



# WEB ACADEMY

## Frameworks Front-end

Daniel Augusto Nunes da Silva

# **Apresentação**

# Ementa

- **Frameworks Front-end. TypeScript.** Introdução ao **Angular**. Interface de linha de comando (**CLI**) do Angular. Espaços de trabalho e estrutura de projetos no Angular. Execução e *deploy*. **Componentes**, serviços e rotas. **Comunicação com aplicações *back-end*** por meio do protocolo HTTP.



# Objetivos

- **Geral:** Conhecer os principais **procedimentos e técnicas** de desenvolvimento de aplicações para a WEB utilizando **frameworks front-end**, com **ênfase no Angular** e seus principais recursos.
- **Específicos:**
  - Discutir as principais funcionalidades de um framework front-end;
  - Apresentar o TypeScript e sua relação com o JavaScript, destacando semelhanças e diferenças;
  - Conhecer os fundamentos e principais recursos do framework Angular.
  - Capacitar o aluno para utilização do framework Angular na construção de uma aplicação front-end que se comunica com um serviço back-end.

# Conteúdo programático

## Introdução

- Frameworks front-end;
- Recursos básicos de frameworks front-end;
- Roteamento no lado cliente;
- Introdução ao TypeScript;
- Definição de tipos em TypeScript.

## Fundamentos

- O que é o Angular?;
- Angular CLI;
- Espaços de trabalho;
- Estrutura do projeto, principais arquivos e diretórios.
- Execução e deploy da aplicação.

## Componentes

- Visão geral dos componentes no Angular;
- Templates;
- Diretivas;
- Rotas.

## Comunicação com o back-end

- Serviços e injeção de dependência;
- O serviço HTTP Client;
- Interceptadores;
- Observables (RxJS).

# Bibliografia



## JavaScript e JQuery: desenvolvimento de interfaces web interativas.

Jon Duckett  
1ª Edição – 2016  
Editora Alta Books  
ISBN 9781118871652



## The TypeScript Handbook

Microsoft  
<https://www.typescriptlang.org/docs/handbook/intro.html>



## Engenharia de Software Moderna

Marco Tulio Valente  
<https://engsoftmoderna.info/>

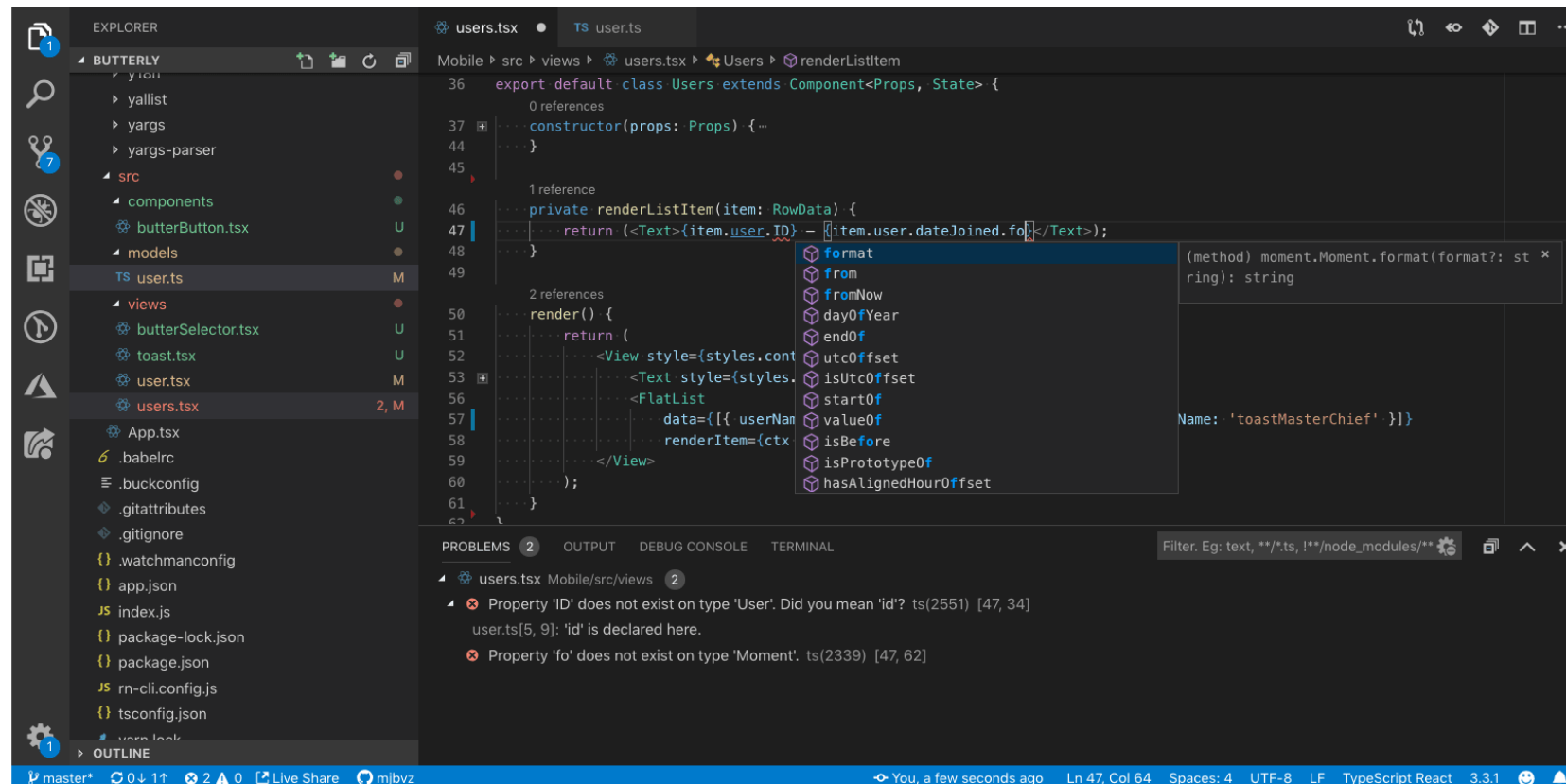


# Sites de referência

- MDN Web Docs: Aprendendo desenvolvimento web.
  - <https://developer.mozilla.org/pt-BR/docs/Learn>
- TypeScript Documentation.
  - <https://www.typescriptlang.org/docs/>
- Angular Docs.
  - <https://angular.io/docs>

# Ferramentas

Visual Studio Code



<https://code.visualstudio.com/docs/languages/typescript>

<https://code.visualstudio.com/docs/nodejs/angular-tutorial>



# Ferramentas: Extensões do VS Code

- **Angular Language Service**
  - <https://marketplace.visualstudio.com/items?itemName=Angular.ng-template>

# Ferramentas

- Node.js (e npm)
  - <https://nodejs.org/en/download/>
- Angular CLI:
  - <https://angular.io/guide/setup-local>
    1. Instalar Node.js
    2. `npm install -g @angular/cli`

# Contato



<https://linkme.bio/danielnsilva/>

# Introdução

# Frameworks front-end (lado cliente)

- **Frameworks** fornecem ferramentas que **simplificam as operações comuns de desenvolvimento**.
- Em geral, **frameworks front-end são sinônimo de frameworks JavaScript**, utilizados para construção aplicações com interface de usuário.
- Construir páginas com JavaScript consiste sobretudo em manipular o DOM (utilizar métodos como *createElement*, *appendChild*, etc.).
- Dificuldade: **toda vez que alteramos algum dado da aplicação, é necessário atualizar a interface do usuário**, o que implica em muitas linhas de código para manipular o DOM.



