



Angular

Introdução



O que é?

- Framework para aplicações clientes com HTML, CSS e JavaScript / TypeScript
- AngularJS (1.x - 2010) e Angular (2+ - 2016)
- Parceria Google + Microsoft (Open Source - github)
- Uso de padrões web e Web Components
- Orientado a componente (componente dentro de componente)



O que é?

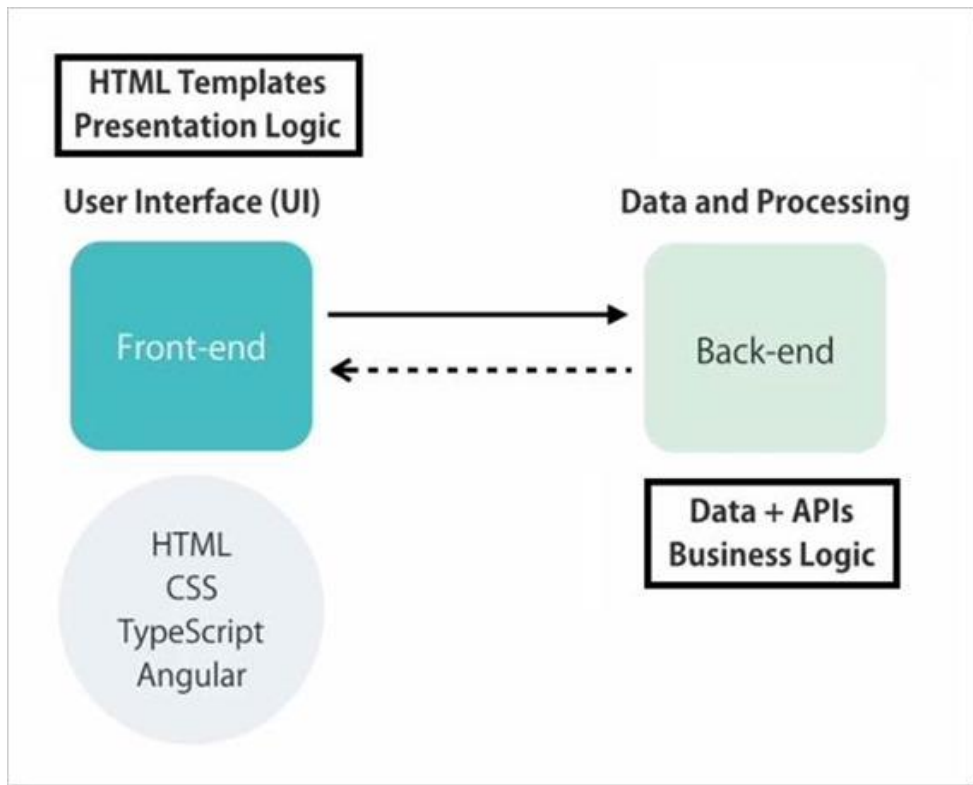
- Framework para aplicações clientes com HTML, CSS e JavaScript / TypeScript
- AngularJS (1.x - 2010) e Angular (2+ - 2016)
- Parceria Google + Microsoft (Open Source - github)
- Uso de padrões web e Web Components
- Orientado a componente (componente dentro de componente)

Benefícios em usar

- Estrutura limpa
- Re-utilização de códigos
- Permite aplicativo ser mais “testável”
- Facilita o desenvolvimento do aplicativo

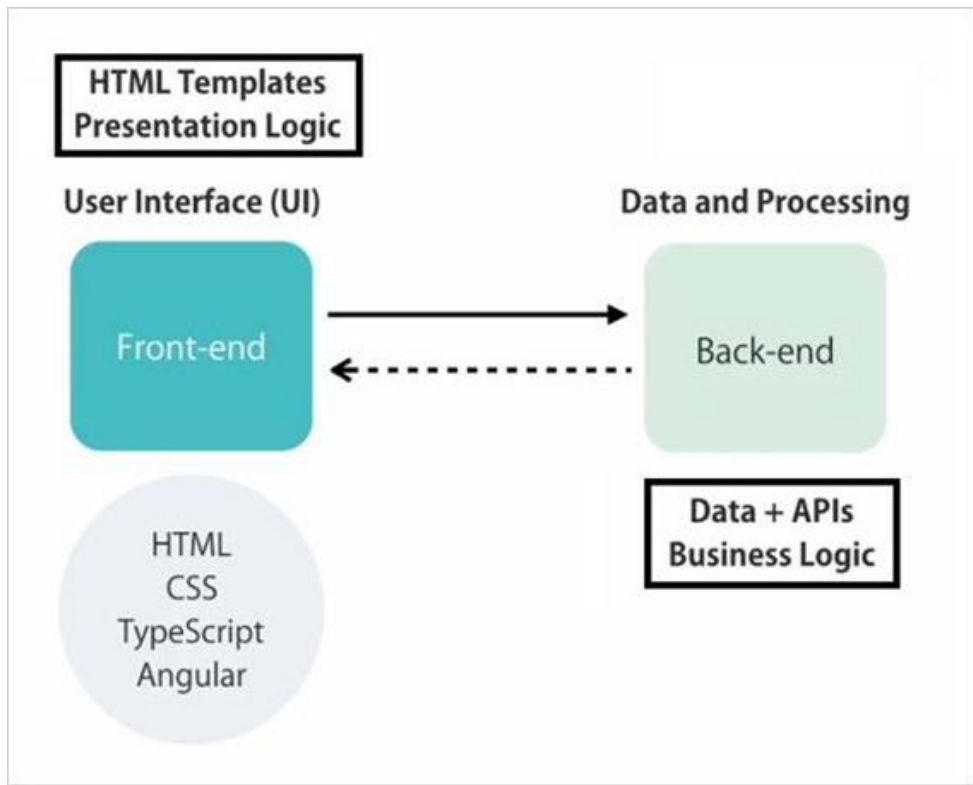


Arquitetura geral





Arquitetura geral



API

- Forma de integrar sistemas
- Permite segurança dos dados
- Facilidade no intercâmbio entre informações (mesmo entre tecnologias diferentes)



Setup básico do nosso Ambiente

Node

- Executar javascript fora do browser
- Ferramentas utilizadas no angular para criar/build dos projetos
- <https://nodejs.org>





Setup básico do nosso Ambiente

Node

- Executar javascript fora do browser
- Ferramentas utilizadas no angular para criar/build dos projetos
- <https://nodejs.org>



Angular CLI

- Criação de projetos e códigos padrões/bases
- `npm install -g @angular/cli`
- <https://cli.angular.io/>





Setup básico do nosso Ambiente

TypeScript

- Responsável por compilar .ts → .js
- `npm install -g typescript`
- <https://www.typescriptlang.org/>





Setup básico do nosso Ambiente

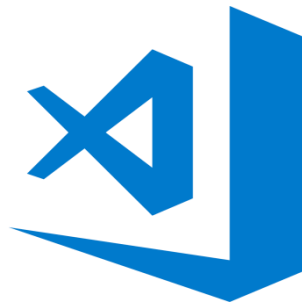
TypeScript

- Responsável por compilar .ts → .js
- `npm install -g typescript`
- <https://www.typescriptlang.org/>



Visual Studio Code

- IDE sensacional, leve e prática!
- <https://code.visualstudio.com>





Hello world!

1) ng new nome-aplicativo

- Cria novo diretório e gera template básico da aplicação angular
- Executa o npm install para baixar bibliotecas

```
C:\WINDOWS\system32\cmd.exe

D:\Projetos\Angular>ng new hello-world_
```

2) ng serve

- Compila projeto e inicia servidor
- Escuta alterações de arquivos

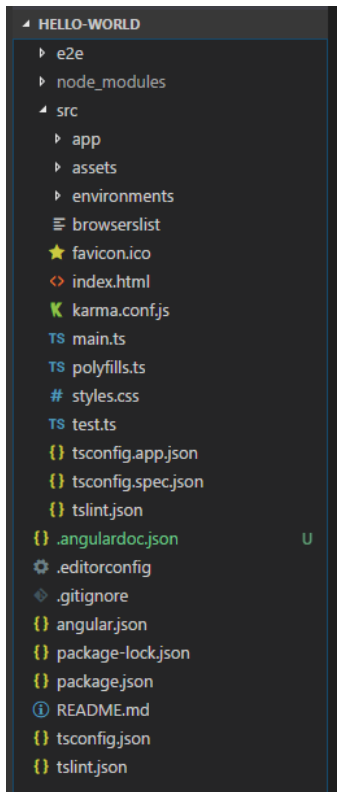
```
C:\ ng

D:\Projetos\Angular>cd hello-world
D:\Projetos\Angular\hello-world>ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser at http://localhost:4200/

Date: 2018-06-28T20:33:05.055Z
Hash: e8e3505bb172de5054c9
Time: 6068ms
chunk {main} main.js, main.js.map (main) 10.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 227 kB [initial]
chunk {runtime} runtime.js, runtime.js.map (runtime) 5.22 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 15.6 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.07 MB [initial] [rendered]
i @wdm: Compiled successfully.
```



Estrutura do projeto



- **e2e** - testes "end to end" pasta de teste automatizados de interface
- **node_modules** - códigos de bibliotecas são baixados para desenvolvimento
- **src** - código do aplicativo
- **src/app** - módulos e components
- **src/assets** - arquivos estáticos como imagens
- **src/environments** - configuração de variáveis para ambientes diferentes
- **src/index.html** - arquivo básico
- **src/main.ts** - arquivo inicial
- **src/polyfills.ts** - importa scripts para rodar o angular em outros browsers
- **src/styles.css** - arquivo global de css
- **package.json** - arquivo base do projeto que indica dados do projeto, suas dependências e versões utilizadas



TypeScript

Introdução

TS TypeScript

```
interface IComponent{
    getId() : string;
}

class Button implements IComponent{
    id:string;
    getId():string{
        return this.id;
    }
}
```

```
var Button = (function () {
    function Button() {
    }
    Button.prototype.getId = function () {
        return this.id;
    };
    return Button;
})();
```

- Desenvolvido pela Microsoft
- Muito parecido com o C# (tipagem, orientação a objetos, erros de compilação, etc)
- Utiliza Decorators (Anotações)
- Transpilation que transforma o código TS em JS
- Prometem ser fiel ao futuro do JS
- **É TUDO OPCIONAL**



TypeScript

Declarando variáveis

- `let i = 5;`

Types (number, boolean, string, any, enum)

