

NETWORK COMMUNICATION

LESSON 03

SWAFE-01

CLIENT-SIDE APPLICATIONS

OVERVIEW

- Most **dynamic** web applications gets data from other **sources**
- Hypertext Transfer Protocol (HTTP) and WebSocket Protocol is commonly used to serve data to web applications

CLIENT-SERVER MODEL

Client—server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

Client-server model—*Wikipedia*

HYPERTEXT TRANSFER PROTOCOL

THE HTTP/1.1 SPECIFICATION

- Defined in IETF RFC 2616
 - IETF – International Engineering Task Force
 - RFC – Request For Comments
- IETF is the Internet standards body
 - Developing open standards through open processes
 - An international community of network designers, operators, vendors, and researchers
 - Concerned with the evolution of the Internet architecture and the smooth operation of the Internet
- An RFC is a publication format used by IETF
 - Protocols, procedures and programs
 - Concepts
 - Meeting notes

HTTP METHODS

- **POST** —send data to the server
- **GET** —retrieve a resource from the server
- **HEAD** —retrieve resource metadata from the server
- **PUT** —send data to the server and update an existing entity (overwrite)
- **DELETE** —remove a resource on the server
- **OPTIONS** —get information about the communication options available

HTTP RESPONSE CODES

- **1xx** **Informational**—provisional response
- **2xx** **Successful**—indicates that the client's request was successfully received, understood, and accepted
- **3xx** **Redirection**—further action needs to be taken by the user agent in order to fulfill the request
- **4xx** **Client error**—intended for cases in which the client seems to have erred
- **5xx** **Server error**—indicate cases in which the server is aware that it has erred or is incapable of performing the request

HTTP SPECIFICATIONS

- Initial releases
 - [RFC 2068](#) Hypertext Transfer Protocol -- HTTP/1.1 *January 1997*
 - [RFC 2616](#) Hypertext Transfer Protocol -- HTTP/1.1 *June 1999*
- Updated and split up in June 2014
 - [RFC 7230](#) HTTP/1.1: Message Syntax and Routing
 - [RFC 7231](#) HTTP/1.1: Semantics and Content
 - [RFC 7232](#) HTTP/1.1: Conditional Requests
 - [RFC 7233](#) HTTP/1.1: Range Requests
 - [RFC 7234](#) HTTP/1.1: Caching
 - [RFC 7235](#) HTTP/1.1: Authentication
- Other versions
 - [RFC 7540](#) Hypertext Transfer Protocol Version 2 (HTTP/2) (*May 2015*)
 - [draft-ietf-quic-http-34](#) Hypertext Transfer Protocol Version 3 (HTTP/3) (*February 2021*)

HTTP IN ANGULAR

OVERVIEW

- The ability to request **typed** response objects
- **Streamlined** error handling
- Request and response **interception**
- **Testability** features

HTTPCLIENTMODULE

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { HttpClientModule } from '@angular/common/http';
4
5 import { AppRoutingModule } from './app-routing.module';
6 import { AppComponent } from './app.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule,
15     HttpClientModule,
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
```

examples/lesson03-network-communication/projects/server-communication/src/app/app.module.ts

HttpClient

- Wraps HTTP requests in `Observable` objects
- Contains methods for some of the most used HTTP requests
 - `get()` returns a configured `GET` HTTP request
 - `post()` returns a configured `POST` HTTP request
 - `delete()` returns a configured `DELETE` HTTP request
 - `request()` returns a generic HTTP request
- Used in combination with `RxJS` operators to filter values and format to desired output
- Can be initiated with `subscribe()` or `async` pipe

SpaceService

```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Astronaut, LaunchVehicles } from 'lib-space';
4 import { Observable } from 'rxjs';
5 import { environment } from '../environments/environment';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class SpaceService {
11   rootUrl = `http://${environment.api.space.host}:${environment.api.space.port}`;
12
13   constructor(private http: HttpClient) { }
14
15   getAstronauts(): Observable<Astronaut[]> {
16     return this.http.get<Astronaut[]>(`${this.rootUrl}/astronauts`)
17   }
18
19   getLaunchVehicles(): Observable<LaunchVehicles[]> {
```

<examples/lesson03-network-communication/projects/server-communication/src/app/space.service.ts>

AppComponent

