APLICAÇÕES INTERATIVAS

INTEGRAÇÃO COM WEB STORAGE

Utilizando Web Storage

- Web Storage são dados armazenados localmente no navegador de um usuário
- Existem dois tipos de armazenamento:
  - Local Storage - dados sem data de validade que permanecem após o fechamento da janela do navegador
  - Session Storage - dados que são limpos após o fechamento da janela do navegador

Utilizando Local Storage

Útil para salvar dados como preferências do usuário (tema de cores claras ou escuras em um site), lembrar itens do carrinho de compras ou lembrar que um usuário está conectado a um site

Utilizando Local Storage

- Anteriormente, os cookies eram a única opção para armazenar dados temporários locais
- O armazenamento local tem um limite de armazenamento significativamente mais alto (5 MB vs 4KB) e não é enviado a cada solicitação HTTP, portanto pode ser uma opção melhor

Métodos do Local Storage

Método	Descrição
setItem()	Adicionar uma chave e um valor ao local storage
getItem()	Obtém de volta um valor através de uma chave
removeItem()	Remove um item pela chave
clear()	Limpa todos os dados

Utilizando Local Storage

- Para salvar dados, utilizamos o método setItem, especificando uma chave e um valor

```
localStorage.setItem('chave', 'valor');
```

Utilizando Local Storage

- Para recuperar dados salvos, utilizamos o método getItem, especificando a chave do item salvo

```
const item = localStorage.getItem('chave');
```

Salvando o Array no Local Storage

- Para nós, é necessário salvar vários dados no local storage (um array), não apenas um dado. Como proceder?
- Podemos transformar o array em um JSON e salvá-lo como uma string

JSON

- Acrônimo para JavaScript Object Notation
- Formato compacto, de padrão aberto e independente, utilizado para troca de dados simples e rápida (parsing) entre sistemas

JSON

- Utiliza texto legível a humanos, no formato chave-valor
- Modelo de transmissão de informações no formato texto, muito usado em web services (REST) e aplicações AJAX, substituindo o uso de XML

JSON

- JSON vai criar uma representação de texto de nosso array, como a seguir:

JSON


```
[
  {
      "name":"Zelda",
      "status":"t",
      "genre":"Adventure",
      "platform":"Switch",
      "id":0
  },
  {
      "name":"Resident Evil 2",
      "status":"q",
      "genre":"Horror",
      "platform":"PC",
      "id":1
  }
]
```

JSON

- Para transformar um array em JSON, podemos utilizar o método stringfy da classe JSON (do navegador)

```
JSON.stringify(array)
```

JSON

- Para transformar uma string JSON de volta em Array, podemos utilizar o método parse da classe JSON

```
JSON.parse(string)
```

Local Storage no Game App

- Para fazer com que o game app seja capaz de salvar e carregar o array do local storage, vamos adicionar dois métodos: o save e o load
- O save irá salvar o array no local storage, convertendo o em JSON antes
- E o load fará o carregamento de volta do LS

Local Storage no Game App

- O método de save será chamado sempre que uma alteração for feita, em todos os demais métodos de alteração
- O load será chamado na listagem, visando atualizar o array do serviço com os dados salvos

Local Storage no Game App

- Atualize o serviço com os métodos novos, como a seguir:

```
...
  save(): void {
    localStorage.setItem('games', JSON.stringify(this.games));
    localStorage.setItem('gameAutoGeneratedId', this.autoGeneratedId.toString());
  }

  load(): void {
    const storageValues = localStorage.getItem('games');
    if (storageValues) {
      this.games = JSON.parse(storageValues)
    }
    else {
      this.games = new Array<Game>();
    }
    const autoGenId = localStorage.getItem('gameAutoGeneratedId');
    if (autoGenId) {
      this.autoGeneratedId = Number(autoGenId);
    }
  }
...
```

Local Storage no Game App

- E atualize os métodos de salvamento/listagem, como a seguir:

Local Storage no Game App

```
...

insert(game: Game) {
    game.id = this.autoGeneratedId;
    this.games.push(game);
    this.autoGeneratedId++;
    this.save();
  }

  list(): Array<Game> {
    this.load();
    return this.games;
  }

...
```

Local Storage no Game App

```
...
remove(id: number) {
    for (let i = 0; i < this.games.length; i++) {
      let m = this.games[i];

      if (m.id === id) {
        this.games.splice(i, 1);
        break;
      }
    }
    this.save();
  }

  update(game: Game) {
    for (let i = 0; i < this.games.length; i++) {
      let g = this.games[i];
      if (g.id === game.id) {
        this.games[i] = game;
        break;
      }
    }
    this.save();
  }
...
```

ROTAS

Rotas do Angular

- O Angular trabalha com a composição de aplicações em página única
- Nesse cenário, os usuários permanecem em uma única página, mas a visualização dessa página muda dependendo do que o usuário faz

Rotas do Angular

- Para lidar com a navegação de uma view para outra, utilizamos o router (roteador) do Angular
- O router permite a navegação interpretando um URL do navegador como uma instrução para alterar a exibição de um componente

Rotas do Angular

- Quando um link for acionado e apontar para determinado endereço (por exemplo, opções de menu forem acionadas) determinado componente poderá ser carregado e exibido na página

Rotas do Angular

- O sistema de rotas precisa de alguns elementos para funcionar corretamente:
  - Um array contendo todas as definições de rotas (no app-routing.module.ts)
  - Um local onde os componentes acionados serão carregados (dentro de app-component), também chamado de outlet
  - Links que acionem as rotas (menu) no app-component

Array de rotas

- O array de rotas possui o seguinte formato:

```
const routes: Routes = [
        { path: 'test1-component', component: Test1Component },
        { path: 'test2-component', component: Test2Component },
];
```

Array de rotas

- Onde **path** indica qual o caminho de URL deve ser digitado na barra de endereços (ou ser fornecido como href de um link) para carregar o componente e **component** corresponde ao componente que deve ser carregado

Array de rotas

- O array deve ser declarado dentro do elemento *app-routing.module.ts* e não precisa conter todos os componentes da aplicação, apenas os componentes que deseja-se permitir acesso por links ou pela URL

Array de rotas no app-routing.module

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { Test1Component } from './test1/test1.component';
import { Test2Component } from './test2/test2.component';


const routes: Routes = [
  { path: 'test1-component', component: Test1Component },
  { path: 'test2-component', component: Test2Component }
];


@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Router Outlet

- O router outlet corresponde ao local onde os componentes serão carregados quando a URL respectiva for acionada
- Utilizado através da tag **router-outlet**

Router Outlet

- Normalmente são definidos **dentro do app-component**, no local onde os elementos principais da aplicação deverão ser carregados (separado de itens de menu, layout e etc), como o *container* principal onde o conteúdo da aplicação ficará

Router Outlet

```
<router-outlet></router-outlet>
```

Links para componentes

- Os links vão disparar o acionamento do router para solicitar o carregamento de determinado componente
- Normalmente definidos no mesmo local onde encontra-se o router-outlet

Links para componentes

- Utilizamos links normais para links de rotas, com alguns atributos adicionais:

```
<a routerLink="/test1-component">Nome</a>
```

Links para componentes

- Onde routerLink corresponde a rota de componente configurada no app-routing.module

EXERCÍCIOS

Exercício

- Implemente um CRUD de sua preferência (livros, cartas de magic, lista de compras, o que preferir)
- O item de modelo desse CRUD deve ter ao menos 5 atributos e não pode ser de um item que já fizemos até aqui (filmes, jogos)

Exercício

- O CRUD deve permitir gerenciar as informações e nome do item de negócio (inserir, excluir, alterar, listar)
- Melhore o CSS da aplicação e deixe-a mais apresentável

Exercício

- DESAFIO 1! Crie um segundo CRUD, que permita gerenciar um atributo do primeiro (por exemplo, tipo, categoria, fabricante, etc)

Exercício

- DESAFIO 2! Implemente também a funcionalidade de busca de itens por um dos atributos do modelo

REST

O que é REST?