- `let i = 5; let i:number;`
- `enum Status { Aguardando = 0, Liberado = 1, Impresso = 2 }`
- `let text = (<string>msg).toLowerCase()` e `let text = (msg as string).toLowerCase()`

Arrow functions

- `let geraLog = (msg) => console.Log(msg);`



TypeScript

Interfaces

- `interface Point { x:number, y:number }`

Classes

- `class Point { x:number; y:number; getDistancia(outroPonto: Point){ } }`

Parâmetros no construtor

- `constructor(private x?:number) { }`

Propriedades

- `get X() set X()`



TypeScript

Modulos

- `export class Point`
- `import { Point } from './point';`



Angular

Fundamentos



Blocos Principais

COMPONENTES

DIRETIVAS

ROTEAMENTO

SERVIÇOS

TEMPLATE

METADATA

DATA BINDING

INJEÇÃO
DEPENDÊNCIA



Componente

Componente



Encapsula:

- Template
- Dado a ser mostrado na tela (Data Binding, associação dos dados do componente como html)
- Lógica da view (não deve conter outra lógica além do comportamento da view)



Componente



Cabeçalho (Barra de navegação)



Barra
Lateral



Posts



Post



Post

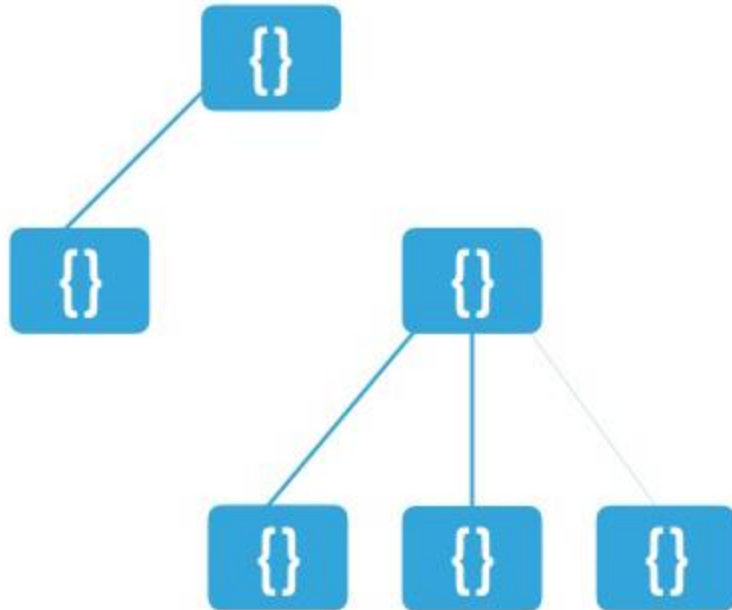


Post



Componente

Componente Raiz (Root)





Componentes

Componente



Backend



Node.JS

Java

.NET

Ruby

Python



Serviço

Contém a lógica de negócio que pode ser injetada em vários componentes





Rotas

Router

Responsável pelo roteamento das páginas



- Angular utiliza o conceito SPA
- Responsável pela navegação (entre telas)



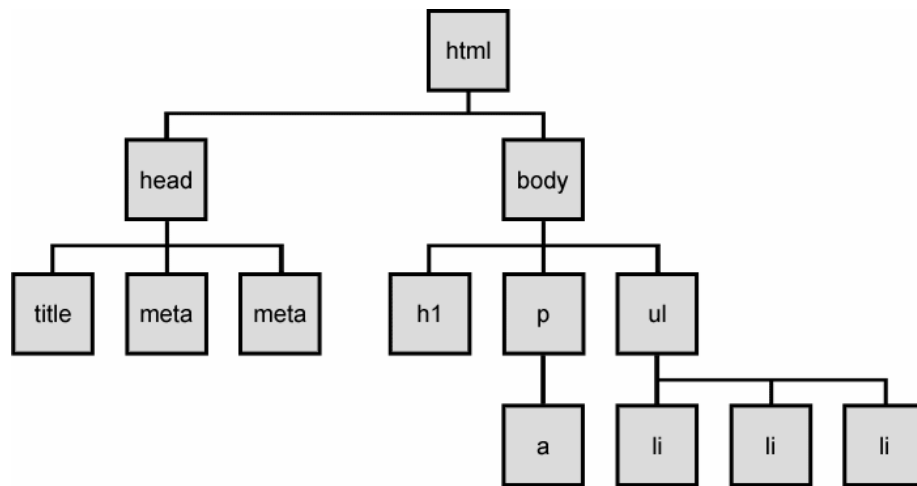
Diretivas

Diretiva



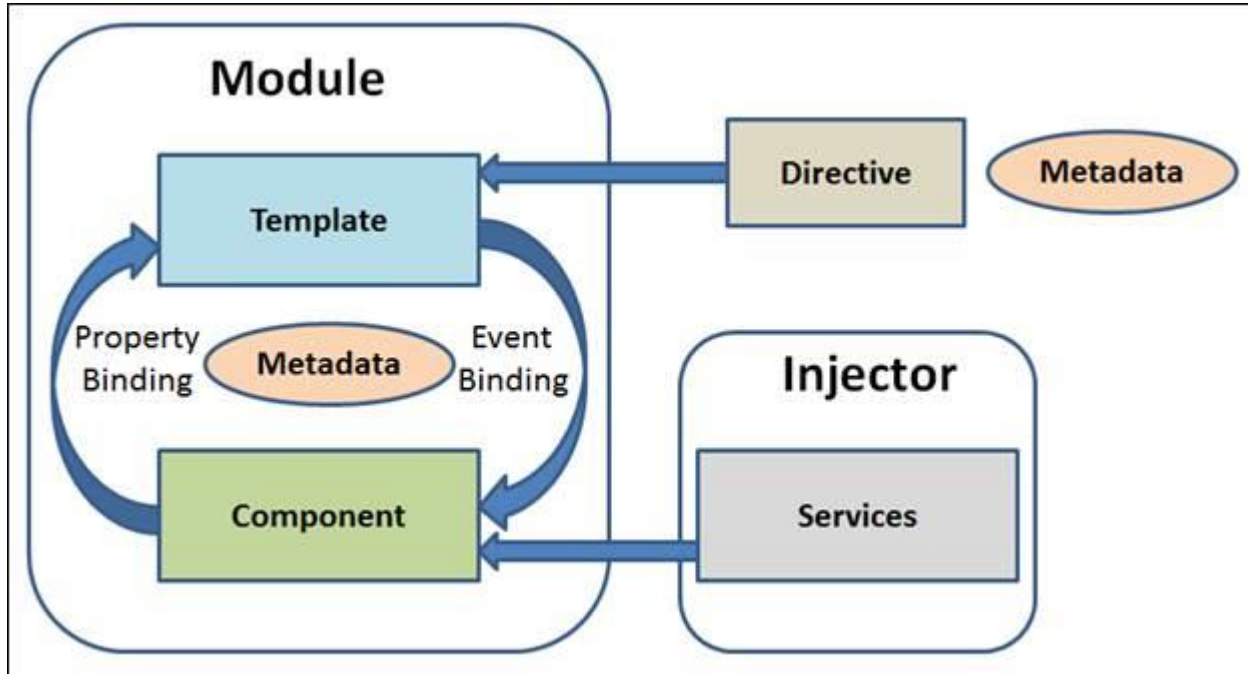
Responsável por modificar elementos do DOM e/ou seu comportamento

- Componentes
- Diretivas Estruturais
- Diretivas Atributos





Blocos Principais





Angular

Componentes



Componente

Novo componente

1. Criar o componente;
2. Registra-lo no modulo (declarations);
3. Adicionar componente no html;



Componente

`meu-primeiro.component.ts`

Palavras separadas por "-"

Ponto

"component" -> para indicar que é um componente

Ponto

ts -> extensão typescript



Componente

```
meu-primeiro.component.ts x
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-meu-primeiro',
5    template: '<h1>Meu primeiro componente</h1>',
6  })
7  export class MeuPrimeiroComponent {
8
9    constructor() { }
10
11  }
12
```



Componente

Novo componente com Angular CLI

- `ng generate component nome-componente`
- `ng g c nome-componente`

```
D:\Projetos\Angular\hello-world>ng g c courses
CREATE src/app/courses/courses.component.html (26 bytes)
CREATE src/app/courses/courses.component.spec.ts (635 bytes)
CREATE src/app/courses/courses.component.ts (273 bytes)
CREATE src/app/courses/courses.component.css (0 bytes)
UPDATE src/app/app.module.ts (400 bytes)

D:\Projetos\Angular\hello-world>
```



Componente

- selector
- template
- templateUrl
- styles
- styleUrls
- data-binding - Interpolation

TS courses.component.ts x

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-courses',
5    templateUrl: './courses.component.html',
6    styleUrls: ['./courses.component.css']
7  })
8  export class CoursesComponent {}
9
10 constructor() { }
11
12
```




Angular

Templates



Templates

O que é?

- Parte do componente responsável pela interface do usuário/view
- Código HTML estático/dinâmico
- Representado no próprio componente ou em um arquivo a parte



Templates - Interpolation

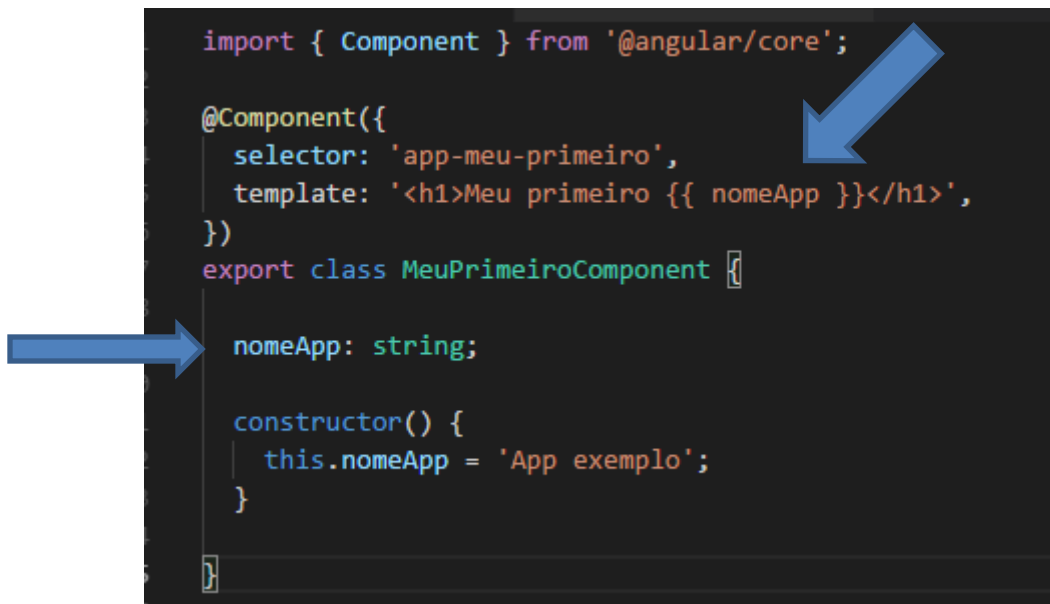
- Forma utilizada para enviar dados do componente para a view
- Permite “enviar” propriedades, métodos ou dados calculados

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

@Component({
  selector: 'app-meu-primeiro',
  template: '<h1>Meu primeiro {{ nomeApp }}</h1>',
})
export class MeuPrimeiroComponent {

  nomeApp: string;

  constructor() {
    this.nomeApp = 'App exemplo';
  }
}
```