```
1 import { Component, OnInit } from '@angular/core';
2 import { LaunchVehicles } from 'lib-space';
3 import { Observable } from 'rxjs';
4 import { map } from 'rxjs/operators';
5 import { SpaceService } from './space.service';
6
7 @Component({
8   selector: 'app-root',
9   templateUrl: './app.component.html',
10  styleUrls: ['./app.component.scss']
11 })
12 export class AppComponent implements OnInit {
13
14   launchVehicles$: Observable<LaunchVehicles[]> | null = null;
15   astronauts$: Observable<string[]> | null = null;
16
17   constructor(private spaceService: SpaceService) { }
18
19   ngOnInit() {
```

<examples/lesson03-network-communication/projects/server-communication/src/app/app.component.ts>

TYPED RESPONSE

OVERVIEW

- **Consume** output more easily and obvious
- Response types can act at type **assertion** at compile time
 - It is **NOT** guaranteed that servers respond with the expected type
 - So be sure to have **proper** **error handling**
- RxJS operators
 - Use **operators** to transform/filter response data
 - Use the responses with **async** pipe in templates

RESPONSE TYPE FOR Astronaut

```
1 export interface Astronaut {  
2   name: string;  
3   nationality: string;  
4   organization: string;  
5   status?: string;  
6   born: number;  
7   died?: number;  
8   time_in_space?: number;  
9 }
```

examples/lesson03-network-communication/projects/lib-space/src/lib/astronaut.type.ts

REQUESTING DATA FROM A SERVER

```
1 import { Component, OnInit } from '@angular/core';
2 import { LaunchVehicles } from 'lib-space';
3 import { Observable } from 'rxjs';
4 import { map } from 'rxjs/operators';
5 import { SpaceService } from './space.service';
6
7 @Component({
8   selector: 'app-root',
9   templateUrl: './app.component.html',
10  styleUrls: ['./app.component.scss']
11 })
12 export class AppComponent implements OnInit {
13
14   launchVehicles$: Observable<LaunchVehicles[]> | null = null;
15   astronauts$: Observable<string[]> | null = null;
16
17   constructor(private spaceService: SpaceService) { }
18
19   ngOnInit() {
```

`examples/lesson03-network-communication/projects/server-communication/src/app/app.component.ts`

DISPLAY DATA IN TEMPLATES

```
1 <h2>Astronauts</h2>
2 <ul>
3   <li *ngFor="let astronaut of astronauts$ | async">
4     {{ astronaut }}
5   </li>
6 </ul>
7 <h2>Launch vehicles</h2>
8 <ul>
9   <li *ngFor="let launchVehicle of launchVehicles$ | async">
10    {{ launchVehicle | json }}
11  </li>
12 </ul>
```

examples/lesson03-network-communication/projects/server-communication/src/app/app.component.html

SAME-ORIGIN SECURITY MODEL & CROSS-ORIGIN RESOURCE SHARING

OVERVIEW

- The Cross-Origin Resource Sharing (CORS) standard let servers describe which origins are **permitted** from the browser
 - Resources are only available at the same origin per default
- CORS failures results in errors, but for security reasons, **specifics** about the error are not available to JavaScript
 - Detailed insights can compromise our application

SAME-ORIGIN POLICY

- The same-origin policy is a critical **security mechanism** that restricts how documents and scripts loaded by one origin can interact with a resource from another origin
- Definition of an origin
 - A origin is a tuple comprised of `[protocol, host, port]`
 - `protocol` —The protocol used: `http://` , `https://` , etc.
 - `host` —The resource location: `swafe-01.dk`, `ece.au.dk`, etc.
 - `port` —The port number, if specified (`80` , `443` , etc.)
- Implemented by all major browsers (Google Chrome, Firefox, Microsoft Edge, etc.)

CROSS-ORIGIN RESOURCE SHARING (CORS)

- The need to **relax** the same-origin policy
- CORS is a protocol
- **Simple** and **preflighted** requests
 - Simple requests (**GET** , **POST** , and **HEAD**) does not trigger a CORS preflight
 - Methods like **PUT** , **DELETE** , and **PATCH** trigger a CORS preflight
- Preflight requests happens when requests that causes **side-effects** on server data
- Set in the header of HTTP requests and responses

GET

```
1 GET /cors/astronauts HTTP/1.1
2 Host: localhost:3000
3 Origin: http://localhost:4200
4 Referer: http://localhost:4200/
5 Accept: application/json, text/plain, */*
6 Accept-Encoding: gzip, deflate, br
7
8 HTTP/1.1 200 OK
9 Access-Control-Allow-Origin: *
10 Content-Type: application/json; charset=utf-8
11 Content-Length: 742
12 Date: Fri, 10 Sep 2021 07:58:23 GMT
13
14 [{"name":"Neil Armstrong","nationality":"US","organization":"NASA","born":1930
```

PREFLIGHT REQUESTS

- Preflight request check if the host **supports** CORS
 - Sent as an **OPTIONS** using three HTTP request headers:
 - **Access-Control-Request-Method** —defines what HTTP method(s) will be used for the actual request
 - **Access-Control-Request-Headers** —defines what HTTP headers the client might send with the request
 - **Origin** —indicates where the request originates from
- A preflight response will contain a list of **allowed** methods and origin
- Preflight request are **automatically** issued by the browser
 - In normal cases, this means that frontends developers do not need to craft such requests themselves 🥳

PREFLIGHT REQUEST