- REST ou Representational State Transfer (Transferência de Estado Representacional) é uma arquitetura utilizada para criação de serviços da web

O que é REST?

- Os serviços web no padrão REST são chamados serviços RESTful e fornecem interoperabilidade entre sistemas na internet

O que é REST?

- Serviços RESTful permitem que sistemas acessem e manipulem recursos pela internet usando um conjunto predefinido de operações sem estado, através de representações textuais

O que é REST?

- Serviços RESTful permitem disponibilizar (na internet) um conjunto de operações que apenas usam dados enquanto estão sendo executadas (sem sessão, sem estado ou *stateless*) através do envio e recebimento de texto (normalmente XML ou JSON)

O que é REST?

- Apesar de serviços RESTful não armazenarem variáveis entre execuções, eles ainda podem manipular banco de dados e realizar persistência de informações

O que é REST?

- **Resumindo, REST permite chamar funções em outros sistemas pela internet =)**
- As funções podem estar em qualquer linguagem (mesmo dois sistemas com linguagens diferentes), visto que a entrada e a saída sempre é JSON

Como REST funciona?

- Um serviço REST está, normalmente associado a uma URL e a um método HTTP

Como REST funciona?

- Uma URL é um identificador único para acesso a um recurso na web. Normalmente utilizamos para acessar uma página
  - http://viacep.com.br/ws/04696000/json/
- Método HTTP indica o que fazer com o recurso
  - Métodos HTTP : GET, PUT, POST, DELETE e etc

Acessando recursos na web

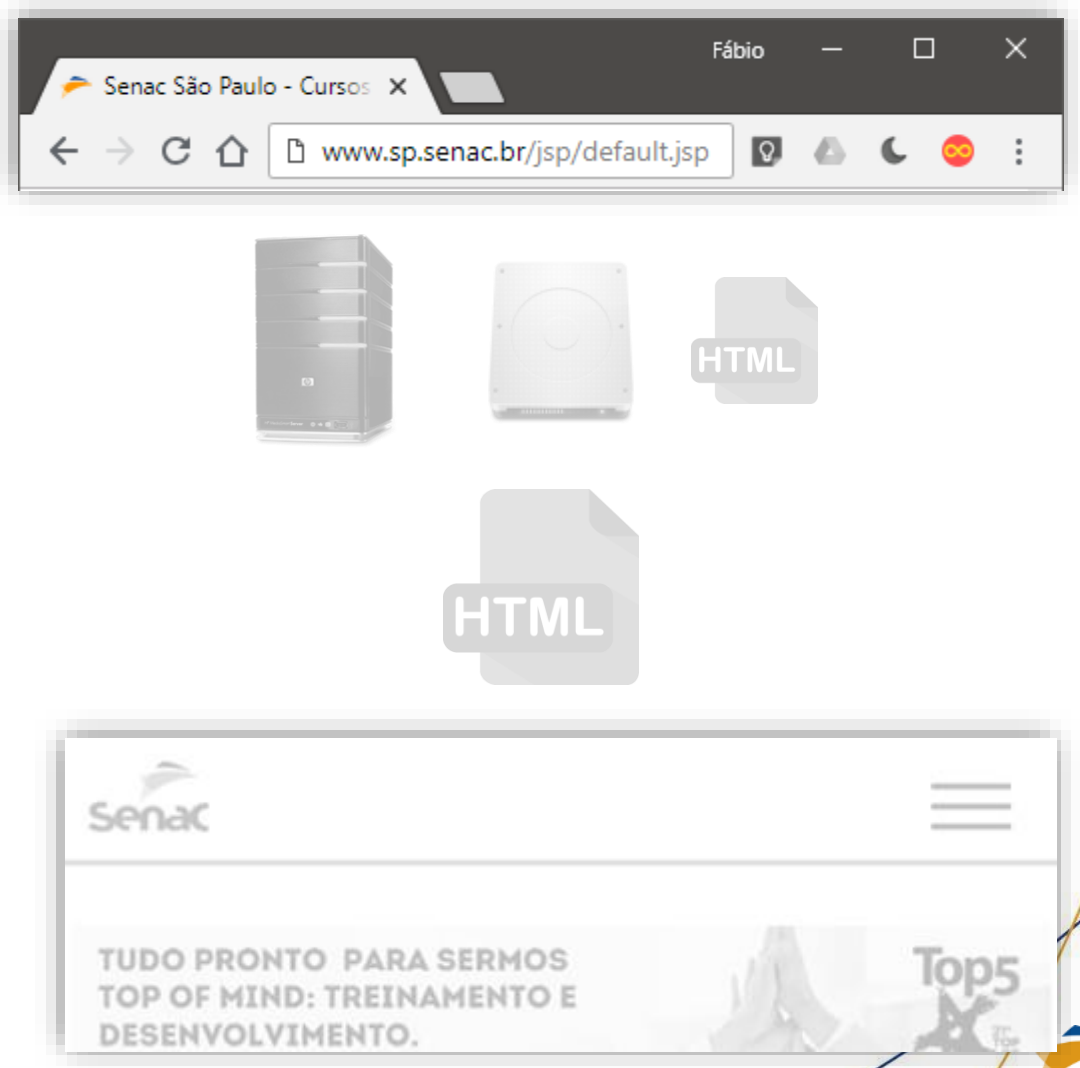
- Por padrão, quando digitamos um endereço no navegador, o método GET é utilizado
- O método GET indica que determinado recurso deve ser obtido naquele endereço e devolvido ao cliente

