Angular 6

Lesson 08 : Http Requests / Observables

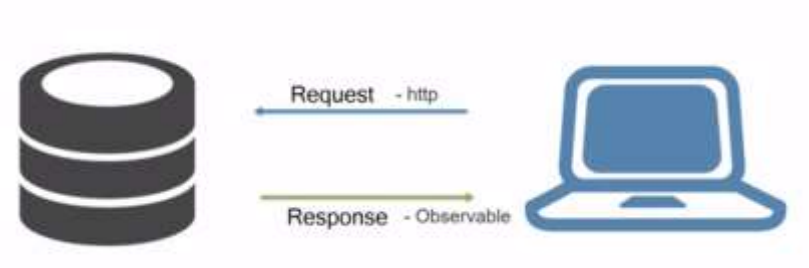
Capgemini

Lesson Objectives

- HTTP Requests
- Sending HTTP Requests
- Transform Responses with Observable Operators (map())
- Catching Http Errors
- Pipe(), map(), catchError()
- Interceptors
- Basics of Observables & Promises
- Building & Using a Custom Observable

HTTP Requests

- Angular applications often obtain data using http
- Application issues http get requests to a web server which returns http response-Observable to the application.
- Application then processes that data



Add instructor notes here.

Introducing RxJs

- RxJs stands for Reactive Extensions for Javascript, and its an implementation of Observables for Javascript.
- It is a ReactiveX library for JavaScript.
- It provides an API for asynchronous programming with observable streams.
- ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming.
- Observable is a RxJS API. Observable is a representation of any set of values over any amount of time. All angular Http methods return instance of Observable. Find some of its operators.
- map: It applies a function to each value emitted by source Observable and returns finally an instance of Observable.
- catch: It is called when an error is occurred. catch also returns Observable.

The RxJS library is quite large.

It's up to us to add the operators we need

// Add map operator
https://cdnjs.cloudflare.com/ajax/libs/rxjs/4.1.0/rx.map

// Add all operators to Observable
https://cdnjs.cloudflare.com/ajax/libs/rxjs/4.1.0/rx.all.js

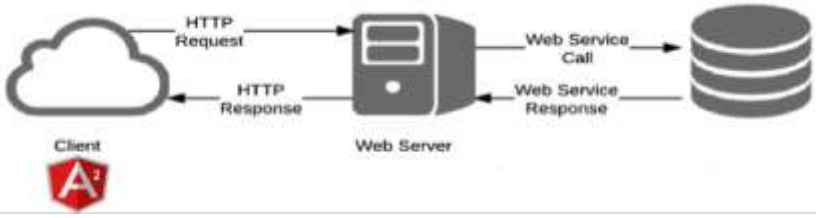
// Add map operator
import 'rxjs/add/operator/map'*;*

// Add all operators to Observable
import 'rxjs/Rx';

Add instructor notes here.

Angular HTTP

- Angular applications often obtain data using http
- Application issues http get requests to a web server which returns http response to the application.
- Application then processes that data



Add instructor notes here.

Http Class

- Performs http requests using `XMLHttpRequest` as the default backend.
- Http is available as an injectable class.
- Calling request returns an Observable which will emit a single Response when a response is received.
- To work with Http Class
  - Include Angular 2 Http script (http.dev.js) in index.html
  - Include script tag for the reactive extensions (Rx.js) in index.html
  - Register HTTP_PROVIDERS
  - Import RxJS

Http is not a part of angular2/core and it is not included in the main angul script file. It is an optional service available in its own library. It needs to added in the index file.

Angular http library has several services that assist with using http. To u http from any of the components we can register this angular service wi root component.

There are several services involved the angular http client library provides single constant that defines the set of service providers from the angul http library called **HTTP_Providers**

Similarly there are several features for reactive extensions such as the ma operator. To load them all in root component using import statement w need to use **import 'rxjs/Rx'**;. It tell to the module loader to load this libra but imports nothing. When the library is loaded its javascript is execute and for this particular library executing the JavaScript loads the full set observable operators that will need throughout application

Http methods
request(url, options)
get (url, options)
delete (url, options)
head (url, options)
post (url,body,options)
put (url,body,options)
patch(url,body,options)

Add instructor notes here.