```
1 OPTIONS /cors/astronauts HTTP/1.1
2 Host: localhost:3000
3 Accept: */*
4 Access-Control-Request-Method: DELETE
5 Origin: http://localhost:4200
6 Referer: http://localhost:4200/
7 Accept-Encoding: gzip, deflate, br
8 Accept-Language: en-US,en;q=0.9,da-DK;q=0.8,da;q=0.7,fr;q=0.6
9
10 HTTP/1.1 204 No Content
11 Access-Control-Allow-Origin: *
12 Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,POST,DELETE
13 Vary: Access-Control-Request-Headers
14 Content-Length: 0
15 Date: Fri, 10 Sep 2021 08:25:35 GMT
```

PROXYING TO A BACKEND SERVER

- Proxying support from `webpack` development server is available
- Follow these steps to set up:
 - Create a file `proxy.conf.json` in the project `src` folder
 - Add the `proxyConfig` option to the `serve target`
 - Check service URLs in the code
- **Remember** to relaunch the application when changing proxy configuration (rerun `ng serve`)

proxy.conf.json

```
1 {  
2   "/api": {  
3     "target": "http://localhost:3000",  
4     "secure": false  
5   }  
6 }
```

examples/lesson03-network-communication/projects/proxying/src/proxy.conf.json

```
1  ... ,  
2  "serve": {  
3    "builder": "@angular-devkit/build-angular:dev-server",  
4    "configurations": {  
5      "production": {  
6        "browserTarget": "proxying:build:production"  
7      },  
8      "development": {  
9        "browserTarget": "proxying:build:development",  
10       "proxyConfig": "examples/lesson03-network-communication/projects/proxyin  
11     }  
12   },  
13   "defaultConfiguration": "development"  
14 }, ...
```

examples/lesson03-network-communication/angular.json

ERROR HANDLING

OVERVIEW

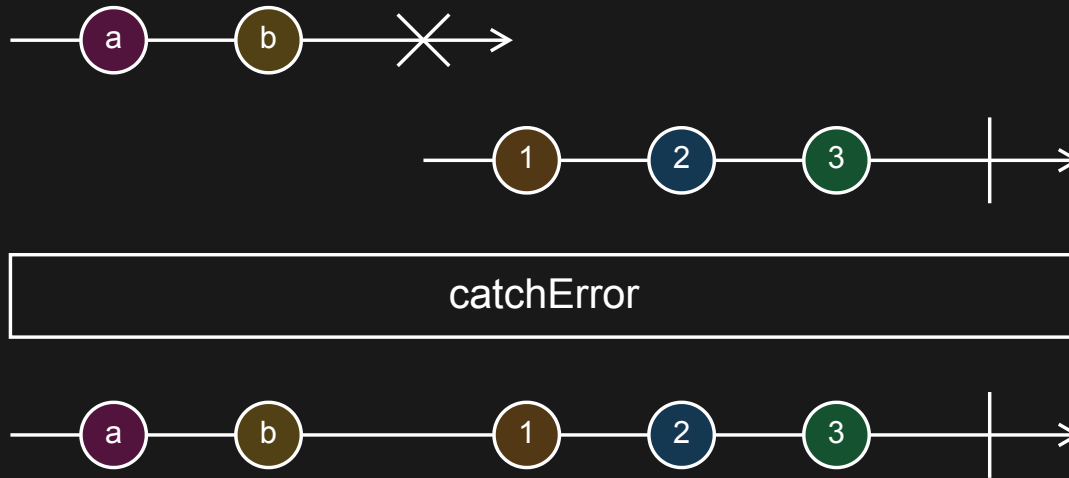
- If the request fails, HttpClient returns an `HttpErrorResponse` error object instead of a successful response
- Two types of errors can occur:
 - **Error responses** The backend might reject a request and return a response with status code (`4xx` or `5xx`)
 - **Client-side errors** Network errors or unhandled exceptions thrown in RxJS operators
- Client-side error will have `status` set to `0`, error responses will have the HTTP status code (such as `404` Not Found, `500` Internal Server Error, etc.

RXJS OPERATORS

- `catchError` catches errors on the `Observable` to be handled by returning a new observable or throwing an error
- `retry` retries the `Observable` a predefined number of times. Defaults to `Infinity`
- `retryWhen` retries the `Observable` based on custom criteria

catchError

Catch errors that happens inside the stream



catchError

Catch errors that happens inside the stream