Acessando recursos na web

- Nesta situação (acesso a um site pelo navegador) este recurso normalmente é a página que se deseja acessar (ou um arquivo de estilos CSS, script, música, imagem, vídeo ou etc)

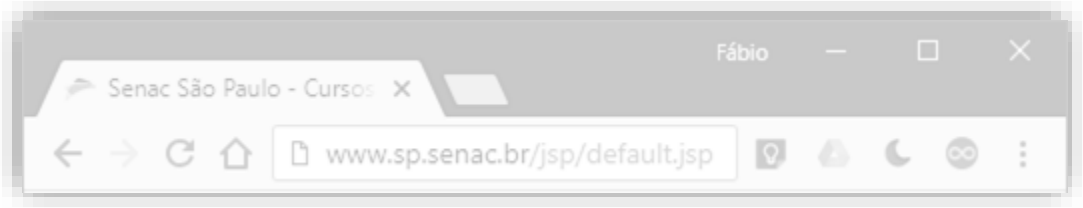
Acessando recursos na web

1 – O endereço da URL desejada é digitado no navegador



Acessando recursos na web

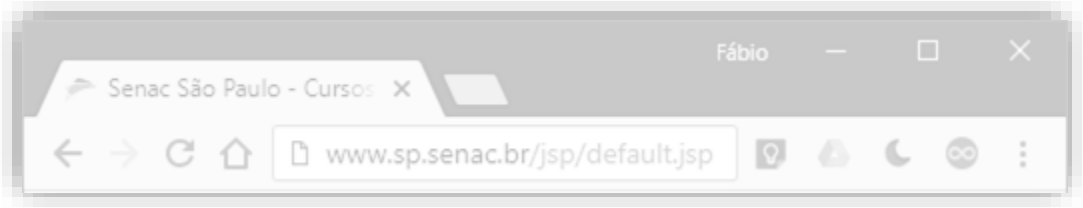
1 – O endereço da URL desejada é digitado no navegador



2 – O navegador localiza o recurso desejado (página)

Acessando recursos na web

1 – O endereço da URL desejada é digitado no navegador

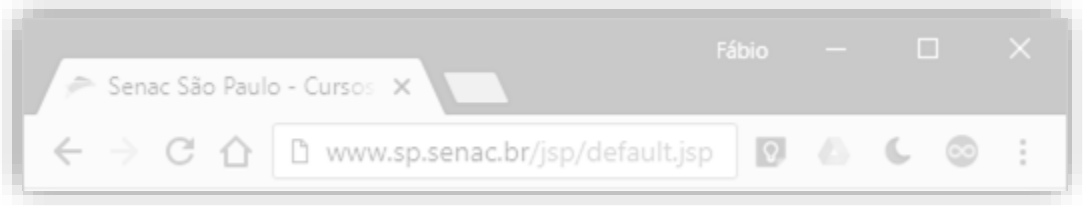


2 – O navegador localiza o recurso desejado (página)

