# Angular

**Tecnologia de Internet**

Prof Vitor Almeida dos Santos

# Conteúdo

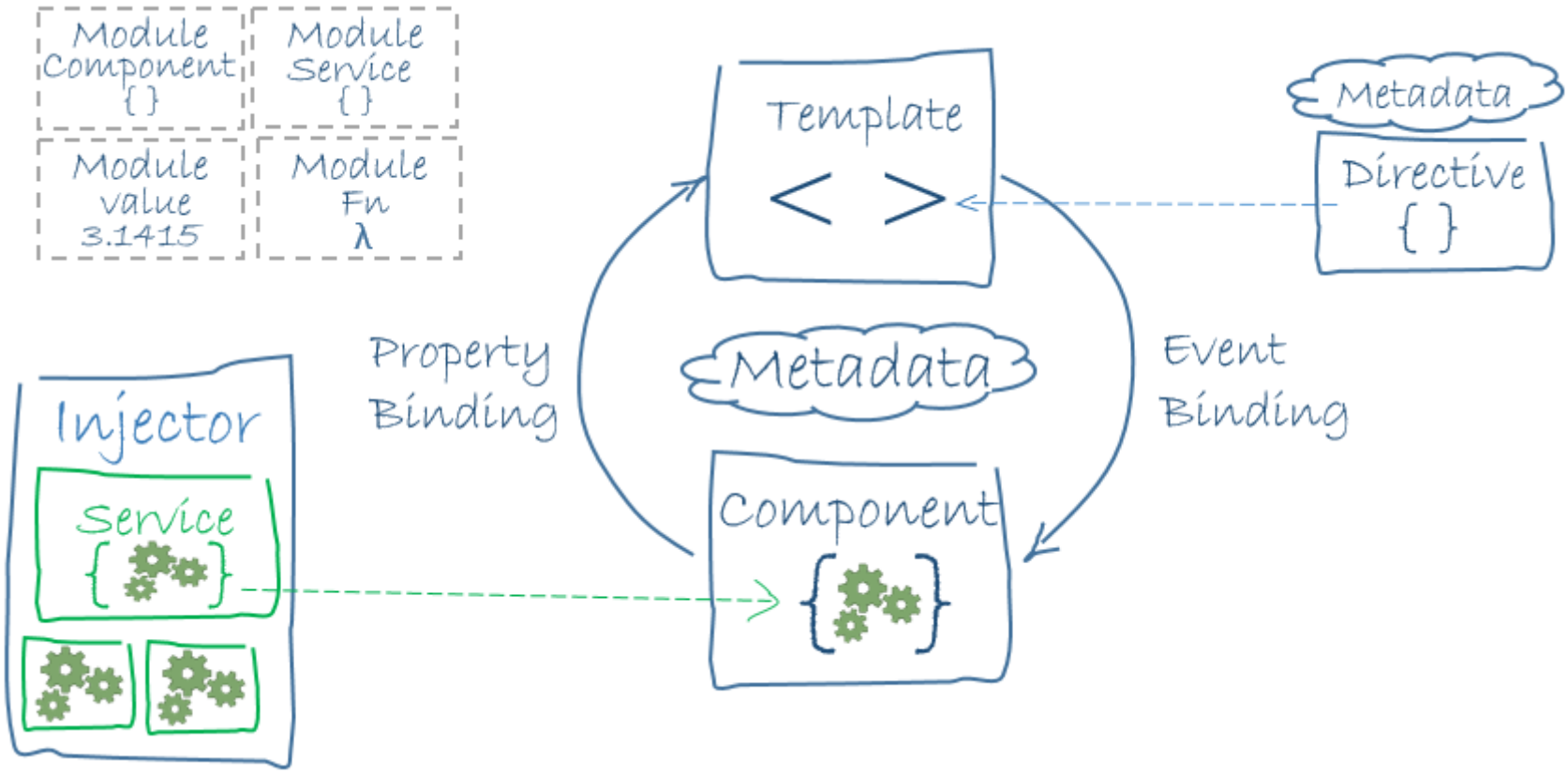
- Aspectos teóricos;
- Criando uma aplicação;
- Visão Geral de um Componente;
- Detalhando o Template de um Componente;
- Detalhando da Classe de um Componente e do uso de Modelo de Dados;
- Utilizando Rotas para Navegar entre Componentes;
- Utilizando Classes de Serviços para Acessar o Back-end (Web Services)