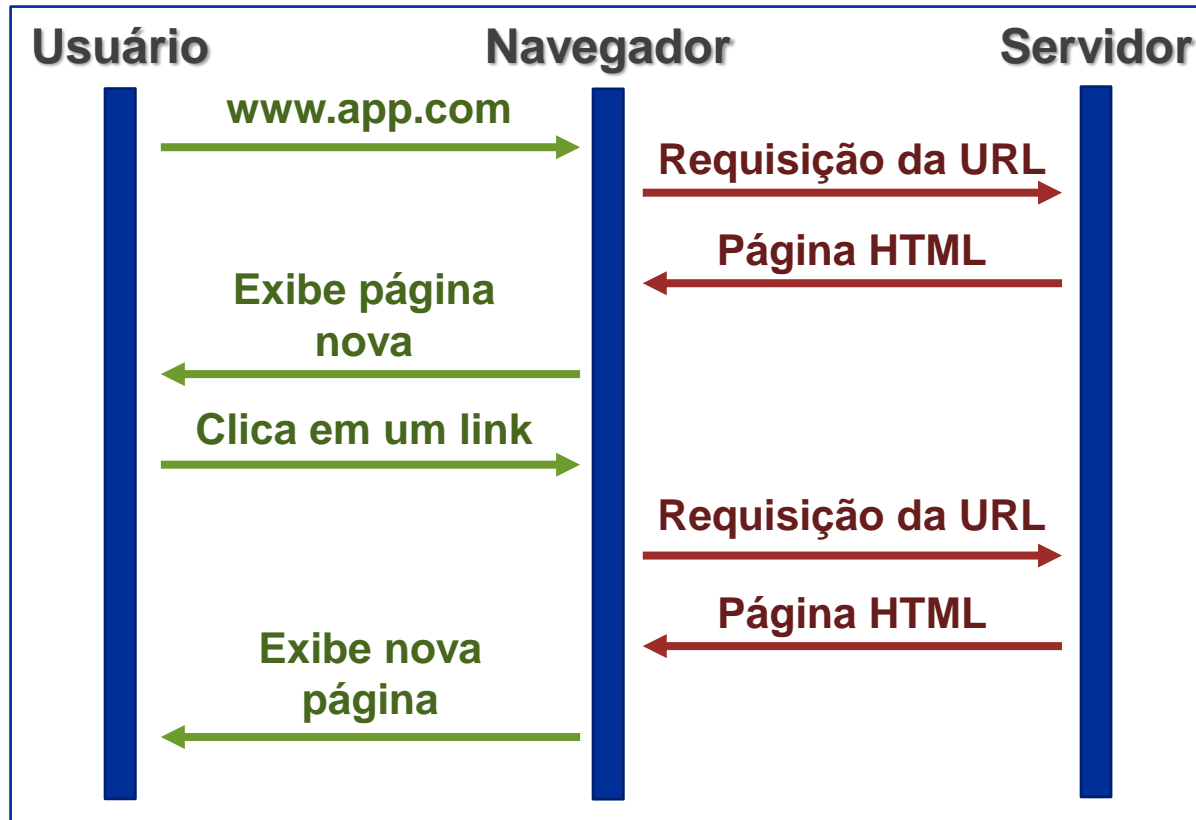
# O que um framework front-end fornece?

- Um **modo declarativo de construção de interfaces de usuário**, permitindo que uma grande volume de código para manipular o DOM seja resumido em algumas linhas de código que descrevem como a interface deve funcionar.
- **Conjunto de ferramentas** que simplificam várias tarefas do processo de desenvolvimento.
- **Organização da aplicação em componentes** e/ou módulos reutilizáveis.
- **Roteamento no lado cliente**, onde a atualização do conteúdo da página, por meio de requisições assíncronas e manipulação do DOM, geram novas “pseudo-páginas” que se comportam como URLs diferentes.