Templates - Diretivas

Exemplos de diretivas:

- ngIf
- ngFor

TS courses.component.ts x

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-courses',
5    template: `
6      <h2>{{ title }}</h2>
7      <ul>
8        <li *ngFor='let curso of cursos; let i = index'>
9          | {{ curso }}<span *ngIf='i % 2 === 0'> - Matriculado!</span>
10         </li>
11      </ul>`
12  })
13  export class CoursesComponent {
14    title = 'Lista de cursos';
15    cursos = ['.net', 'asp.net', 'angular'];
16
17    constructor() { }
18  }
19
```



Angular

Data Binding

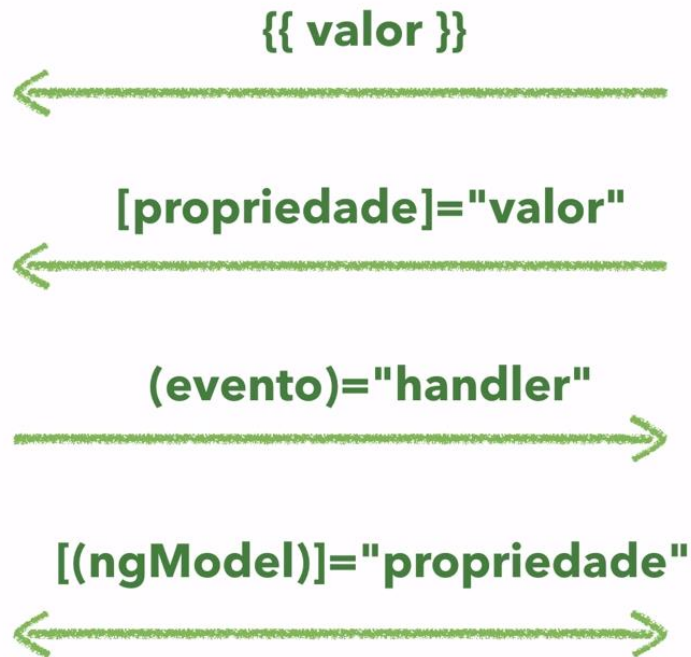


Data binding

O que é?

- Associar dados/informações entre o component e o template
- Dados são atualizados em tempo real

<TEMPLATE>





Data binding - Interpolation

- Valor do component para o template
- É um tipo de property binding
- Permite o uso direto da propriedade
- Permite o uso de expressões como:
 - `{{ 1 + 1 }}`
 - `{{ getValor() }}`
 - `{{ var1 && var2 }}`



Data binding - Property Binding

- Valor do componente para o template
- Usa-se colchetes ``
- Pode ser utilizado também com ``
- Quando não existe uma propriedade no elemento, usa-se `[attr.colspan]`

```
<h3>Property binding</h3>

<img [src]="urlImg" />

```




Data binding - Property Binding

- Valor do componente para o template
- Usa-se colchetes ``
- Pode ser utilizado também com ``
- Quando não existe uma propriedade no elemento, usa-se `[attr.colspan]`

```
<h3>Property binding</h3>

<img [src]="urlImg" />

```

Todas imagens tem
o mesmo resultado



Data binding - Class e Style Binding

- `[class.active]="condicao"`
- `[style.backgroundColor]="condicao ? 'blue' : 'red'"`

```
selector: 'courses',
template: `
  <button class="btn btn-primary" [class.active]="isActive">
  `
}

import class CoursesComponent {
  isActive = true;
```



Data binding - Event Binding

- Valor do Template para o Componente
- Usa-se parênteses
`<button (click)="onClick()" />`
- Método “handler” declarado no componente

```
<button class="btn" (click)="onClick()">Me clique!</button>
```



```
export class DataBindingComponent {  
  
  onClick() {  
    alert('Botão clicado!');  
  }  
}
```



Data binding - Event Binding

Exemplo passando valores

```
<input type="text"  
  (keyup)="onKeyUp($event)"  
  (keyup.enter)="onSave(input.value)"  
  (blur)="onSave(input.value)"  
#input  
>
```

\$event é do tipo
evento DOM

acesso a variável
local declarada



Data binding - Event Binding

Exemplo passando valores

Variável de
template

```
<input type="text"  
  (keyup)="onKeyUp($event)"  
  (keyup.enter)="onSave(input.value)"  
  (blur)="onSave(input.value)"  
#input  
>
```

\$event é do tipo
evento DOM

acesso a variável
local declarada



Data binding - Event Binding

Exemplo passando valores

```
<input type="text"
  (keyup)="onKeyUp($event)"
  (keyup.enter)="onSave(input.value)"
  (blur)="onSave(input.value)"
  #input
>
```

Event filtering

\$event é do tipo
evento DOM

acesso a variável
local declarada



DICA - Event Binding

Quais eventos posso usar?

Lista com todos os eventos - <https://developer.mozilla.org/en-US/docs/Web/Events>

Event reference | MDN

Seguro | <https://developer.mozilla.org/en-US/docs/Web/Events>

MDN web docs
moz://a

Technologies ▾ References & Guides ▾ Feedback ▾

Sign in

Event reference

Languages Edit

Web technology for developers >
Event reference

Related Topics

Events

- ▶ Ambient Light events
- ▶ App Cache events

DOM Events are sent to notify code of interesting things that have taken place. Each event is represented by an object which is based on the [Event](#) interface, and may have additional custom fields and/or functions used to get additional information about what happened. Events can represent everything from basic user interactions to automated notifications of things happening in the rendering model.

This article offers a list of events that can be sent; some are standard events defined in official specifications, while others are events used internally by specific browsers; for the Mozilla...



Data binding - Two-Way

- Valor do Template para o Componente
- Valor do Componente para o Template
- Usa-se binding de eventos + propriedades



Data binding - Two-Way

- Valor do Template para o Componente
- Valor do Componente para o Template
- Usa-se binding de eventos + propriedades

```
<input type="text"  
      [value]="nome"  
      (input)='nome = $event.target.value' />
```



Data binding - Two-Way

- Valor do Template para o Componente
- Valor do Componente para o Template
- Usa-se binding de eventos + propriedades

```
<input type="text"  
      [value]="nome"  
      (input)='nome = $event.target.value' />
```

- Forma mais simples é fazer o binding no `ngModel`
- Usa-se a sintaxe de binding de eventos + propriedades (banana na Caixa)



Data binding - Two-Way

- Valor do Template para o Componente
 - Valor do Componente para o Template
 - Usa-se binding de eventos + propriedades
-
- Forma mais simples é fazer o binding no ngModel (precisa do import FormsModule no AppModule para ser usado)
 - Usa-se a sintaxe de binding de eventos + propriedades

```
<input type="text"
      [value]="nome"
      (input)='nome = $event.target.value' />
```

```
<input type="text" [(ngModel)]="nome" />
```



Data binding - Banana in a Box

Sintaxe “Banana na Caixa”





Angular

Componentes reutilizáveis



Componentes reutilizáveis - Input Property

O que é?

- Enviar dados para um componente

Como?

- Utilizar o decorator Input
- Determinar nome “externo” (opcional)
- Realizar o property binding do componente pelo Input definido

```
@Component({
  selector: 'app-curso',
  templateUrl: './input-property.component.html',
  styleUrls: ['./input-property.component.css'],
  inputs: ['nomeCurso:nome']
})
export class InputPropertyComponent implements OnInit {

  @Input('nome') nomeCurso: string = '';

  constructor() { }

  ngOnInit() {
  }
}
```



Componentes reutilizáveis - ngContent

O que é?

- Receber dados de um componente através de pontos de injeção

Como?

- Definir o conteúdo como ng-content
- Definir um seletor (select) para identificar o ponto de injeção
- Event binding no componente “pai” passando \$event para receber valor

```
<div class="panel panel-default">
  <div class="panel-heading">
    <ng-content select=".heading"></ng-content>
  </div>
  <div class="panel-body">
    <ng-content select=".body"></ng-content>
  </div>
</div>
```

```
<bootstrap-panel>
  <div class="heading">Heading</div>
  <div class="body">
    <h2>Body</h2>
    <p>Some content here...</p>
  </div>
</bootstrap-panel>
```



Componentes reutilizáveis - ngContainer


O que é?

- Renderiza somente o conteúdo no ng-content

Como?

- Da mesma forma utilizada anteriormente, mas utilizando o marcador ng-container.