3 – O recurso (página) é enviado pelo servidor para o navegador

Acessando recursos na web

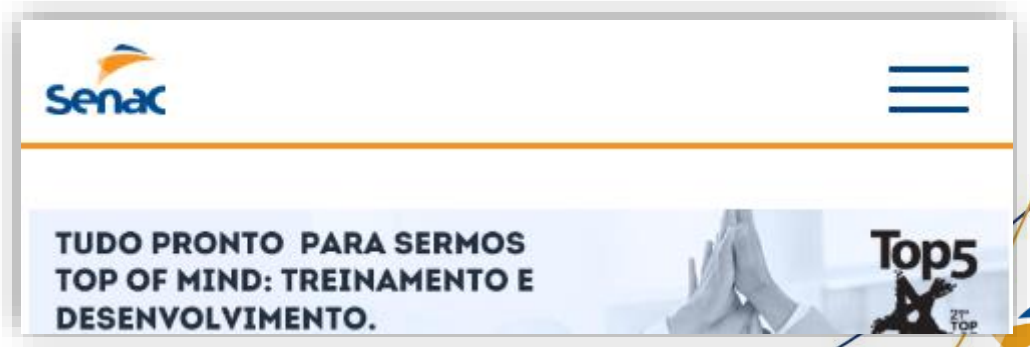
1 – O endereço da URL desejada é digitado no navegador



2 – O navegador localiza o recurso desejado (página)

3 – O recurso (página) é enviado pelo servidor para o navegador

4 – O navegador faz o download do recurso (página) e a exibe



Acessando recursos REST

- Numa visão superficial, as chamadas REST se assemelham ao padrão de requisições de páginas/recursos web pelo navegador

Acessando recursos REST

- Através de uma chamada REST, informando a URL desejada, os parâmetros (caso existam) e o método HTTP a utilizar, é possível executar uma função do servidor de forma remota e obter o resultado da execução

Acessando recursos REST

- Uma função RESTful pode, por exemplo, obter **dados de uma base de dados**, **efetuar cálculos ou processamento** ou realizar **quaisquer outras operações computacionais** (conforme a necessidade) e devolver estes dados para o elemento (cliente) que a chamou

Acessando recursos REST

1 – O serviço RESTful é chamado por um elemento cliente (normalmente o JavaScript de um navegador, mas pode ser qualquer software)

Acessando recursos REST

1 – O serviço RESTful é chamado por um elemento cliente (normalmente um JavaScript num navegador, mas pode ser qualquer software)

2 – O servidor faz todo o processamento e localização de recursos necessária e manda os dados resultantes de volta para o cliente

Acessando recursos REST

1 – O serviço RESTful é chamado por um elemento cliente (normalmente um JavaScript num navegador, mas pode ser qualquer software)

2 – O servidor faz todo o processamento e localização de recursos necessária e manda os dados resultantes de volta para o cliente

3 – O cliente obtém os dados e os utiliza para exibição ou processamento

Formato de dados de texto

- Os serviços REST podem trafegar textos em alguns formatos diferentes. Os principais são XML e JSON (o mais comum)

Formato de dados de texto

- XML é um formato que lembra o HTML, porém, apresenta recursos de tags bastante estrito (menos permissivo) quando comparado ao HTML (mais permissivo)

Formato de dados de texto

- Já o JSON é um formato de armazenamento de dados mais simples, agrupado, basicamente, por elementos de chave (identificadores) e valor