```
1 import { HttpResponse } from '@angular/common/http';
2 import { Component, OnInit } from '@angular/core';
3 import { AstronautResponse } from 'projects/lib-space/src/lib/astronaut-respon
4 import { Observable, of } from 'rxjs';
5 import { catchError } from 'rxjs/operators';
6 import { FaultyService } from '../faulty.service';
7
8 @Component({
9   selector: 'app-catch-error',
10  templateUrl: './catch-error.component.html',
11  styleUrls: ['./catch-error.component.scss']
12 })
13 export class CatchErrorComponent implements OnInit {
14   observable$: Observable<AstronautResponse> | null = null
15   error = ''
16
17   constructor(private faultyService: FaultyService) { }
18
19   ngOnInit(): void {
```

examples/lesson03-network-communication/projects/error-handling/src/app/catch-error/catch-error.component.ts

retry

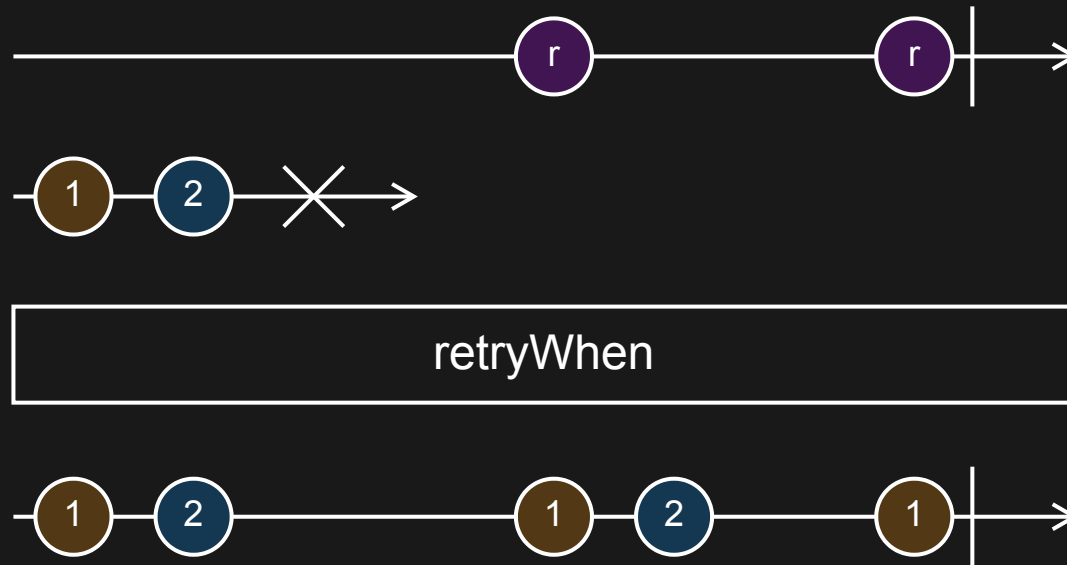
Retry an observable sequence should an error occur

```
1 import { Component, OnInit } from '@angular/core';
2 import { Astronaut } from 'lib-space';
3 import { Observable, throwError } from 'rxjs';
4 import { catchError, retry } from 'rxjs/operators';
5 import { FaultyCall } from '../faulty-call.type';
6 import { FaultyService } from '../faulty.service';
7 import { SpaceError } from '../space-error.type';
8
9 @Component({
10   selector: 'app-retry',
11   templateUrl: './retry.component.html',
12   styleUrls: ['./retry.component.scss']
13 })
14 export class RetryComponent implements OnInit {
15
16   observable$: Observable<Astronaut[] | SpaceError> | null = null
17   faults: FaultyCall[] = []
18
19   constructor(private faultyService: FaultyService) { }
```

examples/lesson03-network-communication/projects/error-handling/src/app/retry/retry.component.ts

retryWhen

Retry an observable sequence on error based on custom criteria



retryWhen

Retry an observable sequence on error based on custom criteria