```
<div class="panel panel-default">
  <div class="panel-heading">
    <ng-content select=".heading"></ng-content>
  </div>
  <div class="panel-body">
    <ng-content select=".body"></ng-content>
  </div>
</div>
```



```
<bootstrap-panel>
  <ng-container class="heading">Heading</ng-container>
  <div class="body">
    <h2>Body</h2>
    <p>Some content here...</p>
  </div>
</bootstrap-panel>
```




Angular

Ciclo de vida do Componente



Ciclo de vida do componente

Angular tem alguns eventos que são disparados durante seu ciclo de vida

#	EVENTO (HOOKS)	QUANDO?
1	ngOnChanges	antes #2 e quando valor property-binding é atualizado
2	ngOnInit	quando Component é inicializado
3	ngDoCheck	a cada ciclo de verificação de mudanças
4	ngAfterContentInit	depois de inserir conteúdo externo na view
5	ngAfterContentChecked	a cada verificação de conteúdo inserido
6	ngAfterViewChecked	a cada verificação de conteúdo / conteúdo filho
7	ngOnDestroy	antes da diretiva/component ser destruído



Angular

Pipes



Pipes

- Forma utilizada no Angular para transformar um valor quando exibidos no HTML
- Obtem um valor de entrada, realiza a transformação, exibe valor transformado
- Alguns exemplos:
 - Uppercase
 - Lowercase
 - Decimal
 - Currency
 - Percent

```
TS courses.component.ts x
4     selector: 'courses',
5     template: `
6         {{ course.title | uppercase | lowercase }} <br/>
7         {{ course.students | number }} <br/>
8         {{ course.rating | number:'2.1-1' }} <br/>
9         {{ course.price | currency:'AUD':true:'3.2-2' }} <br/>
10        {{ course.releaseDate | date:'shortDate' }}`
11    })
12    export class CoursesComponent {
13        course = {
14            title: "The Complete Angular Course",
15            rating: 4.9745,
16            students: 30123,
17            price: 190.95,
18            releaseDate: new Date(2016, 3, 1)
19        }
20    }
```



Pipes

Site <https://angular.io> possui mais detalhes de cada pipe e seus possíveis argumentos

The screenshot shows the Angular documentation website for the DatePipe. The browser window has a single tab titled 'Angular - DatePipe' and the address bar shows 'https://angular.io/api/common/DatePipe'. The website's navigation bar includes links for GETTING STARTED, TUTORIAL, FUNDAMENTALS, TECHNIQUES, and API. The main content area is titled 'DatePipe' and includes a description: 'Formats a date value according to locale rules.' Below this is the pipe signature: `{{ value_expression | date [: format [: timezone [: locale]]] }}`. The 'Input value' section explains that the date expression can be a Date object, a number (milliseconds since UTC epoch), or an ISO string. A right-hand sidebar lists various sections like Input value, Parameters, See also, Description, Usage notes, Pre-defined format options, Custom format options, Format examples, and Usage example.

Angular - DatePipe

Seguro | <https://angular.io/api/common/DatePipe>

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG

GETTING STARTED

TUTORIAL >

FUNDAMENTALS >

TECHNIQUES >

API

stable (v6.0.7)

API / @angular/common

DatePipe

PIPE STABLE

Formats a date value according to locale rules.

```
{{ value_expression | date [ : format [ : timezone [ : locale ] ] ] }}
```

Input value

The date expression: a `Date` object, a number (milliseconds since UTC epoch), or an ISO string (<https://www.w3.org/TR/NOTE-datetime>).

value any

DatePipe

- Input value
- Parameters
- See also
- Description
- Usage notes
- Pre-defined format options
- Custom format options
- Format examples
- Usage example



Custom Pipes

É possível implementar nossos próprios Pipes

- Decorar a classe com “Pipe”
- Implementar a interface PipeTransforms
- Adicionar novo pipe no app.module em declarations

```
TS courses.component.ts  TS summary.pipe.ts x  TS app.module.ts
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'summary'
5  })
6  export class SummaryPipe implements PipeTransform {
7    transform(value: string, limit?: number) {
8      if (!value)
9        return null;
10
11      let actualLimit = (limit) ? limit : 50;
12      return value.substr(0, actualLimit) + '...';
13    }
14  }
```



Angular

Diretivas



Diretivas

O que é?

- São uma forma de passar instruções para o template, permitindo mais flexibilidade na construção do HTML

Existem dois tipos de diretivas

- Diretivas estruturais – utilizadas para modificar a estrutura do DOM / HTML
- Diretivas de atributos – interagem com elementos que foram aplicadas (classe ou estilo por exemplo)



Diretivas estruturais - ngIf

O que é?

- Controla a exibição de um item de um através de uma expressão booleana

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-diretiva-ngif',
  templateUrl: './diretiva-ngif.component.html',
  styleUrls: ['./diretiva-ngif.component.css']
})
export class DiretivaNgifComponent implements OnInit {

  cursos: string[] = ["Angular 2"];
```

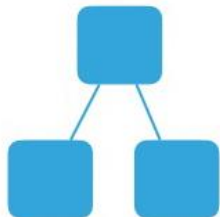
```
<div *ngIf="cursos.length > 0" >
  Lista de cursos aqui.
</div>
<div *ngIf="cursos.length == 0" >
  Não existem cursos para serem listados.
</div>
```



Diretivas estruturais - ngIf x atributo hidden

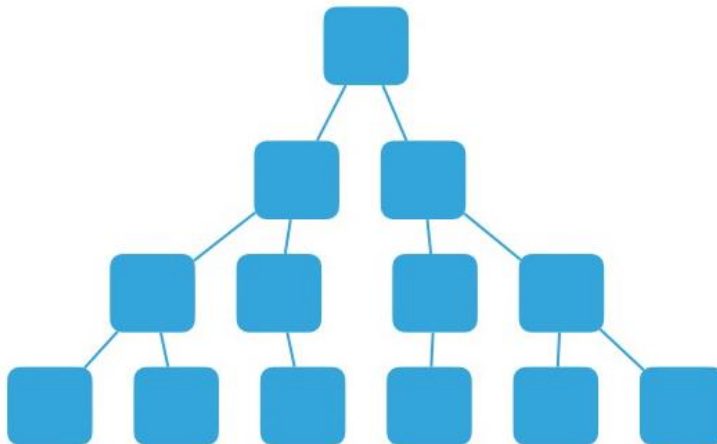
[hidden]

recomendado para árvore de
elementos pequenas



*ngIf

recomendado para árvore de
elementos grandes





Diretivas estruturais - ngSwitch, ngSwitchCase e ngSwitchDefault

O que é?

- Controla a exibição de um item de um através de uma expressão para mais de uma possibilidades

```
<div class="container" [ngSwitch]="aba" >
  <p *ngSwitchCase="'mapa'">Modo Mapa ativado</p>
  <p *ngSwitchCase="'lista'">Modo Lista ativado</p>
  <p *ngSwitchDefault>Home</p>
</div>
```



Diretivas estruturais - ngFor

O que é?

- Realiza a criação dos items a partir de uma iteração de itens de um array

```
})  
export class DiretivaNgforComponent implements OnInit {  
  {  
    cursos: string[] = ["Angular 2", "Java", "Phonegap"];  
  
    constructor() { }  
  
    ngOnInit() {  
    }  
  
  }  
}
```

```
<h5>Diretiva *ngFor</h5>  
  
<ul>  
  <li *ngFor="let curso of cursos, let i = index">  
    {{ i }} - {{ curso }}  
  </li>  
</ul>
```



Diretivas estruturais - Uso do *

O que é?

- Sintaxe simplificada para criação dos conteúdos

```
<template [ngIf]="mostrarCursos">
  <div>Lista de cursos aqui.</div>
</template>

<div template="ngIf mostrarCursos">
  Lista de cursos aqui.
</div>
```

```
<div class="container" [ngSwitch]="aba" >
  <template [ngSwitchCase]="''mapa''">
    <p>Modo Mapa ativado</p>
  </template>
  <template [ngSwitchCase]="''lista''">
    <p>Modo Lista ativado</p>
  </template>
  <template [ngSwitchCase]="''home''" ngSwitchDefault>
    <p>Home</p>
  </template>
```



Diretivas atributos - ngClass

O que é?

- Aplica classes no elemento conforme expressão booleana

```
<h1>
  <i class="glyphicon"
    [class.glyphicon-star-empty]="!meuFavorito"
    [class.glyphicon-star]="meuFavorito"
    (click)="onClick()"
  ></i>
</h1>
```

```
<h1>
  <i class="glyphicon"
    [ngClass]="{
      'glyphicon-star-empty': !meuFavorito,
      'glyphicon-star': meuFavorito
    }"
    (click)="onClick()"
  ></i>
</h1>
```



Diretivas customizadas

Forma para criarmos diretivas próprias

- Elementos que possuem esta diretiva irão aplicar mudanças definidas nesta classe
- ng generate directive nome-da-diretiva
- ng g d nome-da-diretiva

```
import { Directive, HostListener, ElementRef } from '@angular/core';

@Directive({
  selector: '[appInputFormat]'
})
export class InputFormatDirective {
  constructor(private el: ElementRef) { }

  @HostListener('blur') onBlur() {
    let value: string = this.el.nativeElement.value;
    this.el.nativeElement.value = value.toLowerCase();
  }
}
```

```
<input type="text" appInputFormat>
```



Diretivas customizadas com property binding

Possibilita a modificação da diretiva de acordo com dados passados pelo property binding

```
export class InputFormatDirective {  
  @Input('appInputFormat') format;  
  
  constructor(private el: ElementRef) { }  
  
  @HostListener('blur') onBlur() {  
    let value: string = this.el.nativeElement.value;  
  
    if (this.format == 'lowercase')  
      this.el.nativeElement.value = value.toLowerCase();  
    else  
      this.el.nativeElement.value = value.toUpperCase();  
  }  
}
```




Angular

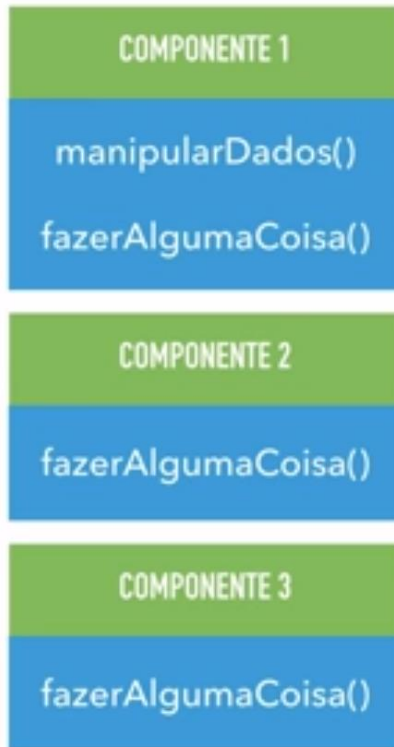
Serviços e Injeção de Dependência



Serviços

O que é? Por que fazemos?