XML vs JSON

```xml
<funcionarios>
  <funcionario>
    <nome>João</nome>
    <sobrenome>da Silva</sobrenome>
  </funcionario>
  <funcionario>
    <nome>Ana</nome>
    <sobrenome>Santos</sobrenome>
  </funcionario>
  <funcionario>
    <nome>Pedro</nome>
    <sobrenome>Teixeira</sobrenome>
  </funcionario>
</funcionarios>
```

```json
{
"funcionarios":[
{ "nome":"João", "sobrenome":"da Silva" },
{ "nome":"Ana", "sobrenome":"Santos" },
{ "nome":"Pedro", "sobrenome":"Teixeira" }]
}
```

Resumo sobre REST

- Serviços REST ou serviços RESTful são uma forma de disponibilizar a execução de funções sem estado pela internet
- REST é construído numa arquitetura onde o cliente realiza uma chamada e o servidor envia uma resposta por texto

Resumo sobre REST

- O acionamento de um serviço REST é feito através de uma URL, parâmetros da URL e o método HTTP a executar
- O serviço REST trabalha com envio e recebimento de texto, normalmente nos formatos JSON ou XML

Resumo sobre REST

- Múltiplas linguagens e plataformas podem trabalhar com REST ao mesmo tempo e permitir a consulta ou criação de serviços REST

CONSUMINDO SERVIÇOS REST EM ANGULAR

Acesso a Dados com Serviços

- Como vimos, o acesso a dados normalmente é feito pelos serviços em Angular
- Dessa forma, o consumo de serviços REST também deve ser feito em serviços

Acesso a Dados com Serviços

- As aplicações SPA requerem que a página esteja responsiva a maior parte do tempo
- Tarefas de persistência e consumo de dados podem ser demoradas, o que travaria o uso da aplicação durante estas operações

Acesso a Dados com Serviços

- Todas as fontes de dados que usamos até agora (memória, local storage) tem persistência praticamente instantânea, de forma que não foi necessário se preocupar com isso por enquanto

Acesso a Dados com Serviços

- Os serviços REST trafegam dados pela internet, de forma que a resposta pode demorar, dependendo de onde o serviço está, a qualidade da conexão e o tamanhos dos dados trafegados