Catch Operator

- Reacts to the error case of an Observable.
- Need to return a new Observable to continue with

```
export class UserProxy{
    constructor(private http :Http){}
    load(){
        return this.http
            .get('http://api.randomuser.me/10')
            .map(res =>res.json())
           .catch(this.logAndPassOn);
}
private logAndPassOn (error: Error) {
    console.error(error);
    return Observable.throw(error);
}
```

Add instructor notes here.

Communication with JSONP

- Angular provides us with a JSONP services which has the same API surface as the Http.
- Only difference that it restricts us to use GET requests only.
- JSONP service requires the JSONP_PROVIDERS.

Transform Responses with Observable Operators

- Operators are functions that build on the observables foundation to enable sophisticated manipulation of collections.
- For example, RxJS defines operators such as map(), filter(), concat(), and flatMap().
- Operators take configuration options, and they return a function that takes a source observable.
- When executing this returned function, the operator observes the source observable's emitted values, transforms them, and returns a new observable of those transformed values.

Transform Responses with Observable Operators

- Map operator

```
import { map } from 'rxjs/operators';
const nums = of(1, 2, 3);
const squareValues = map((val: number) => val * val);
const squaredNums = squareValues(nums);
squaredNums.subscribe(x => console.log(x));
 // Logs
// 1
// 4
// 9
```

Catching Http Errors

- In addition to the error() handler, RxJS provides the catchError operator that lets you handle known errors in the observable recipe.
- For instance,
  - suppose you have an observable that makes an API request and maps to the response from the server.
  - If the server returns an error or the value doesn't exist, an error is produced.
  - If you catch this error and supply a default value, your stream continues to process values rather than erroring out.

Catching Http Errors

- Here's an example of using the catchError operator

```
import { ajax } from 'rxjs/ajax';
import { map, catchError } from 'rxjs/operators';
// Return "response" from the API. If an error happens,// return an empty
array.
const apiData = ajax('/api/data').pipe(
        map(res => {
                if (!res.response) {
                        throw new Error('Value expected!');
                }
                return res.response;
        }),
        catchError(err => of([]))
);
apiData.subscribe({
        next(x) { console.log('data: ', x); },
        error(err) { console.log('errors already caught... will not run');
}
});
```

Add instructor notes here.

Demo

- Demo HttpGet
- Demo HttpDelete
- Demo HttpGetErrorHanding

Add the notes here.

Add instructor notes here.

Basics of Observables & Promises

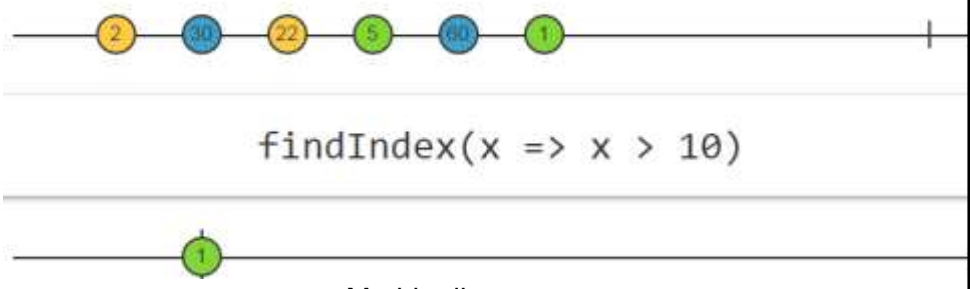
- Observables provide support for passing messages between publishers and subscribers in your application.
- Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values.
- Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they unsubscribe.

Data sequences can take many forms such as a stream of data from backend web service or a set of system notifications or a series of even such as user input.

Reactive extensions represent a data sequence as an observable sequen commonly just called an observable.

A method can be subscribed to an observable to receive asynchrono notifications as new data arrives. The method can then react with t arrived data. The method is notified when there is no more data or one error occurs. Since an observable works like an array we can use the m operator.

We can visualizing observable sequences with interactive diagrams fro



Marble diagram

Add instructor notes here.