# **Angular**
# Aspectos Teóricos

# Introdução

- Histórico
  - Início em 2012;
  - Projeto de um funcionário da Google;
- Características
  - É um framework JavaScript;
  - Ideal para *Single Page Applications*: página web que ***não*** recarregam por inteiro com novas requisições.
  - Linguagem usada: Typescript;

# Arquitetura Angular



# Modules

- Cada app possui um *module* raiz, o qual contém:
  - *imports* de outros *modules;*
  - *declarations* com as classes de visão (view) que pertecem ao módulo;
  - *bootstrap* com o componente principal da aplicação;
  - *providers* com os serviços do módulo.

- Exemplo

```
@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ Componente1 ],
  bootstrap: [ Componente1 ]})
export class AppModule { }
```

# Components

- Cada *module* (app) contém um ou mais componentes;
- Cada componente “controla” uma parte da view (página);

- Exemplo

```
@Component({
      selector: 'componente1',
      template: '<div>Diga {{saudacao}}.
      <button (click)="digaAlo()">Dê um alô!</button></div>‘
})
export class Componente1 {
      saudacao = 'Alo, mundo!';
      digaAlo() {    console.log(this.saudacao)  }
}
```

# Templates

- Cada *template* é um código HTML que “renderiza” um componente;
- Além de código HTML, um template pode conter uma sintaxe própria: NgModel, NgFor, NgIf *etc*.

- Exemplo

```
@Component({
      selector: 'componente1',
      template: '<div>Diga {{saudacao}}.
      <button (click)="digaAlo()">Dê um alô!</button></div>`
})
```

# Data Binding

- Mecanismo para vincular dados do modelo (componente) com a *view* (página);

- Há alguns tipos:

  - *one-way*: somente exibe dados do modelo;

  - two-way: exibe dados do modelo e permite que estes sejam modificados pela view (página).

# Services

- Mecanismo que pode ser consumido por componentes;

- Ajudam a implementar conexão com serviços web.

# Criando uma Aplicação

# Configuração

- Instale node e npm;

- Se for necessário, inclua nas variáveis de ambiente:
  - PATH C:\Program Files\nodejs;C:\Users***usuario***\AppData\Roaming\npm

  * substitua ***usuario*** pela sua identificação de usuário

- Execute: npm install –g @angular/cli

- Execute: ng new minhaapp

- Inicie a aplicação: npm start

- Abra no navegador: http://localhost:4200

# Estrutura de Arquivos

- `angular-cli.json` — Configurações do Projeto
- `package.json` — Principais Dependências
- `src/`
  - `index.html` — Página Inicial
  - `styles.css` — Estilos CSS
  - `main.ts`
  - `app/`
    - `app.module.ts`
    - `app.component.ts` — Componente

# Módulo raiz: app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';

import { Componente1 }  from './app.component1';