# Roteamento no lado cliente e no servidor

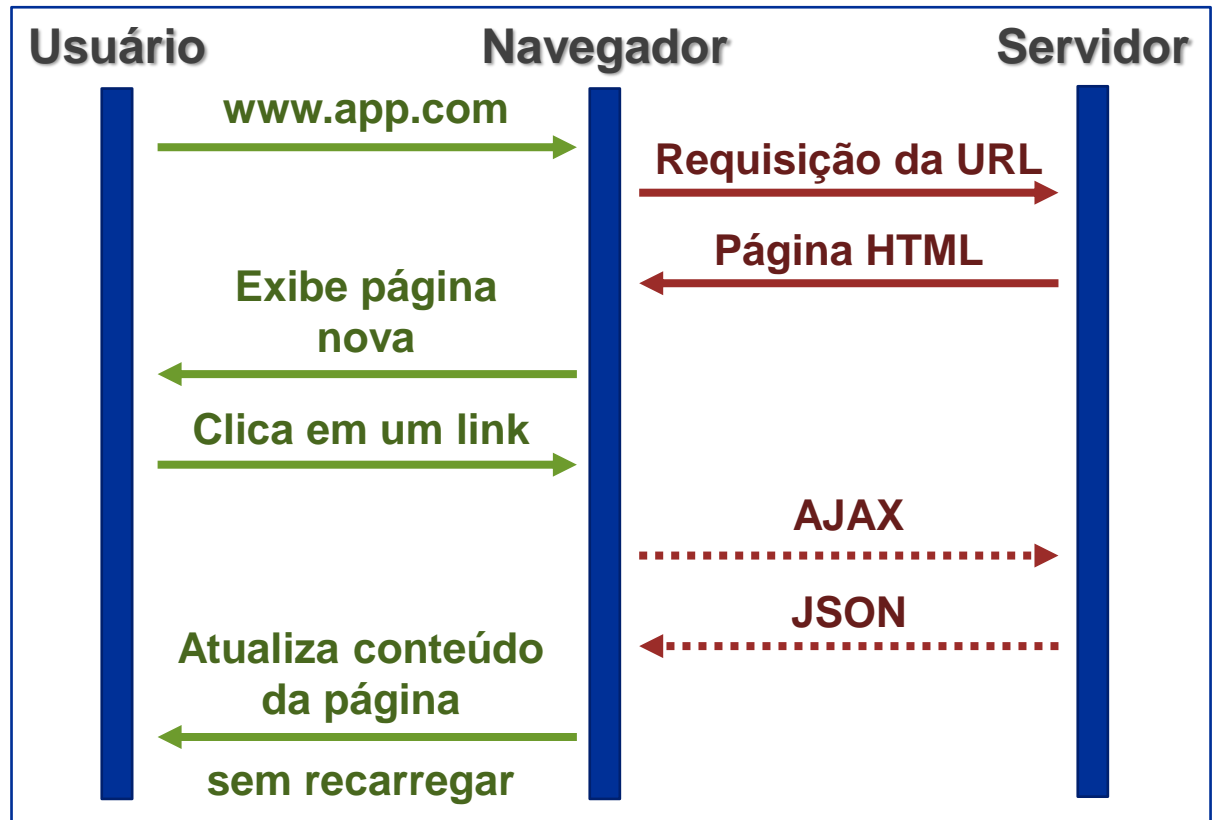
## Roteamento no servidor

Páginas HTML



## Roteamento no cliente

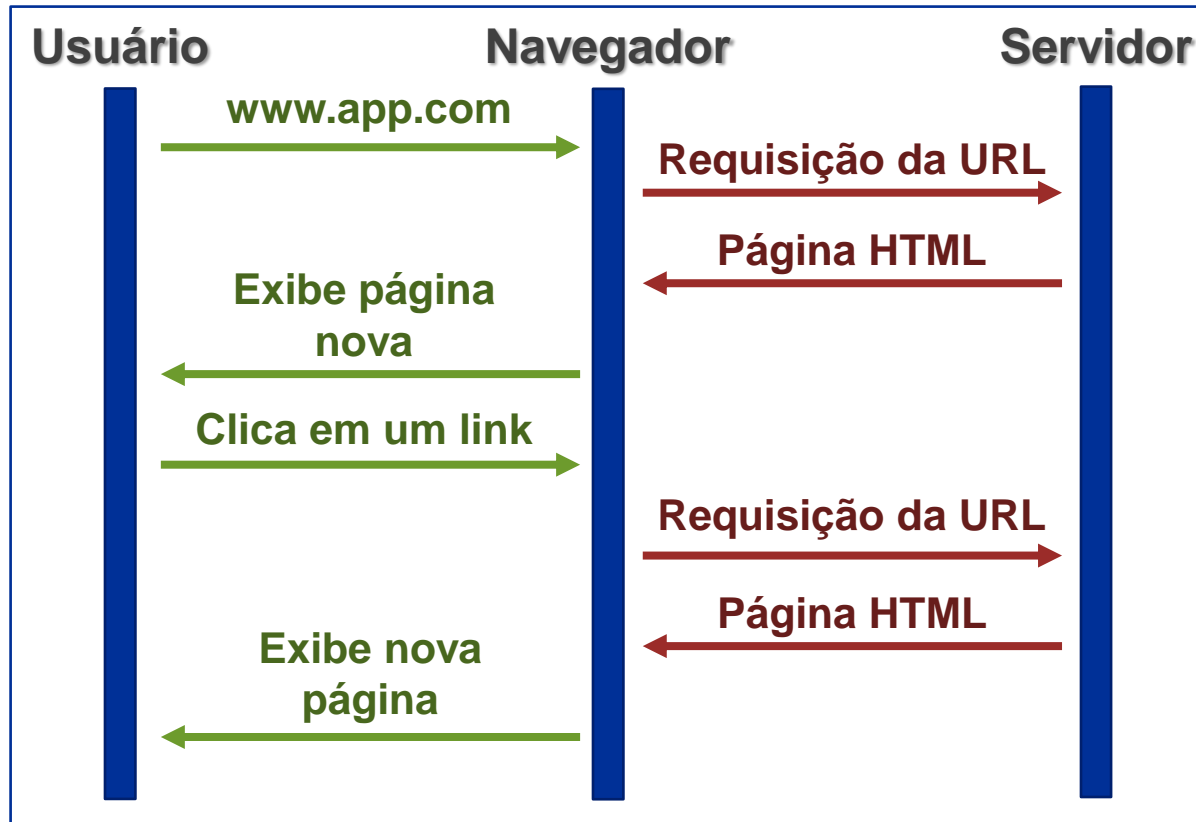
Single Page Application (SPA)