```
1 import { Component, OnInit } from '@angular/core';
2 import { Astronaut } from 'lib-space';
3 import { Observable, throwError, timer } from 'rxjs';
4 import { catchError, delayWhen, retryWhen } from 'rxjs/operators';
5 import { FaultyCall } from '../faulty-call.type';
6 import { FaultyService } from '../faulty.service';
7 import { SpaceError } from '../space-error.type';
8
9 @Component({
10   selector: 'app-retry-when',
11   templateUrl: './retry-when.component.html',
12   styleUrls: ['./retry-when.component.scss']
13 })
14 export class RetryWhenComponent implements OnInit {
15
16   observable$: Observable<Astronaut[] | SpaceError> | null = null
17   faults: FaultyCall[] = []
18
19   constructor(private faultyService: FaultyService) { }
```

examples/lesson03-network-communication/projects/error-handling/src/app/retry-when/retry-when.component.ts

JSON WEB TOKENS

OVERVIEW

- JSON Web Token (JWT) is a compact **claims representation** format
- A JWT consists of three partitions
 - **Header** contains type and signing information
 - **Payload** contains claims and other user information
 - **Signature** contains a hash of **header** and **payload**
- Relevant IETF standards
 - RFC7519 JSON Web Token
 - RFC8725 JSON Web Token Best Current Practices

JWT CLAIMS

- **aud** Audience—what resource is this token intended for?
- **iss** Issuer—who issued the token?
- **iat** Issued At—identifies the time the claim was issued
- **nbf** Not Before—identifies the time before which the JWT must not be accepted
- **exp** Expires—identifies the time on or after which the JWT must not be accepted

INTERCEPTION

OVERVIEW

- Request and response transformation
 - Before sending a request to the server
 - Before receiving a response in the application
- Perform implicit tasks
 - Such as setting an authentication token for all requests sent to the server
 - Alternatively implemented explicitly for every method call
- Multiple interceptors forms a forward-and-backwards chain of request/response handlers

HttpInterceptor

- All interceptors is **managed** by Angular's DI system (just like services)
- Interceptors are dependencies of `HttpClient`
 - Therefore, they must be provided in the **same** injector (or a parent of the injector) that provides `HttpClient`
 - If added after `HttpClient` is created, they are ignored
- Create a "barrel" file in `/app` to keep track of injectors
 - Contains an array of `HttpInterceptor` objects
 - Use `HTTP_INTERCEPTORS` multi-provider token when providing interceptors

HttpInterceptor

- `HttpRequest` and `HttpResponse` are immutable
 - Cannot be modified after creation
 - Use the `clone` method
- Check context with `HttpContext`
 - If only some requests/responses should be handled or passed on without modification

INTERCEPTOR

```
1 import { Injectable } from '@angular/core';
2 import {
3   HttpRequest,
4   HttpResponse,
5   HttpEvent,
6   HttpInterceptor,
7   HttpContextToken,
8   HttpContext
9 } from '@angular/common/http';
10 import { Observable } from 'rxjs';
11 import { AuthService } from '../service/auth.service';
12 import { switchMap } from 'rxjs/operators';
13
14 const AUTH = new HttpContextToken<boolean>(() => false)
15
16 export function auth() {
17   return new HttpContext().set(AUTH, true)
18 }
19
```

examples/lesson03-network-communication/projects/interception/src/app/http-interceptors/auth-token.interceptor.ts

SERVICE

```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Astronaut } from 'lib-space';
4 import { Observable } from 'rxjs';
5 import { auth } from '../http-interceptors/auth-token.interceptor';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class SpaceService {
11
12   rootUrl = `http://localhost:3000/auth`
13
14   constructor(private http: HttpClient) { }
15
16   astronauts(): Observable<Astronaut[]> {
17     return this.http.get<Astronaut[]>(`${this.rootUrl}/astronauts`, {
18       context: auth()
19     })
20   }
21 }
```

examples/lesson03-network-communication/projects/interception/src/app/service/space.service.ts

CALL FROM CLASS

```
1 import { Component, OnInit } from '@angular/core';
2 import { Observable } from 'rxjs';
3 import { SpaceService } from '../service/space.service';
4 import { Astronaut } from 'lib-space';
5
6 @Component({
7   selector: 'app-root',
8   templateUrl: './app.component.html',
9   styleUrls: ['./app.component.scss']
10 })
11 export class AppComponent implements OnInit {
12   token$: Observable<string> | null = null;
13   astronauts$: Observable<Astronaut[]> | null = null;
14
15   constructor(private spaceService: SpaceService) { }
16
17   ngOnInit() {
18     this.astronauts$ = this.spaceService.astronauts()
19   }
```

examples/lesson03-network-communication/projects/interception/src/app/app.component.ts