@NgModule({
        imports:      [ BrowserModule ],

        declarations: [ AppComponent ],

        bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Recursos Usados

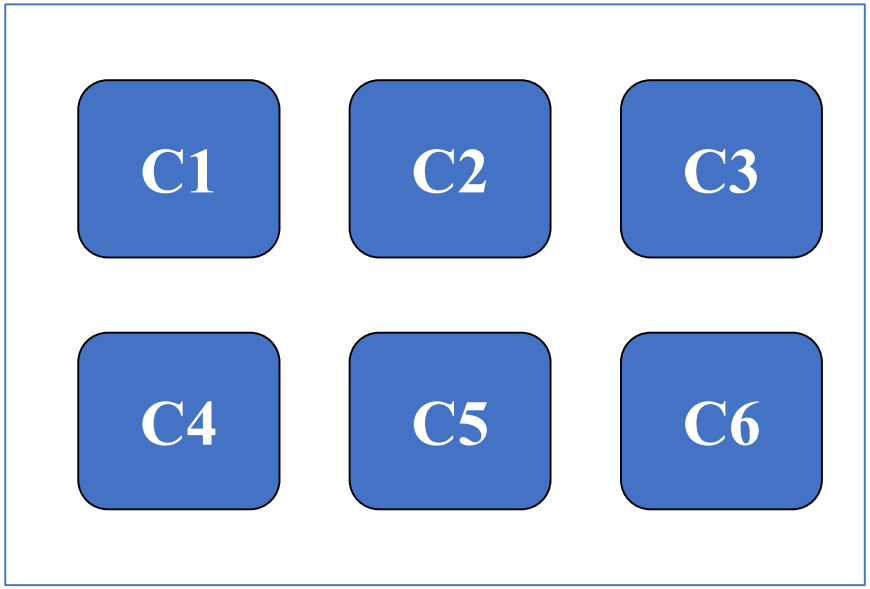
Componentes da Aplicação

Componente Inicial

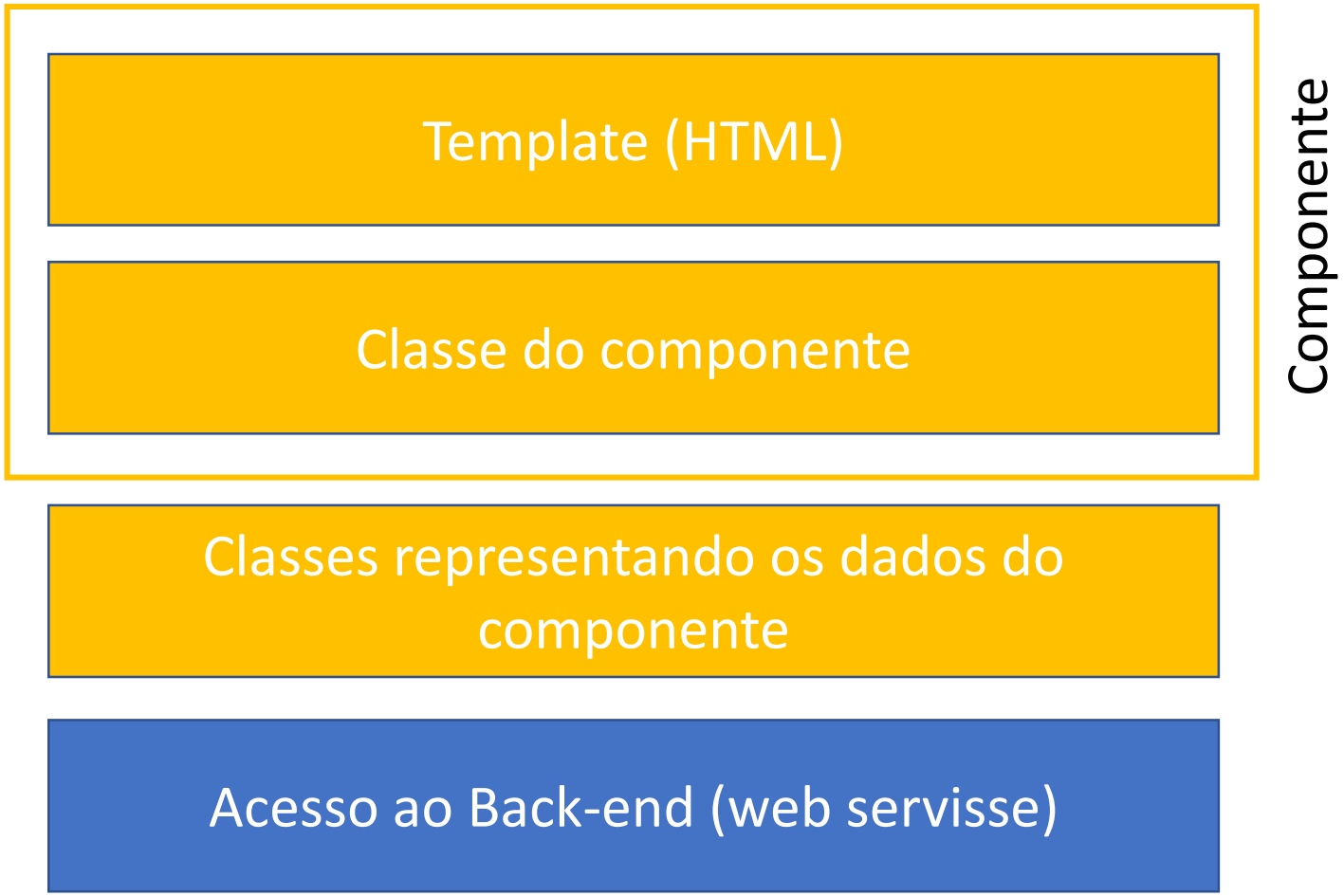
# Visão Geral de um Componente

# Componentes

- Um página pode conter diversos componentes;
- Componentes podem ser reutilizados;
- Componentes podem conter componentes.