# Roteamento no lado cliente e no servidor

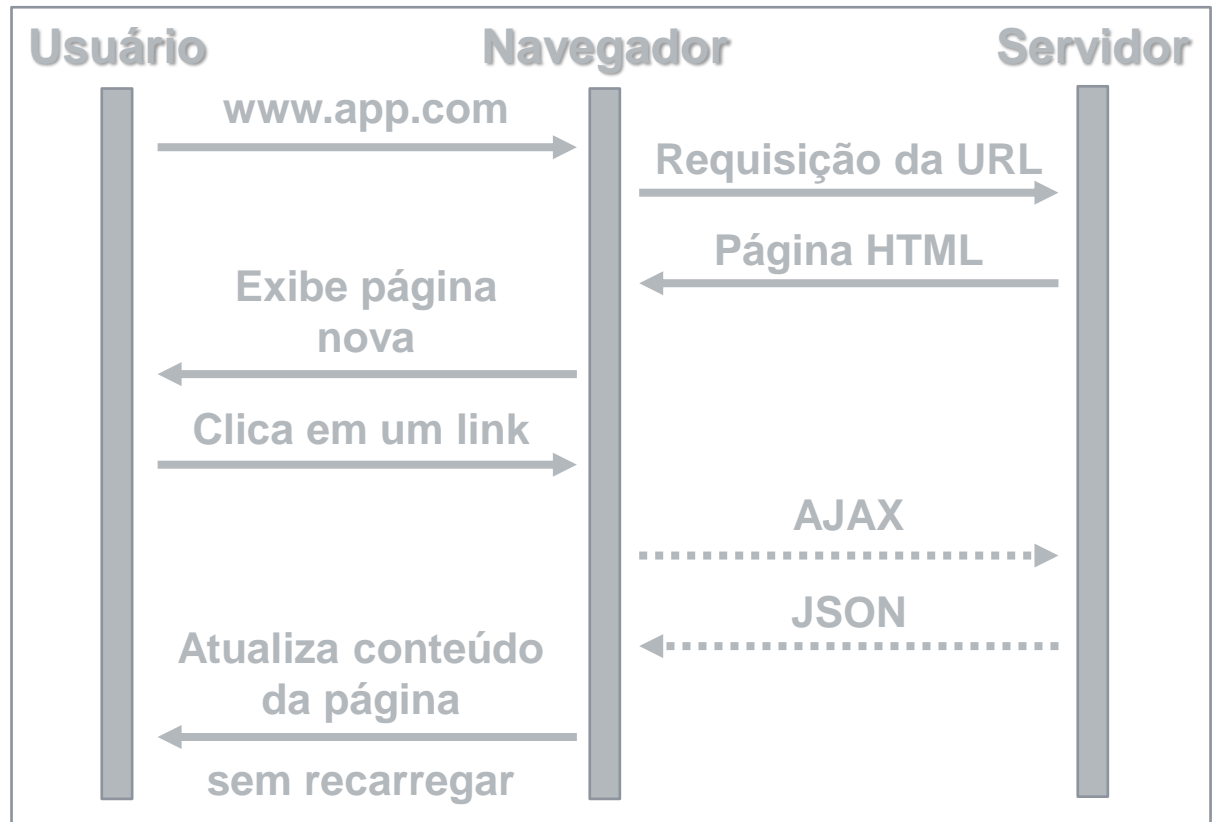
## Roteamento no servidor

Páginas HTML



## Roteamento no cliente

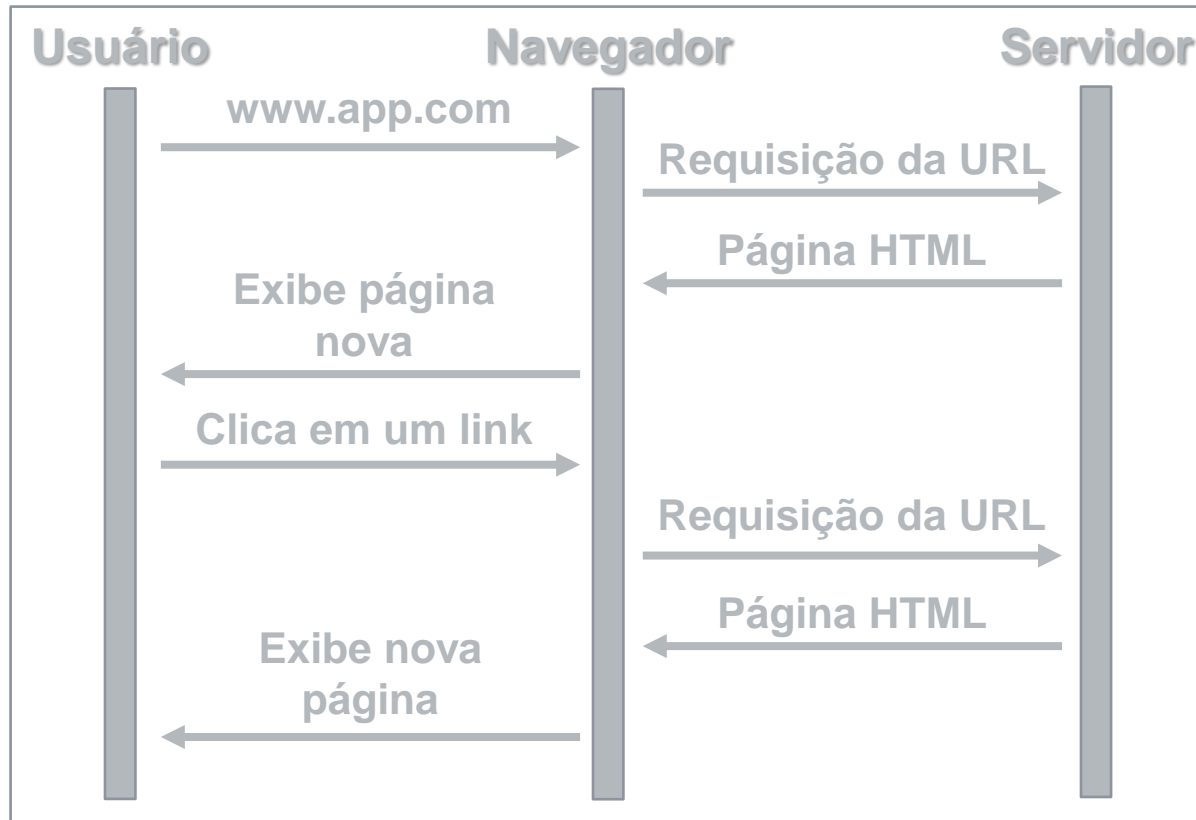
Single Page Application (SPA)