- Classes para isolar a regra de negócio em um lugar específico;
- Reaproveitamento do mesmo serviço para vários componentes;
- Testes

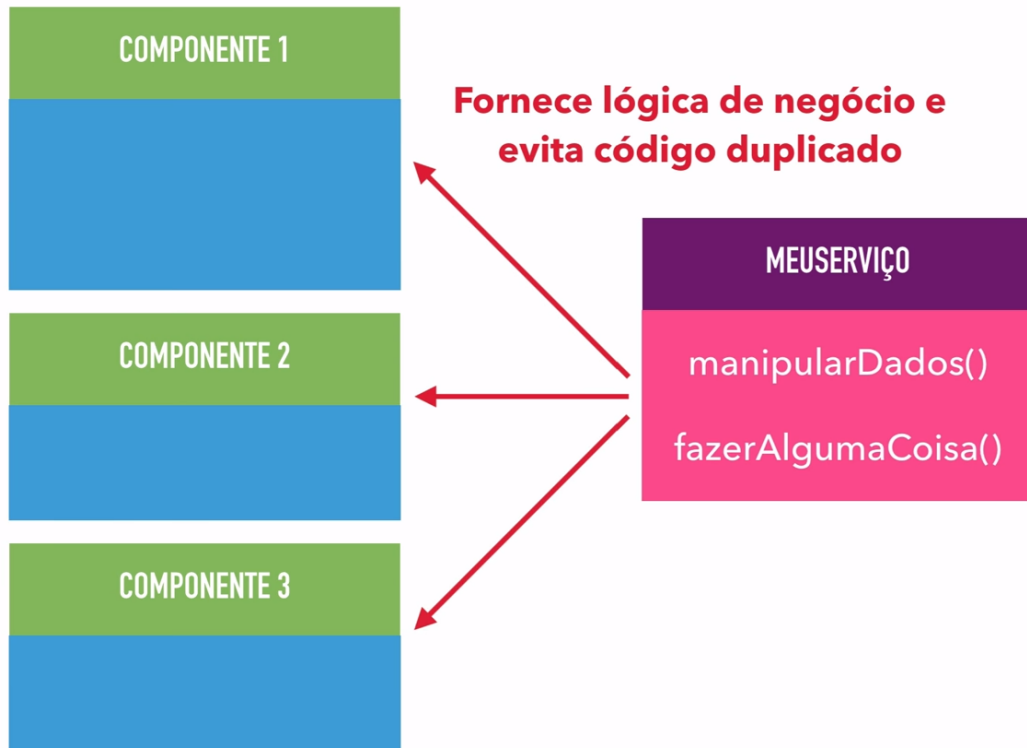




Serviços

O que é? Por que fazemos?

- Classes para isolar a regra de negócio em um lugar específico;
- Reaproveitamento do mesmo serviço para vários componentes;
- Testes
- Componente deve ser responsável somente por pegar os dados e possuir a interação tela x usuário





Serviço

Como criar e usar um novo service

- novo arquivo curso.service.ts

```
export class CursosService {  
  |  
  getCursos() {  
    return ['Angular 2', 'Java', 'Phonegap'];  
  }  
}
```

```
templateUrl: './cursos.component.html',  
styleUrls: ['./cursos.component.css']  
}))  
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  cursosService: CursosService;  
  
  |  
  constructor() {  
    this.cursosService = new CursosService();  
  }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



Serviço

Como gerar com angular CLI

- `ng generate service nome-service`
- `ng g s nome-service`

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\Projetos\Angular\hello-world [master +2 ~2 -0 !]> ng g s services/student
CREATE src/app/services/student.service.spec.ts (380 bytes)
CREATE src/app/services/student.service.ts (136 bytes)
D:\Projetos\Angular\hello-world [master +3 ~2 -0 !]> |
```

TS student.service.ts x

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable()
4  export class StudentService {
5
6      constructor() { }
7
8      getStudents() {
9          return ['Philippe Coutinho', 'Willian', 'Neymar', 'Gabriel Jesus'];
10     }
11 }
12
```



Injeção de Dependência

O que é?

- Injetar/Servir dependências de uma classe no seu construtor
- Angular possui um framework de injeção de dependência

Como sinalizar

- Incluir no modulo o serviço como provider



Injeção de Dependência

O que é?

- Injetar/Servir dependências de uma classe no seu construtor
- Angular possui um framework de injeção de dependência

Como sinalizar

- Incluir no modulo o serviço como provider

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  //cursosService: CursosService;  
  
  constructor(private cursosService: CursosService) {  
    //this.cursosService = new CursosService();  
    //this.cursosService = _cursosService;  
  }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



Injeção de Dependência

Singleton

- Declarar serviço no provider do module

Uma instância para cada componente

- Declarar serviço no provider do componente

```
import { CursosService } from './cursos.service';

@Component({
  selector: 'app-cursos',
  templateUrl: './cursos.component.html',
  styleUrls: ['./cursos.component.css'],
  providers: [CursosService]
})
export class CursosComponent implements OnInit {

  cursos: string[] = [];
  //cursosService: CursosService;

  constructor(private cursosService: CursosService) {
```




Serviços

Comunicação entre componentes com serviço

- Sinalizar através de um EventEmitter alguma mudança
- Este evento deve ser subscrito nas outras classe para ser tratado

```
export class CursosService {  
  
  emitirCursoCriado = new EventEmitter<string>();  
  
  private cursos: string[] = ['Angular 2', 'Java', 'Phonegap'];  
  
  constructor(){  
    console.log('CursosService');  
  }  
  
  getCursos() {  
    return this.cursos;  
  }  
  
  addCurso(curso: string){  
    this.cursos.push(curso);  
    this.emitirCursoCriado.emit(curso);  
  }  
}
```

```
this.cursosService.emitirCursoCriado.subscribe(  
  curso => console.log(curso)  
);  
}
```



Injeção de Dependência

TS student.service.ts

TS courses.component.ts x

```
1 import { StudentService } from '../services/student.service';
2 import { Component } from '@angular/core';
3
4 @Component({
5   selector: 'app-courses',
6   template: `
7     <h2>{{ title }}</h2>
8     <ul>
9       <li *ngFor='let curso of cursos; let i = index'>
10         {{ curso }}<span *ngIf='i % 2 === 0'> - Matriculado!</span>
11       </li>
12     </ul>`
13 })
14 export class CoursesComponent {
15   title = 'Lista de cursos';
16   cursos = ['.net', 'asp.net', 'angular'];
17   alunos: string[] = [];
18
19   constructor(private service: StudentService) {
20     this.alunos = service.getStudents();
21   }
22 }
23
```



Angular

Rotas



Rotas

SPA – Single Page Application

- Páginas são alteradas sem ser recarregadas
- Urls / rotas são alteradas e somente um conteúdo é recarregado

Como funciona Roteamento

- Através da rota o angular carrega um componente



Rotas

SPA – Single Page Application

- Páginas são alteradas sem ser recarregadas
- Urls / rotas são alteradas e somente um conteúdo é recarregado

Como funciona Roteamento

- Através da rota o angular carrega um componente

<http://meuprojeto.com.br/usuarios>



ListaUsuariosComponent



Rotas

<http://meuprojeto.com.br/usuarios/2/edit>



Rotas

<http://meuprojeto.com.br/usuarios/2/edit>

/usuarios

ListaUsuariosComponent

/usuarios/:id

UsuarioDetalhesComponent

/usuarios/:id/edit

UsuarioFormComponent



Rotas - Rotas Simples

Configurando

- Incluir `RouterModule.forRoot()` no `app.module`
- Definir array de rotas pelo objeto `{ path: string, component: MeuComponent }`
- Definir uma rota genérica com pattern `'**'`

```
RouterModule.forRoot([  
  { path: '', component: HomeComponent },  
  { path: 'followers', component: GithubFollowersCon  
  { path: 'profile/:username', component: GithubFol  
])
```




Rotas - Rotas Simples

Definir o router-outlet

- Lugar aonde será renderizado o componente associado a rota durante a navegação

```
1  <navbar></navbar>  
2  <router-outlet></router-outlet>
```



Rotas - RouterLink

Diretiva para indicar rotas no angular

- Evita recarregar os bundles javascripts novamente (conceito do SPA)

RouterLink dinâmicos com parâmetros

- Utilizar o property binding passando um array com os parâmetros

```
<a routerLink="/followers">  
<a [routerLink]="['followers', follower.id]">
```



Rotas - RouterLinkActive

Diretiva para indicar rota ativa e aplicar css

- Valida a rota ativa e caso positivo aplica css definido

```
<ul class="nav navbar-nav">  
  <li routerLinkActive="active current"><a routerL  
  <li routerLinkActive="active current"><a routerL  
</ul>  
</div>
```



Rotas - ActivatedRoute

O que é?

- Serviço utilizado para obtermos dados da rota

Direta



Assíncrona

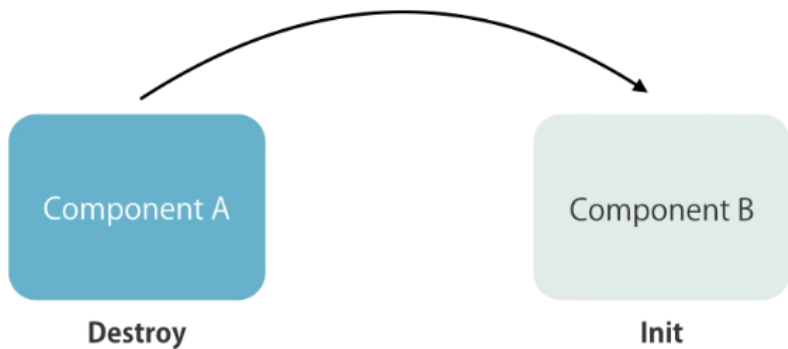


```
constructor(private route: ActivatedRoute) { }  
  
ngOnInit() {  
  this.route.snapshot.paramMap.get('id');  
  
  this.route.paramMap.subscribe(params => {  
    console.log(params.get('id'));  
  });  
}
```



Rotas - ActivatedRoute

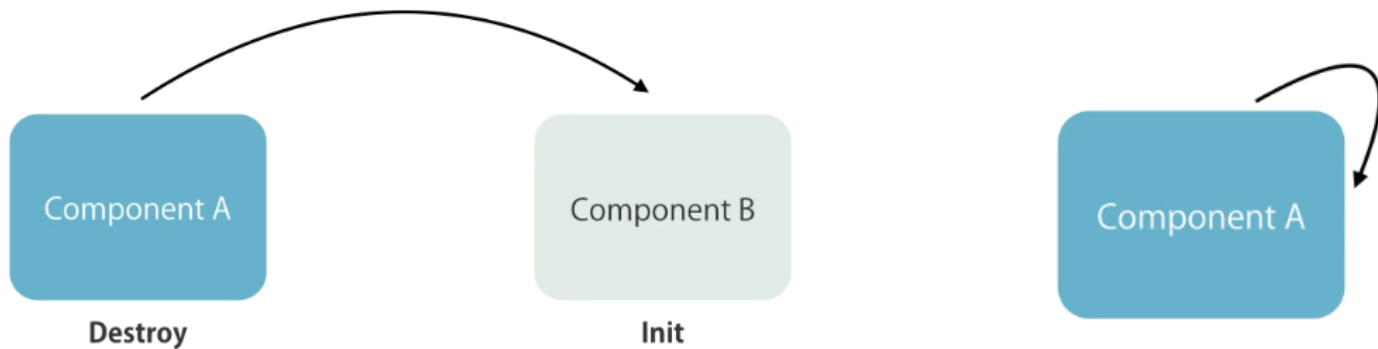
Por que assíncrona?





Rotas - ActivatedRoute

Por que assíncrona?





Rotas - Query parameters

O que é?

- Dados adicionais enviados pela rota

```
<a routerLink="/followers"  
  [queryParams]="{ page: 1, order: 'newest' }"
```

Como obter?

