**­Monique Conceição Rodrigues de Oliveira**

DESAFIO 5.3 – Serviços

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

1. O propósito do Service é atuar na organização, compartilhamento de código na aplicação.

1. Há uma distinção porque o objetivo do componente é definir a visualização através dos elementos e lógicas contidas nesse componente e o serviço fornece a funcionalidade específica ao componente não necessariamente relacionada às visualizações.
2. Primeiro se cria um Service através do Angular CLI

ex: ng g s Disciplinas --spec false --module App, aonde informa que o spec false é para não criar arquivo de testes e –module App é utilizado para que o módulo App seja modificado para prover o serviço.

Um modulo precisa prover ou fornecer um serviço para que ele possa ser utilizado pelos componentes. É realizada uma alteração na declaração do módulo App:

Interface gráfica do usuário, Texto, Email

Descrição gerada automaticamente

O atributo providers está informando para os demais componentes do móduloApp que o Serviço DisciplinasService está disponível para utilização.

VI- Uma Service pode fazer integração com o backEnd, efetuar validações.

**V- Verdadeiro- O serviço precisa existir em pelo um módulo para que a parte lógica seja criada.**

VI- Verdadeiro, pois é criado uma única service na inicialização da aplicação.

VII- *Pasta users*

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**b-**

1. Injeção de dependência é um padrão de projeto usado para evitar o alto nível de acoplamento de código dentro de uma aplicação.
2. Dependendo da necessidade do projeto a injeção de dependência pode ser apenas com services ou utilização de outras classes.
3. ?.

Texto

Descrição gerada automaticamente

**c-**

1. HTTP é um protocolo de transferência que possibilita que as pessoas que inserem a URL do seu site na Web possam ver os conteúdos e dados que nele existem.
2. NNTP: É um Protocolo de transferência de notícia de rede; SFTP: Usado para transferir arquivos com segurança com um arquivo remoto; TFTP: Transfere ficheiros, muito simples, semelhantes ao FTP; SMTP: permite que e-mails seja enviados de um servidor para outro servidor até que eles sejam entregues na sua caixa de mensagens(espécie de carteiro virtual); FTP: Forma fácil e versátil de transferência de arquivos, serve basicamente para que os usuários possam enviar ou receber documentos da Grande Rede por meio de um endereço no navegador ou um software instalado no PC.
3. O protocolo HTTP define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para em um recurso específico.
4. Primeiro devemos importar o módulo httpClientModule no módulo em que se encontra declarado o componente/ serviço em que as requisições serão realizadas; importar o httpClientModule para que a classe httpClient possa ser usada no componente AppComponent;no app.component.ts declarar no construtor da classe um parâmetro do tipo HttpClient.

V- Verdadeiro. Pode ser utilizado pois pode ser feita uma requisição(request) com o httpClient e utilizar o RxJs para não esperar uma resposta(response).

1. GET: Retorna os dados de uma requisição; POST: Insere um novo dado, PUT: atualiza os dados, DELETE: deleta os dados.
2. Verdadeiro. O Protocolo HTTP possui uma lista de códigos que informam através da resposta se obteve erro ou sucesso na solicitação efetuada onde conseguimos obter essa informação setando no método.

String Vazia- string;

arraybuffer- Objeto usado para representar um buffer de dados binários genéricos de comprimento fixo;

blob- representa um objeto do tipo arquivo, com dados imutáveis;

document- Interface que serve como ponto de entrada para o conteúdo da Página(a árvore DOM. incluindo elementos como <body> e <table> e provê funcionalidades globais ao documento (como obter a UR da páginae criar novos elementos no documento);

json- Objeto JS analisado a partir de uma sequência de caracteres JSON retornado pelo servidor;

text- String;

moz-blob- Usado pelo Furefox, recupera os dados parciais do tipo Blob, de eventos de progresso.

moz-chunked-text- Semelhante ao “text”,mas o streaming ainda está fluindo, o que significa que o valor na response só está disponível durante a expedição do progress do evento e contém apenas os dados recebidos desde a última progress do evento.

moz-chunked-arraybuffer- Quando response é acessado durante um progress evento que contém uma sequência com os dados.