# Roteamento no lado cliente e no servidor

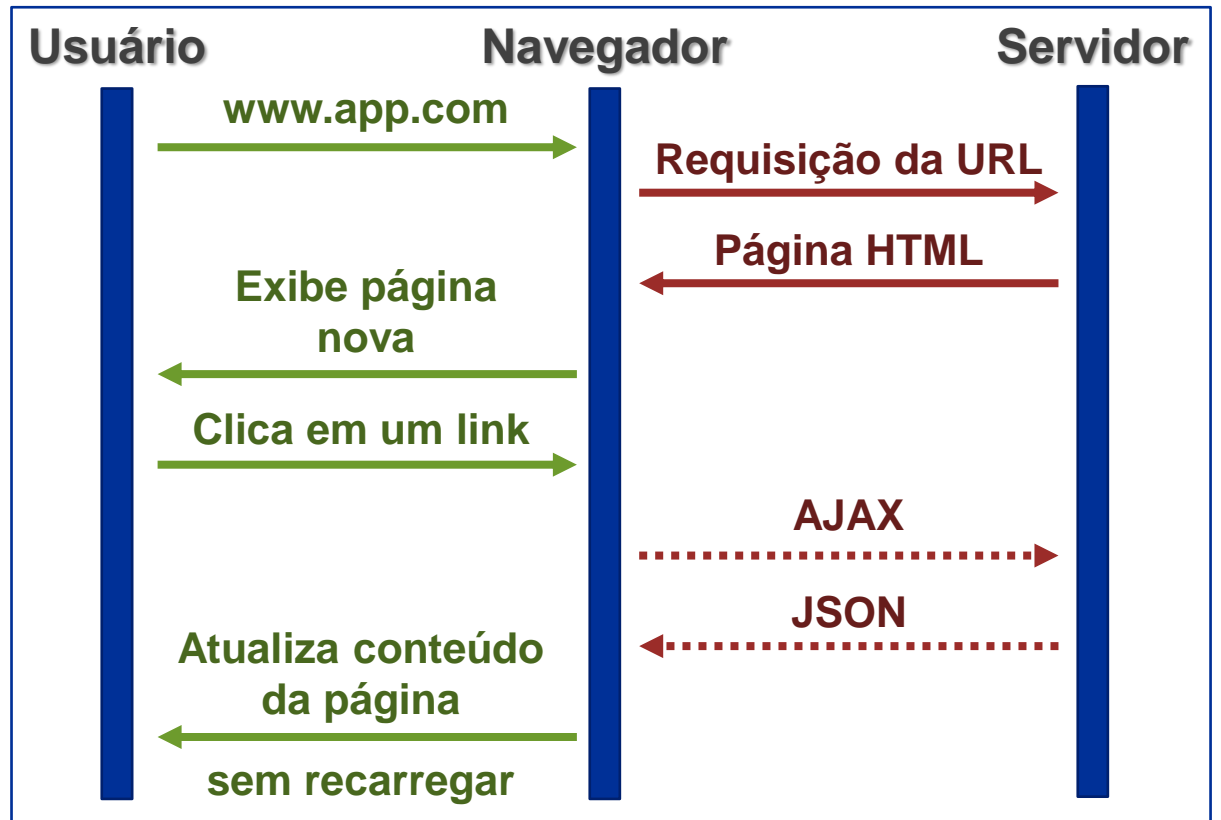
## Roteamento no servidor

Coleção de páginas HTML



## Roteamento no cliente

Single Page Application (SPA)



# Introdução ao TypeScript

- **TypeScript** é uma linguagem de programação desenvolvida pela Microsoft que pode ser considerada **superset do JavaScript**, adicionando algumas funcionalidades.
- Código JavaScript válido também é um código TypeScript válido.
- No processo de **compilação**, o código **TypeScript é convertido para JavaScript**.
- Sendo o mesmo código em tempo de execução, não existem problemas de compatibilidade nos navegadores ao migrar a aplicação de uma linguagem para outra.
- TypeScript = JavaScript + **Verificação de tipos** (análise estática).





# TypeScript: definição de tipos

## Inferência de tipos do JavaScript

1. `let stringA = "WebAcademy";`
2. `let stringB: string = "WebAcademy";`

  
*stringA* e *stringB*  
são do mesmo tipo.

## Definição de tipos


1. `class Usuario {`
2.     `id: number;`
3.     `nome: string;`
4. `}`
5. `const usuario: Usuario = {`
6.     `id: 1,`
7.     `nome: "Daniel"`
8. `};`

# TypeScript: definição de tipos

## Inferência de tipos do JavaScript

1. `let stringA = "WebAcademy";`
2. `let stringB: string = "WebAcademy";`

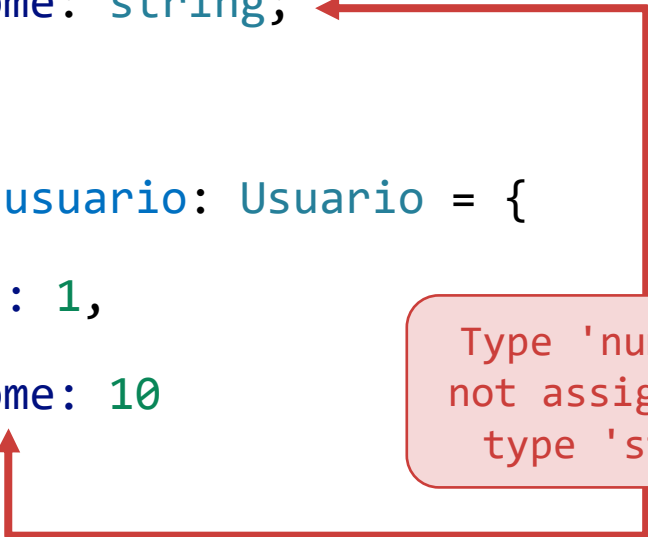
*stringA* e *stringB*  
são do mesmo tipo.



## Definição de tipos

1. `class Usuario {`
2.  `id: number;`
3.  `nome: string;`
4. `}`
5. `const usuario: Usuario = {`
6.  `id: 1,`
7.  `nome: 10`
8. `};`

Type 'number' is  
not assignable to  
type 'string'.



# TypeScript: definição de tipos

## Interface

```
1. interface Usuario {  
2.     id: number;  
3.     nome: string;  
4. }
```

## Type

```
1. type Usuario = {  
2.     id: number;  
3.     nome: string;  
4. }
```

- Diferença entre **Type** e **Interface**:

- <https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#differences-between-type-aliases-and-interfaces>
- <https://blog.logrocket.com/types-vs-interfaces-in-typescript/>

# Fundamentos

# O que é o Angular?

- **Framework criado pelo Google**, baseado em **TypeScript**.
- Ambiente de desenvolvimento, que inclui:
  - Uma **estrutura baseada em componentes** (facilita escalabilidade).
  - Uma **coleção de bibliotecas** integradas que cobrem uma ampla variedade de recursos, incluindo roteamento, gerenciamento de formulários, comunicação cliente-servidor, etc.
  - Um **conjunto de ferramentas** que ajudam na construção, teste e atualização do código.





# Angular CLI

- Ferramenta de **interface de linha de comando** usada para criar e manter aplicações Angular.
- Principais comandos:
  - **ng new**: cria um espaço de trabalho.
  - **ng build**: compila uma aplicação em um diretório de saída (dist/).
  - **ng serve**: compila a aplicação e cria um servidor, recompilando e atualizando os arquivos sempre que houver modificações.
  - **ng generate**: gera ou modifica arquivos com base em um esquema (componentes, classes, etc.).
- Referência de comandos: <https://angular.io/cli#command-overview>

```
ng new <nome-projeto>  
cd <nome-projeto>  
ng serve
```

# Espaços de trabalho

- Um **espaço de trabalho** (*workspace*) no Angular é o contexto **onde as aplicações são desenvolvidas**, podendo ser uma ou múltiplas aplicações no mesmo espaço de trabalho.
- Criação do espaço de trabalho:

```
ng new <nome_app> --routing=true --style=css --skip-git
```

- Múltiplos projetos: <https://angular.io/guide/file-structure#multiple-projects>

# Espaços de trabalho

- Um **espaço de trabalho** (*workspace*) no Angular é o contexto **onde as aplicações são desenvolvidas**, podendo ser uma ou múltiplas aplicações no mesmo espaço de trabalho.

- Criação do espaço de trabalho:  

Habilita o módulo de roteamento no projeto

Não inicializa repositório

```
ng new <nome_app> --routing=true --style=css --skip-git
```

Ttipo de estilo usado no projeto

- Múltiplos projetos: <https://angular.io/guide/file-structure#multiple-projects>

# Arquivos de configuração do espaço de trabalho

Arquivo/Diretório	Descrição
angular.json	Padrões de configuração do Angular CLI para todos os projetos no espaço de trabalho, incluindo localização de arquivos CSS e JavaScript.
package.json	Configurações aplicadas aos pacotes npm que estão disponíveis para todos os projetos no espaço de trabalho. Arquivo de configuração de projetos baseados em NodeJS, onde também são configurados os scripts NPM ( <i>start</i> , <i>build</i> , etc.).
src/	Código-fonte do projeto.
node_modules/	Local onde estão armazenados os pacotes npm para todo o espaço de trabalho.
tsconfig.json	Configuração básica do TypeScript para todos os projetos no espaço de trabalho. Todos os outros arquivos de configuração herdam deste arquivo base. A presença do arquivo em um diretório indica que o diretório é a raiz de um projeto TypeScript.

