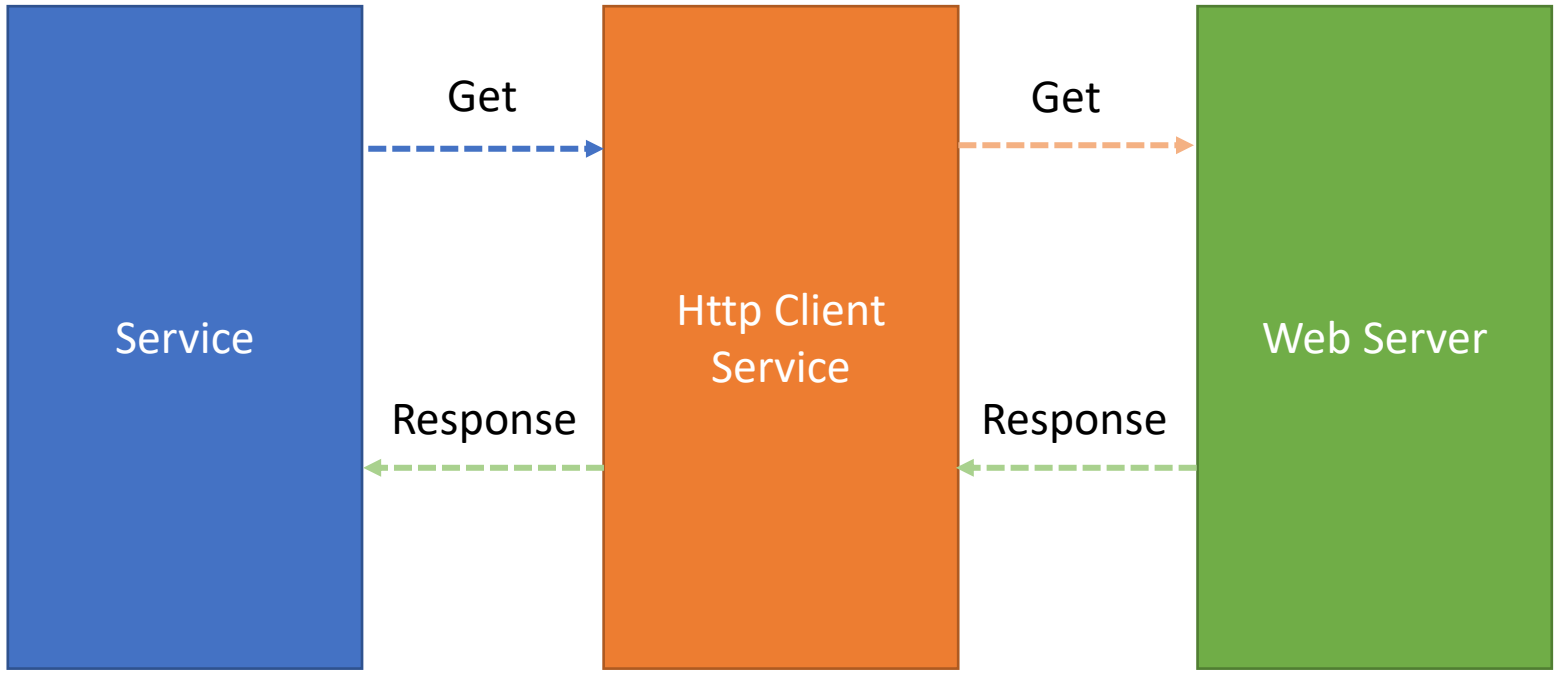
Error Handling

In addition to the error() handler that you provide on subscription, RxJS provides the **catchError** operator that lets you handle known errors in the observable recipe.

```
// Return "response" from the API. If an error happens,
// return an empty array.
const apiData = ajax('/api/data').pipe(
  map((res: any) => {
    if (!res.response) {
      throw new Error('Value expected!');
    }
    return res.response;
  }),
  catchError(() => of([]))
);

apiData.subscribe({
  next(x: T) { console.log('data: ', x); },
  error() { console.log('errors already caught... will not run'); }
});
```

Server Communication



Server Communication

HTTP Module

- Simplifies usage of the XHR and JSONP APIs
- API conveniently matches RESTful verbs
- Returns an observable

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
```

Server Communication

HTTP Module Methods

- **request**: performs any type of http request
- **get**: performs a request with GET http method
- **post**: performs a request with POST http method
- **put**: performs a request with PUT http method
- **delete**: performs a request with DELETE http method
- **patch**: performs a request with PATCH http method
- **head**: performs a request with HEAD http method

Server Communication

HTTP Module Methods

```
loadItems() {
  return this.http.get(BASE_URL);
}

createItem(item: Item) {
  return this.http.post(`${BASE_URL}`, item);
}

updateItem(item: Item) {
  return this.http.patch(`${BASE_URL}${item.id}`, item);
}

deleteItem(item: Item) {
  return this.http.delete(`${BASE_URL}${item.id}`);
}
```

Server Communication

Observable.subscribe

- We finalize an observable stream by subscribing to it
- The subscribe method accepts three event handlers
  - **onNext** is called when new data arrives
  - **onError** is called when an error is thrown
  - **onComplete** is called when the stream is completed

Server Communication

Observable.subscribe

```
loadItems() {
  return this.http.get(BASE_URL);
}
```

```
export class ItemsComponent {
  items: Item[];
  selectedItem: Item;

  constructor(
    private itemsService: ItemsService
  ) {}

  getItems() {
    this.itemsService.loadItems()
      .subscribe((items: Item[]) => this.items = items);
  }
}
```

Server Communication

Headers

- HttpClient methods have an optional parameter which contains options for configuring the request
- This options object has a headers property which is an HttpHeaders object
- We can use the HttpHeaders object to set additional parameters like Content-Type

Server Communication

Headers

```
uploadFile(file: File, url: string) {
  const headers = new HttpHeaders().set('Content-Type', file.type);
  const options = {
    headers,
    reportProgress: true,
    observe: 'events',
    responseType: 'text' as 'text',
  };

  const req = new HttpRequest('PUT', url, file, options);
  return this.http.request(req);
}
```

Server Communication

Error Handling

- We use the catchError operator to well… catch the error
- We can create a general error handling method that can process the error internally while surfacing a user friendly error via throwError

Server Communication

Error Handling

```
private handleError(error: HttpErrorResponse) {
  if (error.error instanceof ErrorEvent) {
    // A client-side or network error occurred. Handle it accordingly.
    console.error('An error occurred:', error.error.message);
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may contain clues as to what went wrong,
    console.error('Error Status: ', error.status);
    console.error('Error Details: ', error.error);
  }
  // return an observable with a user-facing error message
  return throwError('Something bad happened; please try again later.');
};
```

```
loadItems() {
  return this.http.get(BASE_URL)
  .pipe(
    catchError(this.handleError)
  );
}
```

Demo

HTTP