```
this.route.queryParamMap.subscribe();  
this.route.snapshot.queryParamMap.get('page')
```



Rotas - Redirecionamento no código

O que é?

- Navegar para uma rota diretamente no código

Como?

- Através do router utilizar o método navigate

```
constructor(private router: Router) { }

submit() {
  this.router.navigate(['/followers'], {
    queryParams: { page: 1, order: 'newest' }
  });
}
```




Rotas - Guarda de rotas

O que é?

- Serviço para validar se a rota é acessível de acordo com alguma condição
- Implementa a interface CanActivate que possui o método canActivate indicando a permissão
- Para funcionar precisa indicar na rota qual classe é o Guarda de rotas

```
export class AuthGuard implements CanActivate {  
  
  constructor() { }  
  
  canActivate(  
    route: ActivatedRouteSnapshot,  
    state: RouterStateSnapshot  
  ) : Observable<boolean> | boolean {  
  
    return true;  
  }  
}
```

```
{ path: '', component: HomeComponent,  
  canActivate: [AuthGuard]  
}
```



Angular

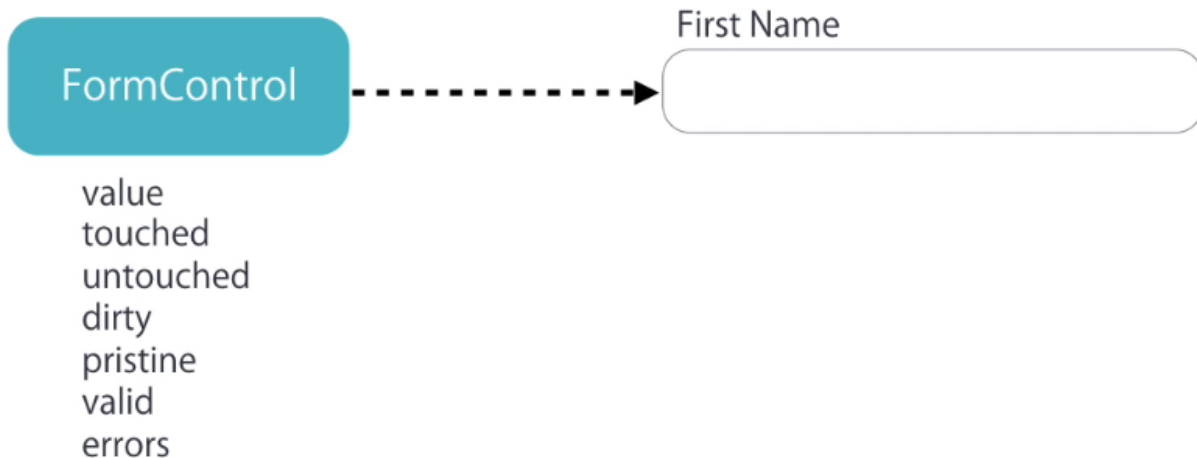
Formulários



Formulários

FormControl

- Cada campo input do formulário é necessário ter uma instância de controle
- Com esse tipo de classe do angular nós conseguimos verificar diversas propriedades

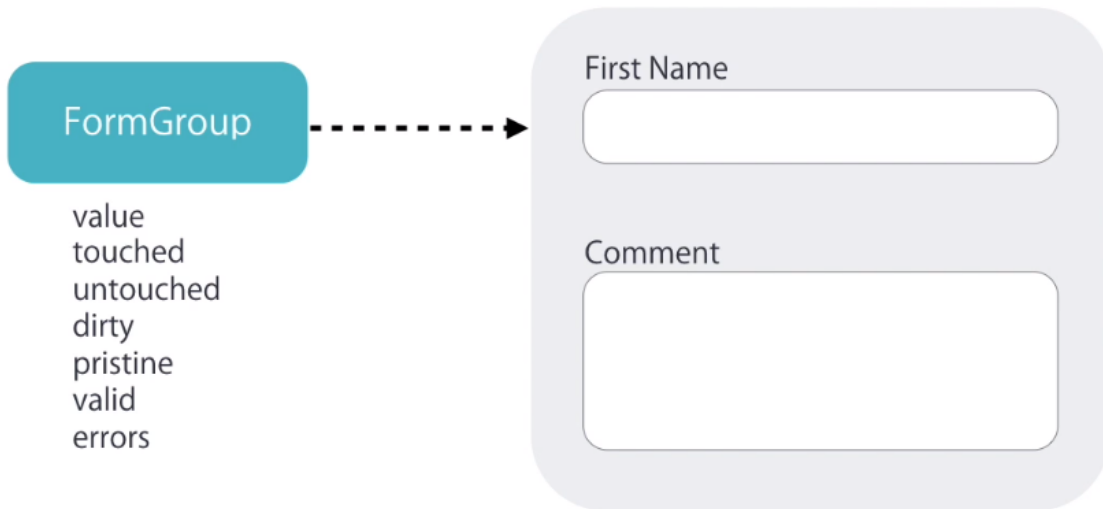




Formulários

FormGroup

- Possui um grupo de FormControl
- As propriedades são referentes campos do seu grupo





Formulários - Tipos

Aplicando diretivas diretamente no formulário (Template-driven forms)

- Formulários simples;
- Simples validações;
- Fácil de criar;
- Menos código

Criando esses controles no código (Reactive Forms)

- Validações e lógica mais complexas
- Formulários complexos
- Testes unitários



Angular

Formulário Template-driven



Formulários - Template-driven

ngModel

- Utilizado para indicar que o campo será um FormControl
- Precisa definir um name para diferenciar dos outros campos

Variavel local

- É possível utilizar uma variável local para manipular o formControl

```
<div class="form-group">
  <label for="firstName">First Name</label>
  <input ngModel name="firstName" #firstName="ngModel"
</div>
<div class="form-group">
```



Formulários - Template-driven

Validações

- São feitas com validações html 5 (required, maxlength, minlength, pattern)
- Através da variável local conseguimos ter acesso aos valores do FormControl

```
<input required minlength="3" maxlength="10" pattern  
<div class="alert alert-danger" *ngIf="firstName.tou  
  <div *ngIf="firstName.errors.required">First name  
  <div *ngIf="firstName.errors.minlength">First name  
  <div *ngIf="firstName.errors.pattern">First name d  
</div>
```




Formulários - Template-driven

Validações com css

- Angular fornece algumas classes em cima do estado atual com controle

```
.form-control.ng-touched.ng-invalid {  
  border: 2px solid red;  
}
```



Formulários - Template-driven

ngForm

- Diretiva aplicada ao Form como sendo o grupo dos FormControls

ngSubmit

- Propriedade Output do ngForm para ser usado ao enviar o formulário

```
<form #f="ngForm" (ngSubmit)="submit(f)">
  <div class="form-group">
    <label for="firstName">First Name</label>
```



Formulários - Template-driven

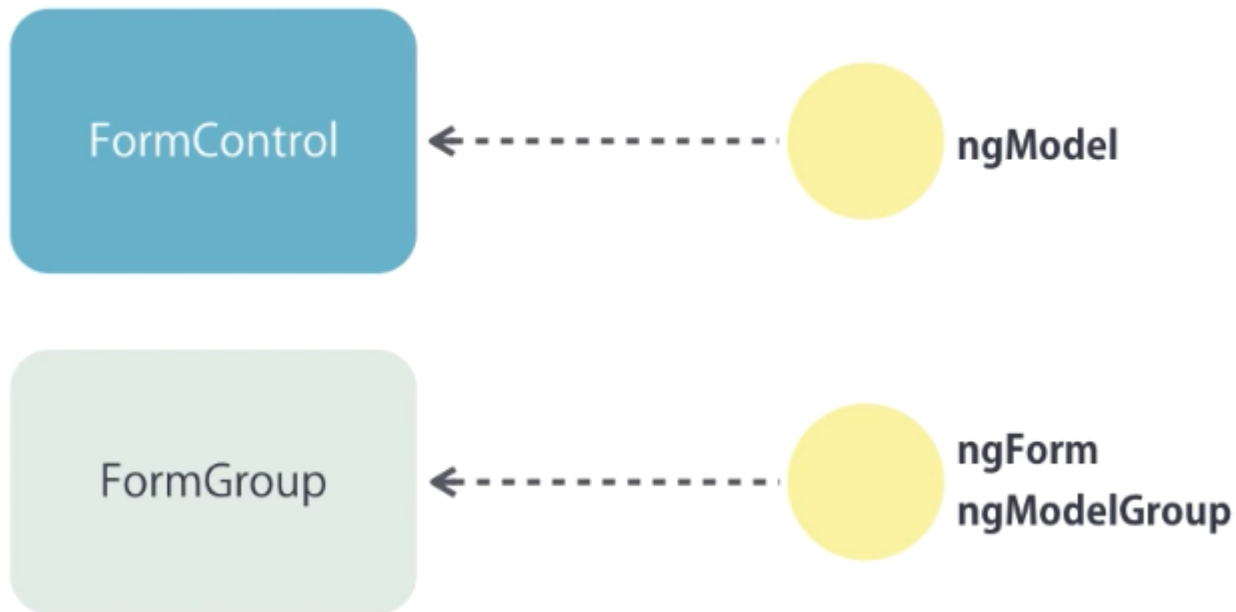
ngModelGroup

- Diretiva para indicar um FormGroup
- Valor do formulário corresponde a organização do FormGroup e FormControl