# Arquivos da aplicação (src/)

Arquivo/Diretório	Descrição
app/	Contém a lógica e os dados do projeto, incluindo componentes, templates HTML e arquivos de estilo (CSS).
assets/	Contém arquivos de imagem e outros conteúdos estáticos (CSS, JavaScript, etc.).
environments/	Contém opções de configuração de compilação para ambientes específicos. Por padrão, há um ambiente de desenvolvimento e um ambiente de produção ("prod").
index.html	A página HTML principal exibida quando a aplicação é acessada. Arquivos JavaScript e CSS são adicionados automaticamente no index.html no processo de <i>build</i> .
main.ts	Ponto de entrada da aplicação. Inicializa o módulo principal (AppModule) para ser executado no navegador.
styles.css	Reúne configurações CSS globais do projeto.



# Arquivos da aplicação (src/app/)

Arquivo/Diretório	Descrição
app.component.ts	Contém a lógica do componente principal da aplicação (AppComponent).
app.component.html	Define o documento HTML (template) associado ao AppComponent.
app.component.css	Define as propriedades estilo CSS para o AppComponent.
app.component.spec.ts	Arquivo criado para realizar teste de unidade do AppComponent.
app.module.ts	Define o módulo principal (AppModule), que informa ao Angular como montar a aplicação. Inicialmente declara apenas o AppComponent, e conforme vão sendo adicionados mais componentes na aplicação, eles devem ser declarados neste arquivo.

# Executando o projeto

- Iniciar a aplicação:

- **ng serve**

- Executar com protocolo SSL

habilitado:

- **ng serve --ssl**

- Sempre que um arquivo da aplicação for alterado, o projeto é recompilado e a modificação é refletida no cliente.

```
✓ Browser application bundle generation complete.

Initial Chunk Files | Names          | Raw Size
vendor.js           | vendor         | 2.34 MB
polyfills.js        | polyfills      | 294.84 kB
main.js             | main           | 249.94 kB
styles.css, styles.js | styles        | 179.17 kB
runtime.js          | runtime        | 6.51 kB
scripts.js          | scripts        | 1.60 kB

| Initial Total | 3.05 MB

Build at: 2022-05-03T15:56:42.382Z - Hash: 7305a34be157c203 - Time: 34690ms

** Angular Live Development Server is listening on localhost:4200, open your
browser on http://localhost:4200/ **

✓ Compiled successfully.
```

# Deploy

- O servidor de aplicação do Angular ([webpack](#)) não é recomendado para o modo de produção.
- Para um *deployment* simples são suficientes os seguintes passos:
  1. Compilar o projeto: **ng build** [--configuration production]
  2. Copiar todo o conteúdo do diretório **dist/<nome\_app>** para o servidor.
  3. (Opcional, mas importante) Configurar o servidor para **redirecionar solicitações de arquivos ausentes para index.html**. (<https://angular.io/guide/deployment#routed-apps-must-fallback-to-indexhtml>)
- Opções de *deployment* automático:
  - <https://angular.io/guide/deployment#automatic-deployment-with-the-cli>

# Componentes

# Visão geral dos componentes no Angular

- Os **componentes** são os **principais elementos** na construção de aplicações Angular.
- Cada componente consiste em:
  - Um **template HTML** que declara o que é renderizado na página.
  - Uma **classe TypeScript** que define o comportamento do componente.
  - Um **seletor CSS** que define como o componente é usado em um template.
  - **(Opcional) Estilos CSS** aplicados ao template.
- **AppComponent** (seletor *app-root*) é o **principal componente da aplicação**, sendo o primeiro a ser renderizado no cliente.

# Criação de componentes

- Novos componentes podem ser gerados pelo Angular CLI:
  - **ng generate component** <nome-componente>
- Por padrão, o seguinte conteúdo é gerado:
  - Uma **pasta com o nome do componente**.
  - Um **arquivo de componente**, .component.ts
  - Um **arquivo de template**, .component.html (*--inline-template* impede criação)
  - Um **arquivo CSS**, .component.css (*--inline-style* impede criação)
  - Um **arquivo de especificação de teste**, .component.spec.ts (*--skip-tests* impede criação)

# Criação de componentes

```
1.  import { Component } from '@angular/core';

2.  @Component({
3.      selector: 'app-root',
4.      templateUrl: './app.component.html',
5.      styleUrls: ['./app.component.css']
6.  })
7.  export class AppComponent {
8.      constructor() {}
9.  }
```

# Templates

- No Angular, um template é uma parte de um HTML, não o arquivo completo (modularização).
- Fornece uma série de recursos:
  - Mostrar *strings* geradas dinamicamente.
  - Atribuir eventos para elementos HTML.
  - Transformação de dados com os *pipes*.
  - Alterar propriedades do elementos HTML dinamicamente (uni e bidirecional).
  - Variáveis de template.



# Templates: *strings* geradas dinamicamente

- O Angular vai interpretar como uma expressão tudo entre os delimitadores `{{ }}`.
- Exemplo:
  - `src/app/app.component.ts`
    - `title = 'SGCM';`
  - `src/app/app.component.html`
    - `<div>{{ title }}</div>`
- Referência: <https://angular.io/guide/interpolation>

# Templates: *pipes*

- *Pipes* podem usados para formatar *strings*.
- Exemplo:
  - `<div>{{ data | date:'dd/MM/yyyy' }}</div>`
- Referência: <https://angular.io/guide/pipes>

# Templates: atribuição de eventos

- No Angular, para associar um evento a um elemento HTML, é necessário utilizar os delimitadores ().
- Exemplo:
  - `<a (click)="delete(id)">Remover</a>`
- O método `delete()` deve ser declarado no arquivo TypeScript que controla o componente.
- Referência: <https://angular.io/guide/event-binding>

# Templates: alterar propriedades

- A atribuição de valores dinâmicos para propriedades de elementos HTML pode ser **unidirecional ou bidirecional**, onde além atribuir valor à propriedade, é possível associar a um evento que atualiza valores de objetos compartilhados (por exemplo, um modelo que representa um usuário).
- Atribuição unidirecional utiliza os delimitadores [ ], e a bidirecional combina também os delimitadores de eventos [( )].
- Exemplo: `<input type="submit" value="Salvar" [disabled]="form.invalid">`  
`<input type="hidden" name="id" [(ngModel)]="usuario.id">`
- Referência: <https://angular.io/guide/property-binding>