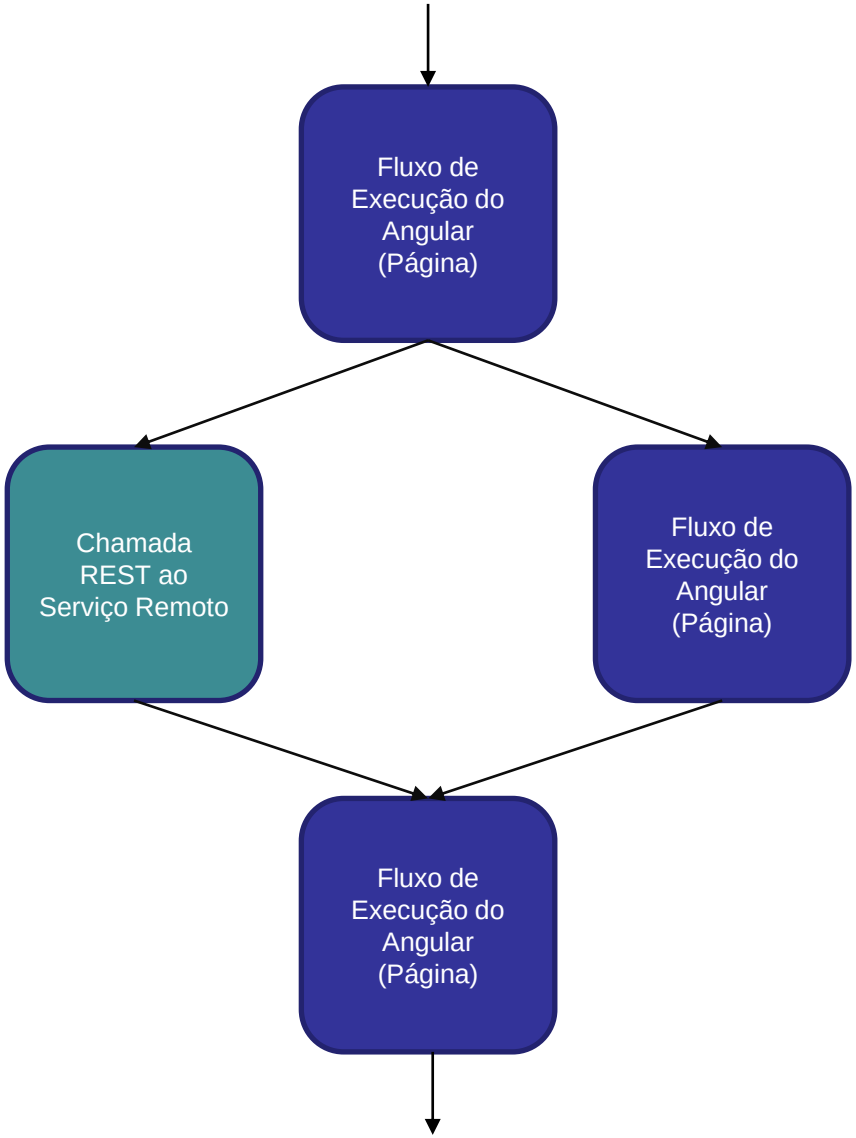
Acesso a Dados com Serviços

- Para evitar travamentos das páginas, as operações de consumo de serviços REST devem ser executadas de forma assíncrona
- Ou seja, as ações do usuário não devem travar a página e deveriam executar em paralelo ao consumo dos serviços REST

Acesso a Dados com Serviços

- Para isso, deve ser possível executar funções sem que seja necessário esperar um retorno (caso contrário a página ficaria congelada enquanto o back end não responder)
- Ou seja, funções de acesso a dados devem ser executadas **em paralelo** ao código da interface

Acesso a Dados com Serviços



Configurando métodos do serviço

- Para tanto, Angular propõe o padrão **Observable**
- No padrão Observable, iniciamos um método sem que seja necessário esperar sua resposta**, o que evita travamento da página**

Padrão Observable

- **Quando o método terminar sua execução, seremos notificados (um callback nosso executará) e poderemos realizar alguma ação nos dados retornados da execução**

Padrão Observable

- É como se o método permitisse a **definição de uma função a ser chamada** quando terminar de executar (callback)

Padrão Observable

- Para utilização do padrão observable, basta que o método que poderá ser chamado de forma assíncrona retorne um elemento do tipo **Observable**

Padrão Observable

- O elemento do tipo Observable pode especificar qual tipo de dado será retornado quando o callback for acionado
- É necessário realizar a importação de Observable para usá-lo

Chamadas HTTP

- As chamadas HTTP em si são feitas no serviço através de uma classe denominada HttpClient, que deve ser injetada no serviço
- O HttpClient possui funções específicas para cada tipo de método HTTP e todas elas lidam também com elementos observable

Chamadas HTTP

- O objetivo é construir funções nos serviços que utilizem o HttpClient para chamadas REST e devolvam os elementos observable das chamadas HTTP para quem os chamar

Chamadas HTTP

- Os componente visuais, então, devem fazer as chamadas e lidar com o retorno (o observable) dos serviços
- Para isso, eles se “inscrevem” nos observables, definindo o que deverá ser feito quando a chamada REST tiver uma resposta

Chamadas HTTP

- “Se inscrever” num observable significa definir uma função (de callback) a executar quando o observable concluir sua execução
- A função normalmente atualizará a interface de alguma forma com dados obtidos do serviço REST

Estrutura Básica de Chamadas

Serviço

```
...
constructor(private http: HttpClient) { }

public funcaoServico(paramEntrad: tipo): Observable<TipoSaida> {
  return this.http.get<TipoSaida>('endereço/' + params_get);
}
...
```

Componente

```
...
funcaoDoComponente() {
  this.service.funcaoServico(paramEntrada).subscribe(objSaida => {
    //Fazer Algo com o retorno do servidor (objSaida)
  });
}
...
```


"That's all Folks!"