```
<form #f="ngForm" (ngSubmit)="submit(f)">
  <div ngModelGroup="contact" #contact="ngModelGroup">
    <div class="form-group">
      <label for="firstName">First Name</label>
      <input
        required
        minlength="3"
        maxlength="10"
        pattern="banana"
        ngModel
        name="firstName">
```



Formulários - Template-driven





Formulários - Template-driven

Checkbox

```
<div class="checkbox">
  <label>
    <input type="checkbox" ngModel name="isSubscribed">
  </label>
</div>
```



Formulários - Template-driven

Dropdown

```
<div class="form-group">
  <label for="contactMethod">Contact Method</label>
  <select ngModel name="contactMethod" id="contactMeth
    <option value=""></option>
    <option value=""></option>
    <option value=""></option>
    <option value=""></option>
  </select>
</div>
```



Formulários - Template-driven

Radio buttons

```
<div class="radio">
  <label>
    <input ngModel type="radio" name="contactMethod" value="Email" />
    Email
  </label>
</div>
<div class="radio">
  <label>
    <input ngModel type="radio" name="contactMethod" value="Phone" />
    Phone
  </label>
</div>
```



Angular

Formulário Reactive



Formulários - Reactive

- Precisa importar no app.module o ReactiveFormsModule
- Controles tem que ser declarados no código;
- Classes FormGroup e FormControl tem que ser importada no Component

```
export class SignupFormComponent {  
  form = new FormGroup({  
    username: new FormControl(),  
    password: new FormControl()  
  });  
}
```

```
<form [formGroup]="form">  
  <div class="form-group">  
    <label for="username">Username</label>  
    <input  
      formControlName="username"  
      id="username"  
      type="text"  
      class="form-control">  
  </div>  
  <div class="form-group">
```



Formulários - Reactive

Validações

- Não são feitas com validações HTML 5
- São feito utilizando Validators (do pacote @angular/forms)
- São passados diretamente ao definir os controles do formulário (1 ou array)
- Acesso as propriedade são feitas pela variavel do formGroup

```
export class SignupFormComponent {  
  form = new FormGroup({  
    username: new FormControl('', Validators.required),  
    password: new FormControl('', Validators.required)  
  });  
}
```

```
<input  
  formControlName="username"  
  id="username"  
  type="text"  
  class="form-control">  
<div *ngIf="form.get('username').touched" >  
</div>  
div class="form-group">
```



Formulários - Reactive

Validações

- Não são feitas com validações HTML 5
- São feito utilizando Validators (do pacote @angular/forms)
- São passados diretamente ao definir os controles do formulário (1 ou array)
- Acesso as propriedade são feitas pela variavel do formGroup

```
<input
  FormControlName="username"
  id="username"
  type="text"
  class="form-control">
<div *ngIf="form.get('username').touched" c
/div>
div class="form-group">
```

```
get username() {
  return this.form.get('username');
}
```



Formulários - Reactive

Validações customizadas

- Retorna um `validationError` ou `null` (se ok)
- A função que irá validar geralmente é implementada `static`

```
export class UsernameValidators {  
  static cannotContainSpace(control: AbstractControl) {  
    if ((control.value as string).indexOf(' ') >= 0) {  
      return { cannotContainSpace: true };  
    }  
    return null;  
  }  
}
```



Formulários - Reactive

Validações customizadas assíncronas

- Validações que envolvem chamadas no servidor (API)
- Retorna um `Promise<ValidationErrors>` ou `Promise<null>` (se ok)
- A função que irá validar geralmente é implementada static
- Validações assíncronas são indicadas no `FormControl` no terceiro parametro do construtor

```
static shouldBeUnique(control: AbstractControl) : Promise<ValidationErrors> {  
    return new Promise((resolve, reject) => {  
        setTimeout(() => {  
            if (control.value === 'mosh')  
                resolve({ shouldBeUnique: true });  
            else  
                resolve(null);  
        }, 2000);  
    });  
}
```

```
form = new FormGroup({  
    username: new FormControl('',  
        Validators.required,  
        UsernameValidators.shouldBeUnique),  
    password: new FormControl('', Validators.required)  
});
```



Formulários - Reactive

Validações customizadas assíncronas

- Para exibir que uma validação assíncrona está sendo executada podemos utilizar a propriedade `pending`

```
class="form-control">  
<div *ngIf="username.pending">Checking for unique  
<div *ngIf="username.touched && username.invalid"
```



Formulários - Reactive

Validações ao realizar o submit

- É possível realizar validações no método que realiza o submit e incluir erros no nível do form

```
login() {  
  let isValid = authService.login(this.form.value);  
  if (!isValid) {  
    this.form.setErrors({  
      invalidLogin: true  
    })  
  }  
}
```

```
<form [formGroup]="form" (ngSubmit)="login()">  
  <div *ngIf="form.errors" class="alert alert-danger">  
    The username or password is invalid.  
  </div>
```



Formulários - Reactive

FormGroups

- Para trabalhar com grupo é necessário incluir sua declaração no TS e ajustar o template com a propriedade formGroupName="nome"

```
export class SignupFormComponent {  
  form = new FormGroup({  
    account: new FormGroup({  
      username: new FormControl(''),  
      password: new FormControl('')  
    })  
  });  
  
  get username() {  
    return this.form.get('account.username');  
  }  
}
```

```
<form [formGroup]="form">  
  <div formGroupName="account">  
    <div class="form-group">  
      <label for="username">Username</label>  
      <input  
        formControlName="username"  
        id="username"  
      >  
    </div>  
  </div>  
</form>
```




Formulários - Reactive

FormBuilder

- Outra maneira para declarar formulários menos repetitiva

```
form = new FormGroup({  
  name: new FormControl('', Validators.required),  
  contact: new FormGroup({  
    email: new FormControl(),  
    phone: new FormControl()  
  }),  
});
```

```
constructor(fb: FormBuilder){  
  fb.group({  
    name: ['', Validators.required],  
    contact: fb.group({  
      email: [],  
      phone: []  
    })  
  })  
}
```



Angular

Cosumindo HTTP Services



Consumindo HTTP Services

- Importar o HttpClientModule (@angular/http) no app.module e adicionar no imports
- Importar a classe HttpClient (@angular/http) na classe que vai ser utilizada (Service)
- Iniciar por Injeção de Dependência um objeto do tipo Http

```
constructor(http: Http) {  
  http.get('http://jsonplaceholder.typicode.com/posts')  
}
```



Consumindo HTTP Services

- Não deve ser acionado no construtor (que deve ser simples e leve)
- Retorno da chamada é um `Observable<Response>` (para trabalhar de forma assíncrona com chamadas para o servidor)
- Para trabalhar com `Observable` devemos utilizar o `subscribe`, que é acionado quando temos a resposta da execução do método anterior

```
ngOnInit() {  
  this.http.get(this.url)  
    .subscribe(response => {  
    this.posts = response.json();  
  });  
}
```



Consumindo HTTP Services

HTTP Request

- Utilizamos os verbos/métodos do protocolo/classe HTTP para realizar as requisições na API

Verb	Description
GET	Get data
POST	Create data
PUT	Update data
DELETE	Delete data



Consumindo HTTP Services

GET – argumentos: url

```
http.get('http://jsonplaceholder.typicode.com/posts')  
  .subscribe(response => {  
    this.posts = response.json();  
  });
```

POST – argumentos: url e o objeto a ser adicionado

```
this.http.post(this.url, JSON.stringify(post))  
  .subscribe(response => {  
    console.log(response.json());  
  });
```



Consumindo HTTP Services

PUT/PATCH – argumentos: url com id e o objeto a ser atualizado

```
updatePost(post) {  
  this.http.patch(this.url + '/' + post.id, JSON.stringify(post))  
    .subscribe(response => {  
      console.log(response.json());  
    })  
}
```

DELETE – argumentos: url com id

```
deletePost(post) {  
  this.http.delete(this.url + '/' + post.id)  
    .subscribe(response => {  
      console.log(response.json());  
    })  
};
```



Consumindo HTTP Services

- Estas chamadas ao backend devem ser encapsuladas em Services
- Tratamento de erros devem ficar encapsulados em Services

Unexpected

- Server is offline
- Network is down
- Unhandled exceptions

Expected

- “Not Found” errors (404)
- “Bad request” errors (400)



Consumindo HTTP Services

Tratamento de Erros

```
deletePost(id) {  
  return this.http.delete(this.url + '/' + id)  
    .catch((error: Response) => {  
      if (error.status === 404)  
        return Observable.throw(new NotFoundError());  
      return Observable.throw(new AppError(error));  
    });  
}
```