# Templates: variáveis

- As variáveis de template ajudam a identificar um elemento em qualquer parte do template, sendo sempre declaradas caractere #.
- Exemplo:
  1. `<form #form>`
  2. `<input type="submit" value="Salvar" [disabled]="form.invalid">`
  3. `</form>`
- Referência: <https://angular.io/guide/template-reference-variables>

# Rotas

- Em SPAs é necessário controlar a navegação entre as diferentes visualizações da aplicação.
- Para proporcionar uma melhor experiência ao usuário, a **rotas representam caminhos únicos para estas visualizações**, de forma que o usuário pode navegar entre elas como se fizesse requisições de novas páginas ao servidor.

```
1.  const routes: Routes = [  
2.    { path: 'agenda',  
3.      component: AgendaListComponent },  
4.    { path: 'agenda/form',  
5.      component: AgendaFormComponent }  
6.  ];  
7.  @NgModule({  
8.    imports: [RouterModule.forRoot(routes)],  
9.    exports: [RouterModule]  
10.  })  
11.  export class AppRoutingModule { }
```

# Diretivas

Diretiva	Descrição
NgClass	Atribui uma ou mais classes para um elemento HTML.
NgModel	Usada para exibir propriedades de objetos e atualizar estes valores nos objetos quando houver modificações.
NgForm	Representa uma instância de um objeto que permite manipular o comportamento de um formulário.
*NgIf	Controla a exibição de elementos HTML.
*NgFor	Repete a exibição de um bloco HTML para cada elemento de uma coleção.

Mais sobre diretivas: <https://angular.io/guide/built-in-directives>

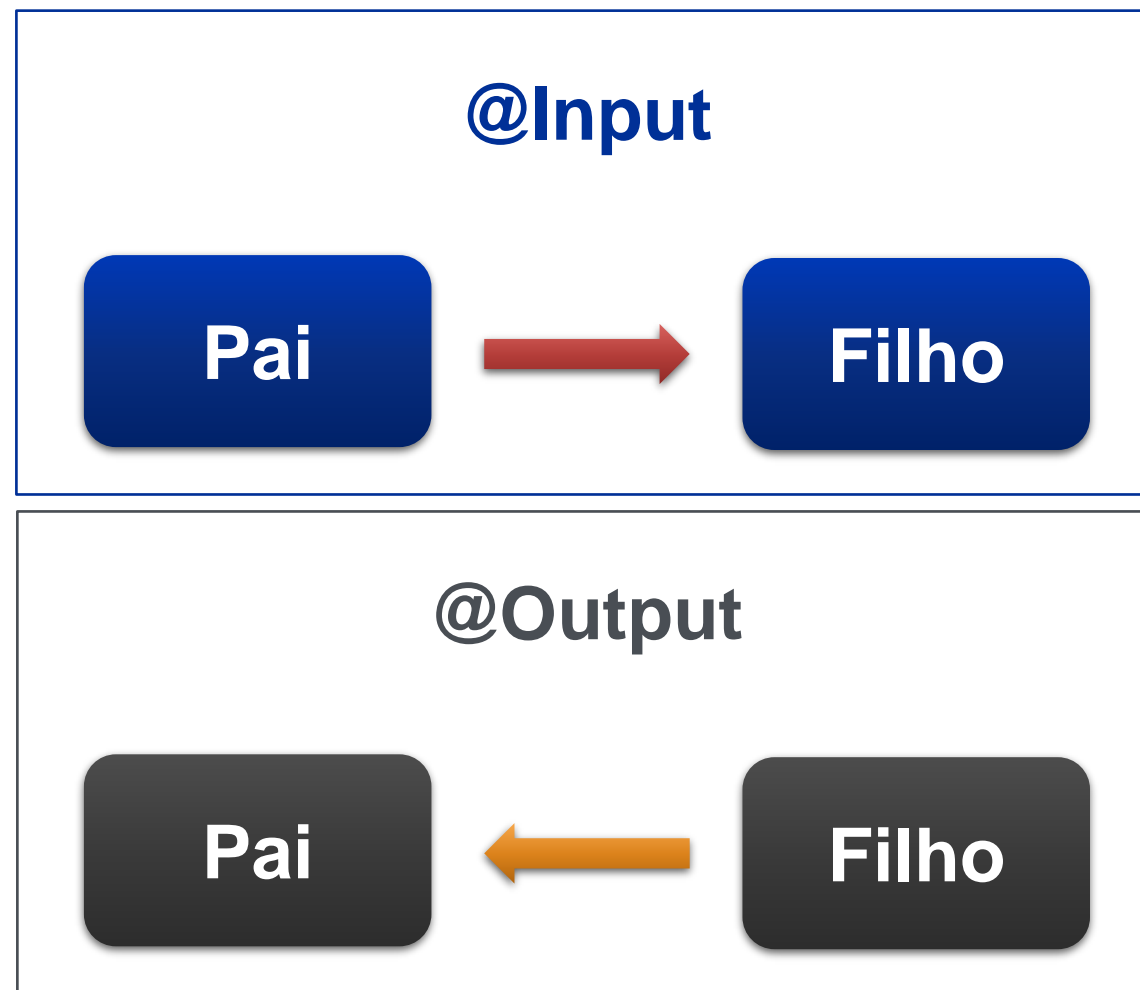
# Compartilhamento de dados entre componentes

- No Angular é possível compartilhar dados entre um componente pai e um ou mais componentes filhos, por meio dos **@Input()** e **@Output()**.

```
<component-pai>
```

```
  <componente-filho></componente-filho>
```