200-Ok- requisição bem sucedida;

301-movido permanentemente- URI do recurso requerido mudou;

302- Movido temporariamente/ Descoberto (Found)- URI do recurso requerido mudou temporariamente;

400- Unautrorized- O cliente deve se autenticar para obter a resposta solicitada;

403- Forbidden- Cliente sem permissão de acesso ao conteúdo portanto o servidor está rejeitando dar a resposta;

404- Not Found- Servidor não pode encontrar o recurso solicitado;

405- Method Not Allowed- Método de solicitação é reconhecido pelo servidor, mas foi desativado e não pode ser usado;

500- Internal Server Error- Servidor encontrou uma situação ao qual não sabe lidar, e portanto, lança um erro interno, impedindo o fluxo normal da aplicação;

502- Service Inavailable- Servidor não está pronto para manipular a requisição.

504- Bad Gateway Timeout- Erro atuando com gateway (Ponto de ligação), e não obtém um uma resposta à tempo.

1. *Pasta exemplo chamadas*
2. Esse cabeçalho permite que o cliente e o servidor passem informações adicionais com a solicitação ou a resposta HTTP.
3. Interceptores são uma maneira de fazer algum trabalho para cada solicitação ou resposta HTTP; ele permite você interceptar solicitações HTTP de entrada e saída. E depois de acordo com as necessidades de sua aplicação.
4. Podemos usar para autenticar usuários, adaptar usuários e tratar erros do servidor.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**d-**

1. É um framework reativo para Javascript, que também existe para outras linguagens.
2. Promise sao feitas para um unico valor, rxjs pode ser aplicada para multiplos valores; Promise não é cancelável, o rxjs é;

Promise inicializam junto com a aplicacão, rxjs e lazy, por ser um framework reativo.

1. Significa que a execucao, por exemplo, de uma função, ocorrerá somente quando um determinado evento for disparado.

VI- É um padrão de projetos comportamental, criado pelo GoF onde tem-se um objeto que fica observando uma ação e quando ela acontece ele notifica os objetos que pediram para serem notificados quando a ação ocorrer.

V-É um padrão de projetos comportamental que tem como função percorrer conjuntos, abstraindo a lógica da iteração, ou fornecendo funções que auxiliam na tomada de decisão.

VI- É um paradigma de programação que baseia-se em na teoria matematica chamada lambda calculus que basicamente acontece por meio de funções, é imutável, não possui variáveis, ocorre de forma lazy(as funções são executadas somente, quando chamadas) e como eles são executados em uma pipeline, pode-se aproveitar melhor os recursos computacionais, por exemplo processamentos paralelos.

1. -Observable: representa a ideia do pattern observable, ou seja, notificar uma colecao de quem pediu para ser notificado;

- Observer : uma coleçã de métodos que sabem lidar com as respostas do Observable.

- Subcription: representa a assinatura de execução de um Observable. bem útil para cancelar execuções;

- Operators: funcoes que sao colocadas na pipeline de execucao para manipulação dos dados, por exemplo: map, reduce, filter .....

- Subject: equivalente a um EventEmitter, e é o único caminho de enviar um valor para múltiplos;

- Scheduler: são controladores de concorrência, permitindo coordenar quando uma computacão deve acontecer;

import { fromEvent } from 'rxjs';

fromEvent(document, 'click').subscribe(() => console.log('Clicked!'));

1. Os operadores de criação são:

- ajax

- bindCallback

- bindNodeCallback

- defer

- empty

- from

- fromEvent

- fromEventPattern

- generate

- interval

- of

- range

- throwError

- timer

- iif

-ajax: Cria um observável para uma solicitação Ajax com um objeto de solicitação com url, cabeçalhos, etc. ou uma string para uma URL.

Ex: Busca de objeto de resposta que está sendo retornado da API:

import { ajax } from 'rxjs/ajax';

import { map, catchError } from 'rxjs/operators';

import { of } from 'rxjs';

const obs$ = ajax(`https://api.github.com/users?per\_page=5`).pipe(

map(userResponse => console.log('users: ', userResponse)),

catchError(error => {

console.log('error: ', error);

return of(error);

})

);

-from: Tranforma um array, promessa ou iterável em um observável.