```
deletePost(post) {  
  this.service.deletePost(345)  
    .subscribe(  
      response => {  
        let index = this.posts.indexOf(post);  
        this.posts.splice(index, 1);  
      },  
      (error: AppError) => {  
        if (error instanceof NotFoundError)  
          alert('This post has already been deleted.');
```



Angular

Authentication e Authorization



Authentication

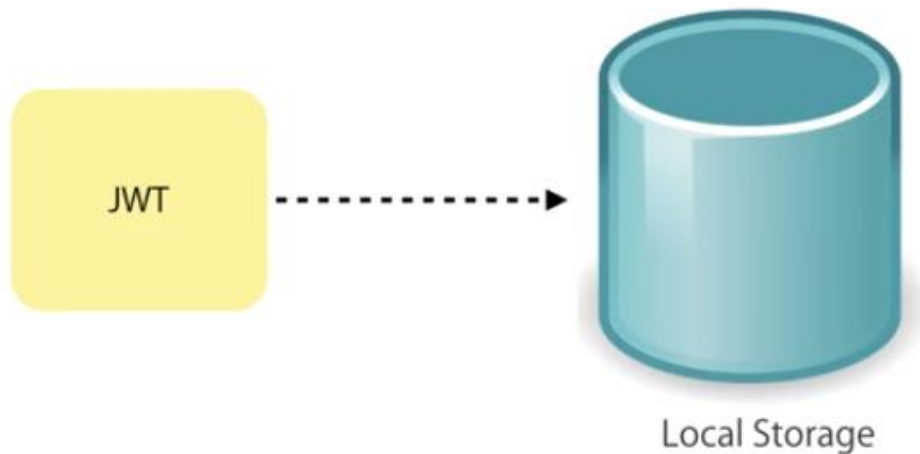
- Identificar o usuario no client e no servidor
- Através de usuario e senha recebemos um JWT que contem informações sobre o usuario e suas permimssões





Authentication

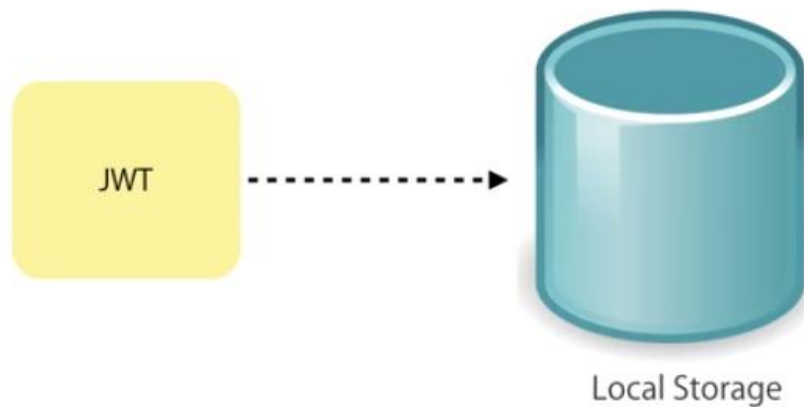
- Uma vez recebido esse token possui uma validade e é armazenado no browser pelo Local Storage (funcionalidade disponível na maioria dos browsers)





Authentication

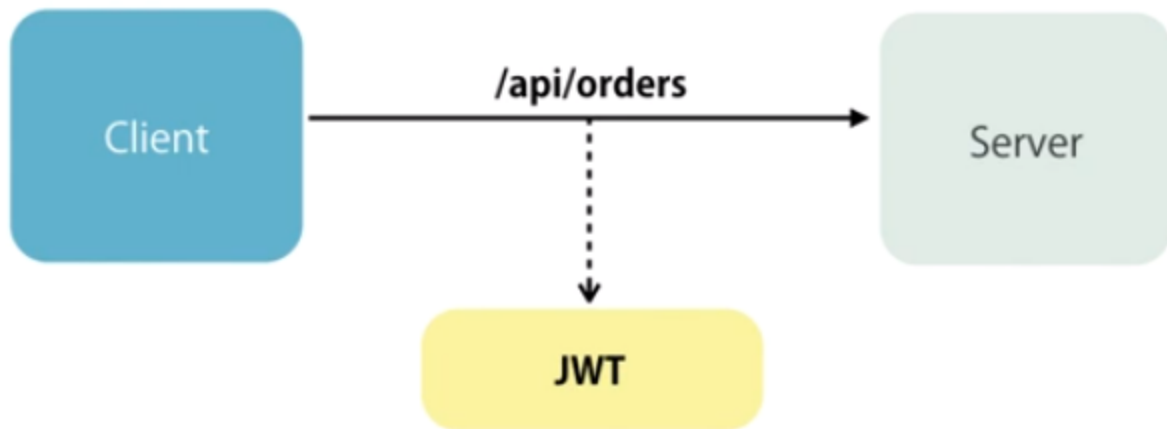
- Uma vez recebido esse token possui uma validade e é armazenado no browser pelo Local Storage (funcionalidade disponível na maioria dos browsers)
- Através do JWT podemos:
 - Obter dados do usuário
 - Exibir/esconder partes da página
 - Tratar acesso a determinadas rotas





Authorization

- Enviando esse token para o servidor conseguimos extrair os dados e validar se o usuario tem autorização para acessar determinado endpoint da API





- Encoded

eyJhbGciOiJIUzI1NiIsInR5cCI6I
kpXVCJ9.eyJzdWIiOiIxMjM0NTY3O
DkwIiwibmFtZSI6IkpvaG4gRGR9Ii
wiaWF0IjoxNTE2MjM5MDIyfQ.Sf1K
xwRJSMeKkF2QT4fwpMeJf36P0k6yJ
V_adQssw5c

Decoded

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    your-256-bit-secret
) ☐ secret base64 encoded
```



Authentication - Login

- Envio de usuario e senha e obtemos um jwt no corpo do retorno HTTP (200 status)

TS auth.service.ts

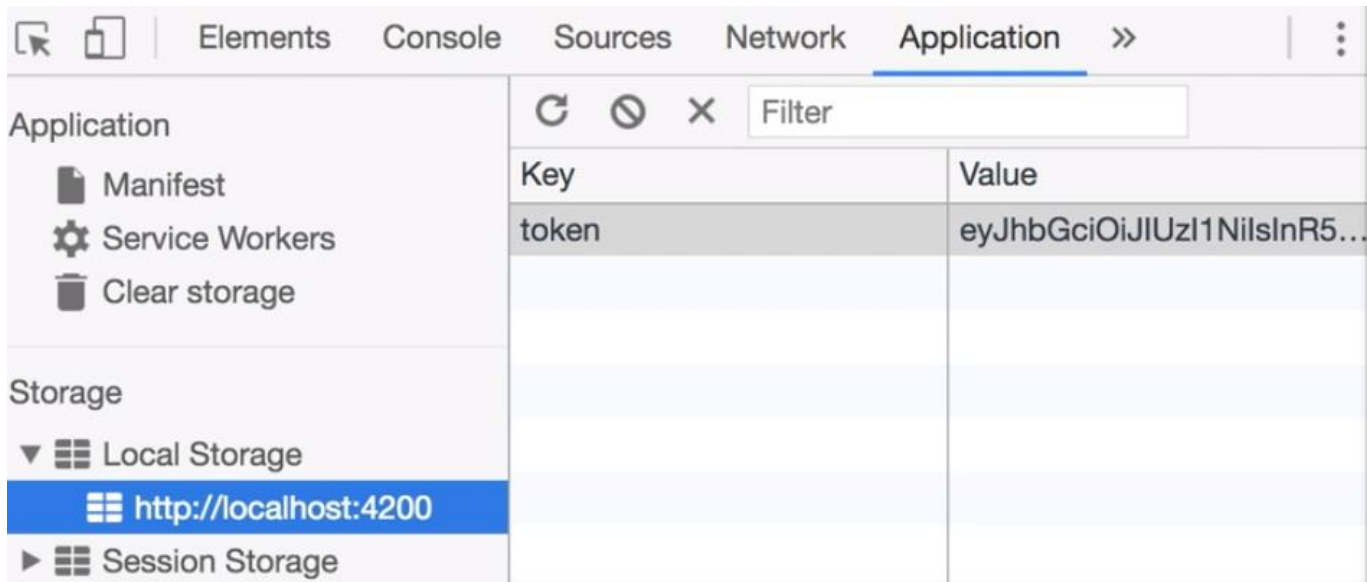
```
16
17   signIn(credentials) {
18     this.authService.login(credentials)
19       .subscribe(result => {
20       if (result)
21         this.router.navigate(['/']);
22       else
23         this.invalidLogin = true;
24     });
25   }
26 }
```

TS login.component.ts x

```
login(credentials) {
  return this.http.post('/api/authenticate',
    JSON.stringify(credentials))
    .map(response => {
      let result = response.json();
      if (result && result.token) {
        localStorage.setItem('token', result.token);
        return true;
      }
      return false;
    });
}
```




Authentication - Login



The image shows the Chrome DevTools Application tab. The left sidebar has a tree view with 'Application' expanded, showing 'Manifest', 'Service Workers', and 'Clear storage'. Under 'Storage', 'Local Storage' is expanded, and 'http://localhost:4200' is selected. The main pane shows a table of storage items.

Key	Value
token	eyJhbGciOiJIUzI1NiIsInR5...



Authentication - Logout

- Basta remover o token do LocalStorage

```
logout() {  
  localStorage.removeItem('token');  
}
```



Authentication

- Através do token obtem se o mesmo ainda é valido
- Utilizamos uma biblioteca que auxilia a manipulação do JWT @auth0/angular-jwt

```
isLoggedIn() {  
  let jwtHelper = new JwtHelper();  
  let token = localStorage.getItem('token');  
  
  if (!token)  
    return false;  
  
  let expirationDate = jwtHelper.getTokenExpirationDate(token);  
  let isExpired = jwtHelper.isTokenExpired(token);  
  
  console.log("Expiration", expirationDate);  
  console.log("isExpired", isExpired);  
  
  return !isExpired;  
}
```



Authentication

- Através do token podemos obter dados enviados no token
- Utilizamos uma biblioteca que auxilia a manipulação do JWT

@auth0/angular-jwt

```
get currentUser() {  
  let token = localStorage.getItem('token');  
  if (!token) return null;  
  
  return new JwtHelper().decodeToken(token);  
}  
}
```

```
<h1>Home Page</h1>  
<p *ngIf="authService.isLoggedIn()">  
  Welcome {{ authService.currentUser.name }}  
</p>  
<ul>  
  <li *ngIf="authService.isLoggedIn() && authService.isAdmin()">  
    <a routerLink="/admin">Admin</a>  
  </li>  
</ul>
```



Authentication

- Usando o AuthGuard e o JWT, a cada mudança de página podemos validar se o usuario esta valido e redirecioná-lo para página de login caso necessário.
- Acesso a páginas fica protegido por usuários não logados

```
TS auth-guard.service.ts x TS auth-guard.service.spec.ts ●
/
8   constructor(
9     private router: Router,
10    private authService: AuthService) { }
11
12   canActivate() {
13     if (this.authService.isLoggedIn()) return true;
14
15     this.router.navigate(['/login']);
16     return false;
17   }
18
19
```



Authorization

- Para fazer o acesso a API devemos passar o JWT no header da request.
- Biblioteca JWT utilizada já possui uma classe AuthHttp que faz o processo de inserir o token no header da request

```
getOrders() {  
  let headers = new Headers();  
  let token = localStorage.getItem('token');  
  headers.append('Authorization', 'Bearer ' + token);  
  
  let options = new RequestOptions({ headers: headers });  
  
  return this.http.get('/api/orders', options)  
    .map(response => response.json());  
}
```

```
getOrders() {  
  return this.authHttp.get('/api/orders')  
    .map(response => response.json());  
}
```



Authentication

- Do lado do servidor validar o acesso a api pelo token fornecido no header do request

```
//  
// Fake implementation of /api/orders  
//  
if (connection.request.url.endsWith('/api/orders') &&  
    connection.request.method === RequestMethod.Get) {  
    if (connection.request.headers.get('Authorization') === 'Bearer ' +  
        connection.mockRespond(new Response(  
            new ResponseOptions({ status: 200, body: [1, 2, 3] })  
        ));  
    } else {  
        connection.mockRespond(new Response(  
            new ResponseOptions({ status: 401 })  
        ));  
    }  
}
```



Obrigado

GitHub



Código fonte do exemplo criado durante esta apresentação:

<https://github.com/idenardi/angular-overview>



@idenardi