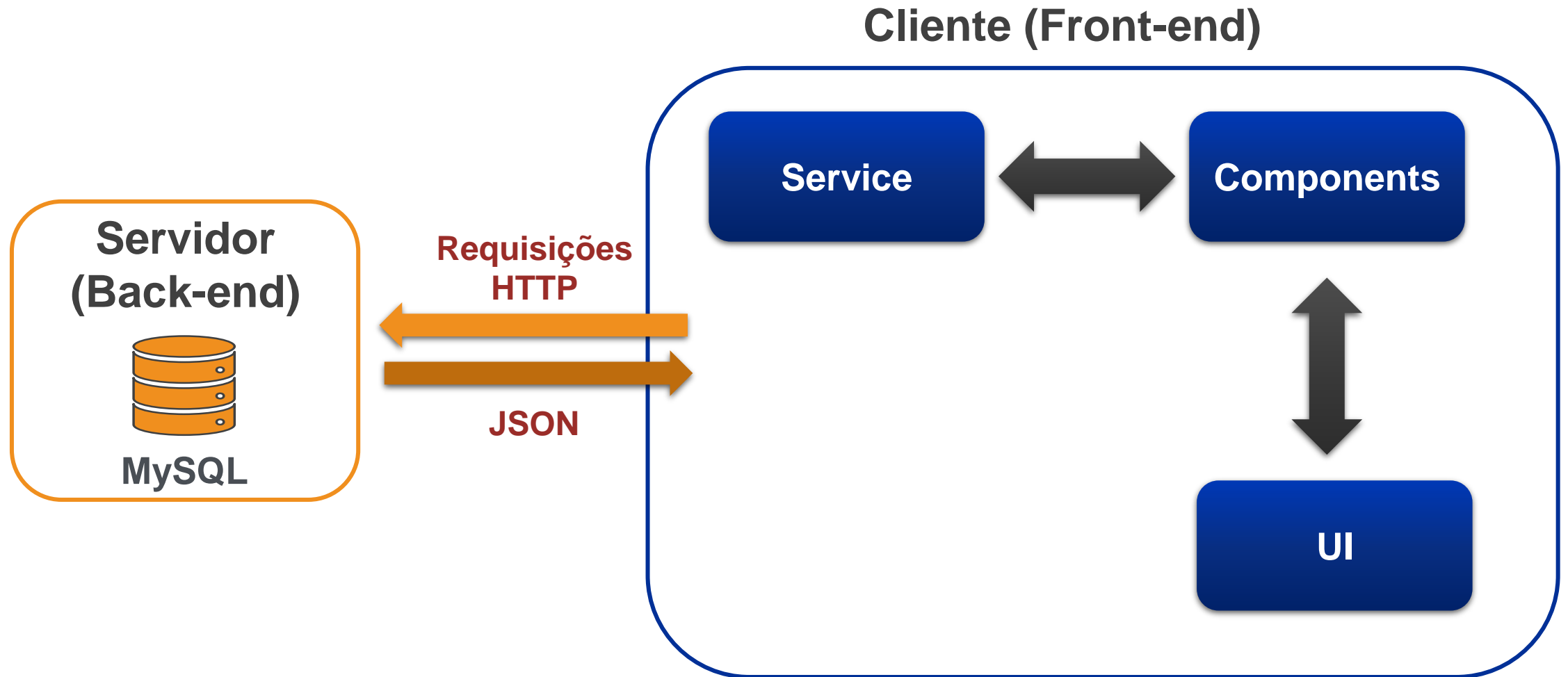
```
</component-pai>
```





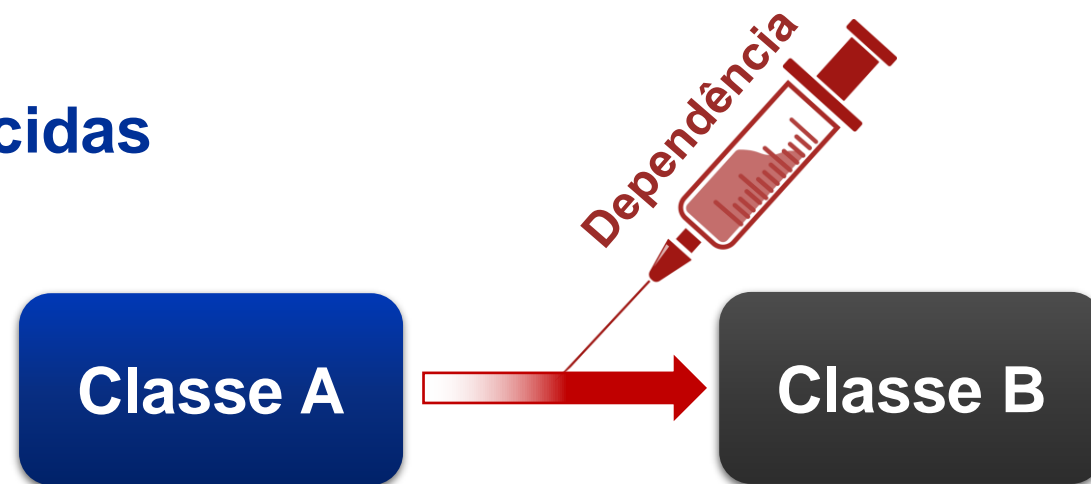
# Comunicação com o back-end

# Arquitetura de uma aplicação front-end



# Injeção de dependência no Angular

- **Dependências** são serviços ou objetos que uma classe precisa para desempenhar sua função.
- **Injeção de dependência** é um padrão no qual uma classe solicita dependências de fontes externas em vez de criá-las.
- No Angular, as **dependências são fornecidas para uma classe na instanciação** (por meio do método construtor).



# Injeção de dependência no Angular

```
1. import { Injectable } from '@angular/core';
2.
3. @Injectable({
4.   providedIn: 'root'
5. })
6. export class LoginService {
7.   constructor() {}
8. }
```

***AppComponent depende  
de LoginService***

```
1. import { Component } from '@angular/core';
2. import { LoginService } from 'login.service';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   constructor(
10.     private login: LoginService) {}
11. }
```

# HTTP Client

- Numa **arquitetura onde o front-end é uma aplicação independente**, é necessário estabelecer um **canal de comunicação**, normalmente por meio do **protocolo HTTP**, para ter acesso a dados e serviços de uma aplicação back-end.
- No Angular, a classe **HttpClient** (@angular/common/http) fornece recursos para realizar esta comunicação de forma assíncrona com a vantagem de **suportar atribuição de um tipo específico para a resposta da requisição HTTP**, além de facilitar tratamento de erros e permitir interceptação das requisições.
- A classe HTTP Client possui métodos para os diferentes tipos de requisição: **HttpClient.get()**, **post()**, **put()**, **delete()**, etc.

# Observable

- Um **Observable** é uma **coleção que retorna os valores de seu itens** (um por vez) **por meio de notificações** (unidirecional), e de forma assíncrona ou síncrona.
- Os valores do itens são acessado por meio do método **subscribe()**, que se assemelha a chamada de funções, porém podendo **retornar múltiplos valores ao longo do tempo**.
- Não é o mesmo que um **EventListener**.
  - Um *Observable* não mantém uma lista de seus “observadores”.

# Observable

```
1. import { Observable } from 'rxjs';
2.
3. const obs = new Observable(objeto => {
4.   objeto.next(1); ←----- Notificação
5.   objeto.next(2); ←-----
6.   setTimeout(() => {
7.     objeto.next(3); ←-----
8.     objeto.complete(); ←-----
9.   }, 1000);
10. });
```



```
1. obs.subscribe({
2.   next: (x) => {console.log('Valor: '+x);},
3.   error: (erro) => {console.error(erro);},
4.   complete: () => {console.log('Fim.')}});
5.
6. console.log('Após o subscribe.');
```

Valor: 1

Valor: 2

Após o subscribe.

Valor: 3

Fim.

Antes do fim  
da execução  
do Observable



**Continua...**