# Como desenvolver cada componente?



# Template de um Componente – Visão Geral

- Código HTML que representa a parte visual. Utiliza dados da classe do componente. Exemplo:

```
<div>
  <h2>{{titulo}}</h2>
  <div>id: {{pessoa.Id}}</div>
  <div>nome: {{pessoa.Nome}}</div>
</div>
```

Neste exemplo, os valores de “título”, “pessoa.Id” e “pessoa.Nome” estão definidos na classe do componente.

# Classe de um Componente – Visão Geral

- Código Typescript com dados e métodos utilizados na parte visual. Exemplo:

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

@Component({
  selector: 'at-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'at';
}
```

Tag do componente

Código HTML do componente

CSS do componente

# Modelo de Dados um Componente – Visão Geral

- Classes que representam os tipos de dados manipulados em um componente;
- Podem ser definidos em arquivos separados e importados pelo componente.
- Exemplo:

```
export class Pessoa {
        Id: number;
        Nome: string;
}
```

# Usando Componentes

```
<body>
  <componente1>Carregando...</componente1>
</body>
```

# Detalhando o Template de um Componente

# Exibindo Dados

- One-way data binding: somente exibe dados existentes no modelo.
  - Exemplos

```
<div>{{saudacao}}</div>

<input value="{{nome}}" placeholder="nome">

<input [value]="nome" placeholder="nome">
```

# Listando Dados

- Utilize *ngFor para iterar sobre uma coleção.
  - Exemplo:

```
<ul>
   <li *ngFor="let x of [10,20,30,40]">
     <span>{{x}}</span>
   </li>
 </ul>
```

# Exibindo Condicionalmente

- Utilize *ngIf para exibir condicionalmente.
  - Exemplo:

```
<div *ngIf="nota < 7">Compareça à VS</div>
```

# Detalhando a Classe de um Componente e o uso de Modelo de Dados

# Utilizando Valores do Modelo

```
export class Pessoa {
        Id: number;
        Nome: string;
}
```

- Representa o modelo de dados;
- Pode ser colocado em um arquivo a parte.
- Este arquivo deve ser importado por outros que desejarem usar esse tipo.

# Utilizando Valores do Modelo

```
import { Component } from '@angular/core';
import { Pessoa }  from './pessoa.modelo';

@Component({
  selector: 'componente2',
  template: `
    <div>
      <h2>{{titulo}}</h2>
      <div><label>id: </label>{{pessoa.Id}}</div>
      <div><label>nome: </label>{{pessoa.Nome}}</div>
    </div>
`
})
export class Componente2 {
  titulo="Dados de pessoa";
  pessoa: Pessoa = {
    Id: 1,
    Nome: 'Fulano'
  };
}
```

# Listando Valores do Modelo

```
import { Component } from '@angular/core';
import { Pessoa }  from './pessoa.modelo';

@Component({
  selector: 'componente3',
  template: `
    <h2>Pessoas</h2>
    <ul>
      <li *ngFor="let pessoa of pessoas">
        <span>{{pessoa.Id}}</span> {{pessoa.Nome}}
      </li>
    </ul>
    <div *ngIf="pessoas.length > 3”>Há muitas pessoas!</div>
  `
})
export class Componente3 {
  pessoas : Pessoa[] = [
    { Id: 11, Nome: 'Fulano' },
    { Id: 12, Nome: 'Sicrano' },
    { Id: 13, Nome: 'Beltrano' },
    { Id: 14, Nome: 'Varano' }
  ];
}
```

# Eventos

- Permitem a execução de funções baseadas em eventos.
  - Exemplos:

```
<button(click)=“cadastrar()">Cadastrar</button>

<p(mouseover)=“passouMouse()">Texto...</p>
```

# Eventos

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

@Component({
  selector: ‘componente4',
  template: '<div>{{saudacao}} <button(click)=“digaAlo()"> Dê
um alô! </button></div>'
})
export class Componente4 {
  saudacao = ‘Alo, mundo!’;
  digaAlo() {
    console.log(this.saudacao)
  }
}
```

# Entrada de Dados

- Two-way data binding: exibe dados do modelo e altera dados do modelo.
  - Passo 1: Alterar app.module

```
import { FormsModule } from '@angular/forms';
imports: [ BrowserModule, FormsModule ]
```