Ex: observável da string:

import { from } from 'rxjs';

const source = from('Hello World');

//output: 'H','e','l','l','o',' ','W','o','r','l','d'

const subscribe = source.subscribe(val => console.log(val));

-fromEvent: Cria um Observable que emite eventos de um tipo específico provenientes de determinado destino de evento.

Ex: Eventos de clique acontecendo no documento DOM:

import { fromEvent } from 'rxjs';

const clicks = fromEvent(document, 'click');

clicks.subscribe(x => console.log(x));

-generate: Gera uma sequência observável executando um loop controlado por estado produzindo os elementos da sequência, usando o agendador específico para enviar mensagens de observador.

Ex: Sequência de números:

import { generate } from 'rxjs';

const result = generate(0, x => x < 3, x => x + 1, x => x);

result.subscribe(x => console.log(x));

// Logs:

// 0

// 1

// 2

-interval: Cria um Observable que emite números sequênciais a cada intervalo de tempo específico, em um SchedulerLike.

Ex: Emitindo números ascendentes, um a cada segundo (1000ms) até o número 3:

import { interval } from 'rxjs';

import { take } from 'rxjs/operators';

const numbers = interval(1000);

const takeFourNumbers = numbers.pipe(take(4));

takeFourNumbers.subscribe(x => console.log('Next: ', x));

// Logs:

// Next: 0

// Next: 1

// Next: 2

// Next: 3

-throwError: Cria um observável que criará uma instância de erro e a enviará para o consumidor como um erro imediatamente após a assinatura.

Ex: Observável simples que criará um novo erro com um carimbo de data/hora e registre-o a mensagem toda vez que você se inscrever nele:

import { throwError } from 'rxjs';

let errorCount = 0;

const errorWithTimestamp$ = throwError(() => {

const error: any = new Error(`This is error number ${++errorCount}`);

error.timestamp = Date.now();

return error;

});

errorWithTimesptamp$.subscribe({

error: err => console.log(err.timestamp, err.message)

});

errorWithTimesptamp$.subscribe({

error: err => console.log(err.timestamp, err.message)

});

-timer: Após determinada duração, emite números em sequência a cada duração especificada.

Ex: Temporizador emite 1 valor e depois completa:

import { timer } from 'rxjs';

//emit 0 after 1 second then complete, since no second argument is supplied

const source = timer(1000);

//output: 0

const subscribe = source.subscribe(val => console.log(val));

X- Os operadores são:

-combineAll

- combineLatest

- concat

- concatAll

-endWith

- forkJoin

-merge

-mergeAll

-pairwise

-rase

-startWith

-withLatestFrom

-zip

XI- concat- Inscreve-se nos observáveis na ordem em que os anteriores são concluídos.

Ex: Uso básico de concat com três observáveis

import { of, concat } from 'rxjs';

concat(

of(1, 2, 3),

// subscribed after first completes

of(4, 5, 6),

// subscribed after second completes

of(7, 8, 9)

)

// log: 1, 2, 3, 4, 5, 6, 7, 8, 9

.subscribe(console.log);

-forkJoin- Emite o último valor de cada um quando todos os observáveis são concuídos.

Ex: Uso de um dicionário de fontes para fazer uma solicitação AJAX

// RxJS v6.5+

import { ajax } from 'rxjs/ajax';

import { forkJoin } from 'rxjs';

/\*

when all observables complete, provide the last

emitted value from each as dictionary

\*/

forkJoin(

// as of RxJS 6.5+ we can use a dictionary of sources

{

google: ajax.getJSON('https://api.github.com/users/google'),

microsoft: ajax.getJSON('https://api.github.com/users/microsoft'),

users: ajax.getJSON('https://api.github.com/users')

}

)

// { google: object, microsoft: object, users: array }

.subscribe(console.log);

-merge- Transforma vários observáveis em um único observável.