Basics of Observables & Promises

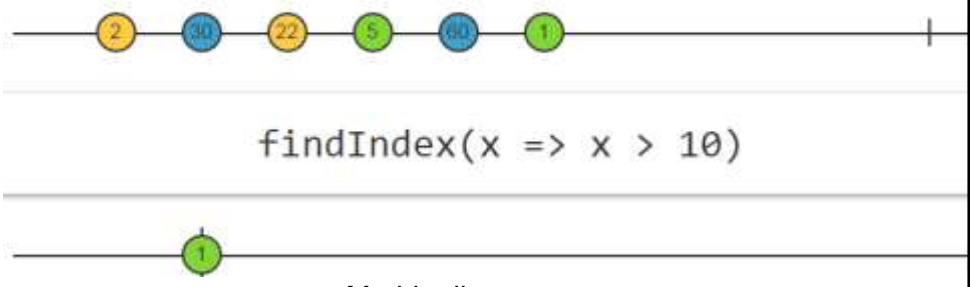
- An observable can deliver multiple values of any type—literals, messages, or events, depending on the context. The API for receiving values is the same whether the values are delivered synchronously or asynchronously. Because setup and teardown logic are both handled by the observable, your application code only needs to worry about subscribing to consume values, and when done, unsubscribing. Whether the stream was keystrokes, an HTTP response, or an interval timer, the interface for listening to values and stopping listening is the same.
- Because of these advantages, observables are used extensively within Angular, and are recommended for app development as well.

Data sequences can take many forms such as a stream of data from backend web service or a set of system notifications or a series of even such as user input.

Reactive extensions represent a data sequence as an observable sequen commonly just called an observable.

A method can be subscribed to an observable to receive asynchrono notifications as new data arrives. The method can then react with t arrived data. The method is notified when there is no more data or one error occurs. Since an observable works like an array we can use the m operator.

We can visualizing observable sequences with interactive diagrams fro



Marble diagram

Add instructor notes here.

Basics of Observables & Promises

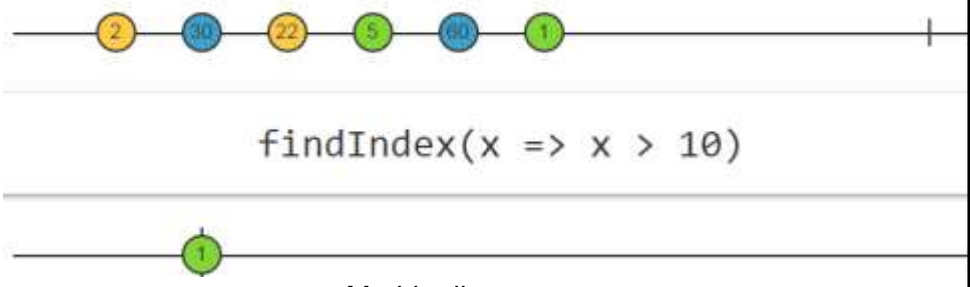
- Observables is a part of ReactiveX library also known as rxjs
  - import { Observable } from 'rxjs/Observable';
- Observables is like an array whose items arrived asynchronously. The role of ReactiveX to provide asynchronously programming
- Observable help to manage asynchronous data, such as data coming from a backend service. That data we are going to subscribe
- Observable work with multiple value
- Observable are cancellable
- Observable use javaScript function such as map filter & reduce

Data sequences can take many forms such as a stream of data from backend web service or a set of system notifications or a series of even such as user input.

Reactive extensions represent a data sequence as an observable sequen commonly just called an observable.

A method can be subscribed to an observable to receive asynchrono notifications as new data arrives. The method can then react with t arrived data. The method is notified when there is no more data or one error occurs. Since an observable works like an array we can use the m operator.

We can visualizing observable sequences with interactive diagrams fro



Marble diagram

Building & Using a Custom Observable

- Use the Observable constructor to create an observable stream of any type.
- The constructor takes as its argument the subscriber function to run when the observable's subscribe() method executes.
- A subscriber function receives an Observer object, and can publish values to the observer's next() method.

Building & Using a Custom Observable

- Create observable with constructor

```
// This function runs when subscribe() is called
function sequenceSubscriber(observer) {
// synchronously deliver 1, 2, and 3, then complete
observer.next(1);
observer.next(2);
observer.next(3);
observer.complete();
// unsubscribe function doesn't need to do anything in this
// because values are delivered synchronously
return {unsubscribe() {}};}
 // Create a new Observable that will deliver the above sequence
const sequence = new Observable(sequenceSubscriber);
 // execute the Observable and print the result of each notification
sequence.subscribe({
 next(num) { console.log(num); },
complete() { console.log('Finished sequence'); }});
// Logs:// 1// 2// 3// Finished sequence
```