  - Passo 2: utilizar [(ngModel)] ao invés de value

```
<input  [(ngModel)]="nome" placeholder="nome">
```

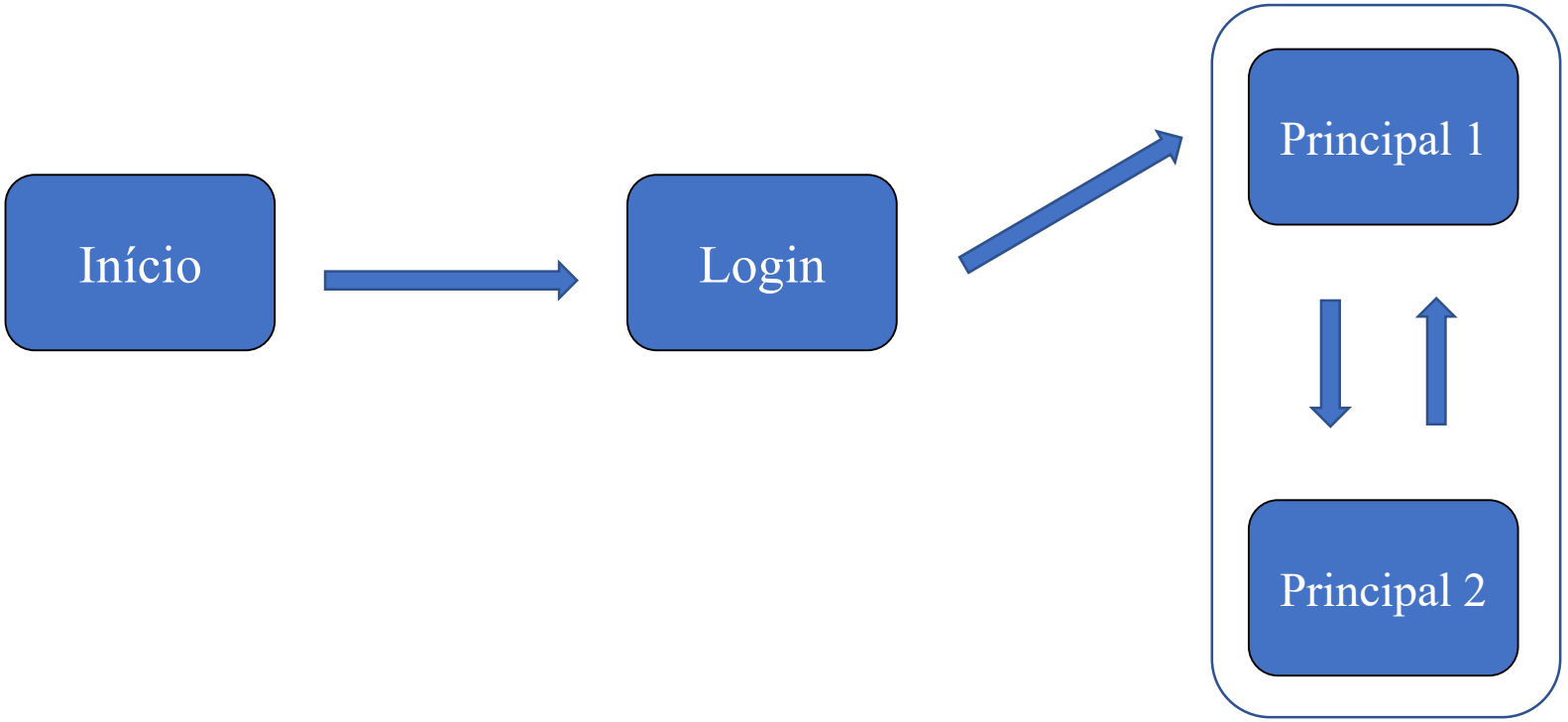
# Componente com Entrada de Dados (exemplo completo)

```
import { Component } from '@angular/core';
import { Pessoa }  from './pessoa.modelo';

@Component({
  selector: 'componente5',
  template: `
    <h2>{{titulo}}</h2>
    <div><label>id: </label>{{pessoa.Id}}</div>
    <div><label>nome: </label>
    <input  [(ngModel)]="pessoa.Nome" placeholder="nome">
    <div> Seu nome é {{pessoa.nome}}</div>
  `
})
export class Componente5 {
  titulo=”Dados de pessoa”;
  pessoa: Pessoa = {Id: 1, Nome: 'Fulano'};
}
```