Ex: Mesclando 2 observáveis, método de instância;

import { merge } from 'rxjs/operators';

import { interval } from 'rxjs';

//emit every 2.5 seconds

const first = interval(2500);

//emit every 1 second

const second = interval(1000);

//used as instance method

const example = first.pipe(merge(second));

//output: 0,1,0,2....

const subscribe = example.subscribe(val => console.log(val));

XII-

Buffer

bufferCount

bufferTime

bufferToggle

bufferWhen

concatMap

concatMapTo

exhaustMap

expand

groupBy

map

mapTo

mergeMap/flapMap

mergeScan

partition

pluck

reduce

scan

switchMap

swichMapTo

toArray

window

windowCount

windowTime

windowToggle

windowWhen

XIII-

concatMap- Mapeia valores para observáveis internos, assina e emite em ordem.

Ex:

// RxJS v6+

import { of } from 'rxjs';

import { concatMap } from 'rxjs/operators';

//emit 'Hello' and 'Goodbye'

const source = of('Hello', 'Goodbye');

//example with promise

const examplePromise = val => new Promise(resolve => resolve(`${val} World!`));

//result of first param passed to second param selector function before being returned

const example = source.pipe(

concatMap(

val => examplePromise(val),

result => `${result} w/ selector!`

)

);

//output: 'Example w/ Selector: 'Hello w/ Selector', Example w/ Selector: 'Goodbye w/ Selector'

const subscribe = example.subscribe(val =>

console.log('Example w/ Selector:', val)

);

-map- Aplica projeção com cada valor da fonte:

Ex:

// RxJS v6+

import { from } from 'rxjs';

import { map } from 'rxjs/operators';

//emit (1,2,3,4,5)

const source = from([1, 2, 3, 4, 5]);

//add 10 to each value

const example = source.pipe(map(val => val + 10));

//output: 11,12,13,14,15

const subscribe = example.subscribe(val => console.log(val));

-mapTo- Mapeia as emissões para um valor constante.

Ex:

// RxJS v6+

import { interval } from 'rxjs';

import { mapTo } from 'rxjs/operators';

//emit value every two seconds

const source = interval(2000);

//map all emissions to one value

const example = source.pipe(mapTo('HELLO WORLD!'));

//output: 'HELLO WORLD!'...'HELLO WORLD!'...'HELLO WORLD!'...

const subscribe = example.subscribe(val => console.log(val));

-mergeMap- Mapeia para observável/ emite valores.

Ex:

// RxJS v6+

import { fromEvent, of } from 'rxjs';

import { mergeMap, delay } from 'rxjs/operators';

// faking network request for save

const saveLocation = location => {

return of(location).pipe(delay(500));

};

// streams

const click$ = fromEvent(document, 'click');

click$

.pipe(

mergeMap((e: MouseEvent) => {

return saveLocation({

x: e.clientX,

y: e.clientY,

timestamp: Date.now()

});

})

)

// Saved! {x: 98, y: 170, ...}

.subscribe(r => console.log('Saved!', r));

-mergeMapTo- Projeta cada valor de origem para o mesmo Observable que é mesclado várias vezes na saída Observable.

Ex: Para cada exemplo de clique, inicie um intervalo observável marcando a cada 1 segundo.

import { fromEvent, interval } from 'rxjs';

import { mergeMapTo } from 'rxjs/operators';

const clicks = fromEvent(document, 'click');

const result = clicks.pipe(mergeMapTo(interval(1000)));

result.subscribe(x => console.log(x));

-switchMap- Projeta cada valor de origem para um Observable que é mesclado no Observable de saída, emitindo valores apenas do Observable projetado mais recentemente.

import { fromEvent, interval } from 'rxjs';

import { switchMap } from 'rxjs/operators';

const clicks = fromEvent(document, 'click');

const result = clicks.pipe(switchMap((ev) => interval(1000)));

result.subscribe(x => console.log(x));

-switchMapTo- Projeta cada valor de origem para o mesmo Observable que é nivelado várias vezes com switchMap.

Ex: Intervalo observável em cada evento de clique:

import { fromEvent, interval } from 'rxjs';

import { switchMapTo } from 'rxjs/operators';

const clicks = fromEvent(document, 'click');

const result = clicks.pipe(switchMapTo(interval(1000)));

result.subscribe(x => console.log(x));