# Utilizando Rotas para Navegar entre Componentes

# Usando Rotas

- Rotas são usadas para estabelecer fluxo de navegação em sistemas web;
- Vamos desenvolver o seguinte fluxo de navegação:



# Usando Rotas – Parte 1

- Inclua em app.module.ts:

```
import { RouterModule, Routes } from '@angular/router';
```

- Configure as rotas app.module.ts:

```
const appRoutes: Routes = [
{ path: '', component: InicioComponent},
{ path: 'login', component: LoginComponent } ];
```

# Usando Rotas – Parte 1

- Em app.module.ts, a seção “imports” ficará assim:

```
imports: [
    RouterModule.forRoot(
        appRoutes,
        { enableTracing: true }
    ),
    BrowserModule  ]
```

# Usando Rotas – Parte 2

- Inclua no componente raíz:

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

- Ao carregar a página, de acordo com as rotas, `InicioComponent` será carregado dentro de `<router-outlet>`.

# Usando Rotas – Parte 3

- Inclua no componente InicioComponent:

```
<a routerLink="/login">login</a>
```

- Ao carregar a página, `InicioComponent` será carregado dentro de `<router-outlet>`.

- Ao clicar no link, `LoginComponent` será carregado dentro de `<router-outlet>` do componente raíz.

# Usando Rotas – Parte 4

- Atualize as rotas, incluindo a seguinte:

```
{ path: ‘principal', component: PrincipalComponent,
      children:[
      {path : '', component:Conteudo1Component},
      {path : 'c1', component: Conteudo1Component},
      {path : 'c2', component: Conteudo2Component}]
}
```

- Inclua no componente LoginComponent:

```
<a routerLink="/">voltar</a>
<a routerLink="/principal">entrar</a>
```

# Usando Rotas – Parte 5

- Inclua no componente PrincipalComponent:

```
<a routerLink=“c1">Principal 1</a>
<a routerLink=“c2">Principal 2</a>
<a routerLink="">Sair</a>

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

# Usando Rotas – Parte 6

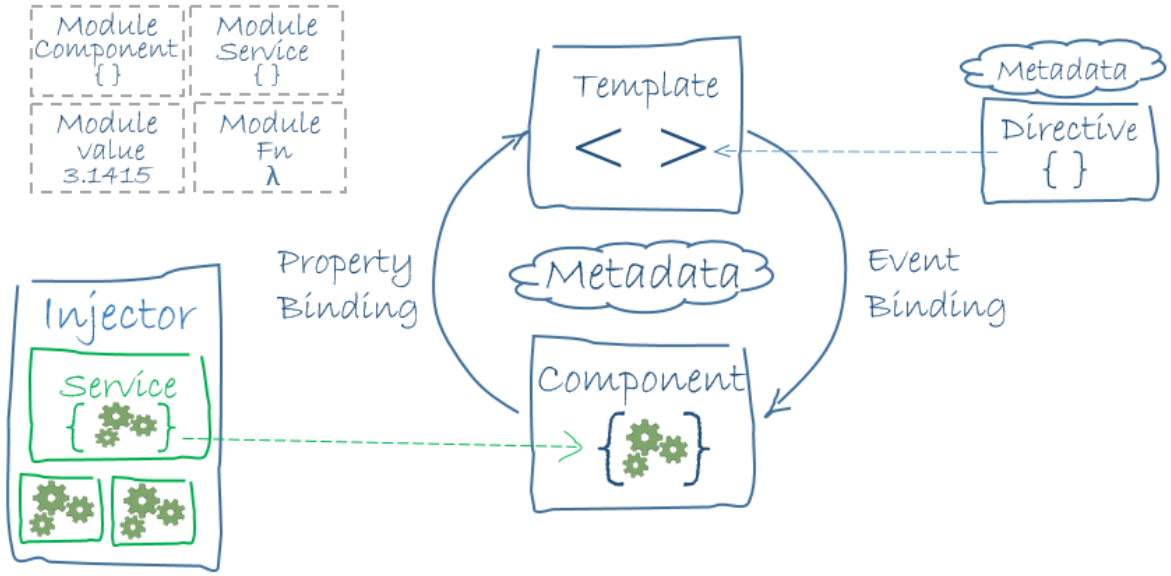
- Roteamento via código:

```
import { Router, ActivatedRoute } from
'@angular/router';


constructor(private route: ActivatedRoute, private
router: Router) {  }


login(){
     this.router.navigate(['/login'],
{relativeTo:this.route});
}
```

# Utilizando Classes de Serviços para Acessar o Back-end (Web Services)

# Utilizando Serviços (Services)

- Para que serviços (*services*)?
  - Isolam funcionalidades e fazem componentes se preocuparem somente com visão (*view*);
  - Acessam a web services;
  - Realizam chamadas assíncronas;
  - São “injetáveis”.



# Implementando Services – Passo 1

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';



@NgModule({
      declarations: [ AppComponent ],
      imports: [    BrowserModule, HttpClientModule ],
      providers: [],  bootstrap: [AppComponent]
})

export class AppModule { }
```

- Em AppModule, inclua HttpClienteModule

# Implementando Services – Passo 2

```
import { Injectable } from '@angular/core';
import { Categoria }  from './categoria.modelo';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable, of } from 'rxjs';

@Injectable({ providedIn: 'root' })
export class CategoriaService {

  private url = 'http://localhost:51698/api/categoria';

  readonly httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/x-
www-form-urlencoded' })
  };

  constructor(private http: HttpClient) { }

}
```

- Cria a classe CategoriaService; Note o uso de @Injectable()

# Implementando Services – Passo 3

- Consultado através de GET

```
getCategorias(): Observable<any> {
  return this.http.get<any>(this.url, this.httpOptions)
}
```

# Implementando Services – Passo 4

- Consultado através de GET por id

```
getCategoria(id: number): Observable<any> {
  let url_ = this.url + '/' + id
  return this.http.get<any>(url_, this.httpOptions)
}
```

# Implementando Services – Passo 4

- Inserindo através de POST

```
addCategoria(categoria: Categoria): Observable<any> {
  let u = new URLSearchParams();
  u.set('Nome', categoria.Nome.toString());
  u.set(‘Responsavel', categoria.Responsavel.toString());

  return this.http.post<any>(this.url, u.toString(),
this.httpOptions)
}
```

# Implementando Services – Passo 5

```
updateCategoria(categoria: Categoria): Observable<any> {
  let u = new URLSearchParams();
  u.set('Id', categoria.Id.toString());
  u.set('Nome', categoria.Nome.toString());
  u.set('Responsavel', categoria.Responsavel.toString());

  let url_ = this.url + '/' + categoria.Id
  return this.http.put<any>(url_, u.toString(), this.httpOptions)
}
```

- Atualizando através de PUT

# Implementando Services – Passo 6

- Removendo através de DELETE

```
deleteCategoria(id: Number): Observable<any> {
 let url_ = this.url + '/' + id
 return this.http.delete<any>(url_, this.httpOptions)
}
```

# Implementando Services – Passo 7

```
...
import { CategoriaService } from `./categoria.service`;
...
export class CatagoriaComponent implements OnInit {
...
        constructor(private pessoaService: PessoaService) { }
        ngOnInit(): void {
                this.getCategorias();
        }
        getCategorias(): void {
          this.categoriaService.getCategorias()
            .subscribe(response => {
              if (response.Status == 0) {
                this.categorias = response.Elementos
              }
              else {
                console.log(response.Detalhes)
              }
        });
      }
}
```

- Método getCategorias() é implementado para consulta.

# Implementando Services – Passo 8

```
inserir(): void {
  this.categoriaService.addCategoria(this.categoria)
  .subscribe(response => {
    if (response.Status == 0) {
      this.getCategorias();
    }
    else {
      console.log(response.Detalhes)
    }
  });
}
```

- Método inserir() no componente.

# Implementando Services – Passo 9

```
atualizar(): void {
  this.categoriaService.updateCategoria(this.categoria)
  .subscribe(response => {
    if (response.Status == 0) {
      this.getCategorias()
    }
    else {
      console.log(response.Detalhes)
    }
  });
}
```

- Método atualizar() no componente.

# Implementando Services – Passo 10

```
remover(id): void {
  this.categoriaService.deleteCategoria(id)
  .subscribe(response => {
    if (response.Status == 0) {
      this.getCategorias();
    }
    else {
      console.log(response.Detalhes)
    }
  });
}
```

- Método remover(id) do componente.

# Fontes de Pesquisa

- https://angular.io/docs/ts/latest/tutorial/
- https://www.tutorialspoint.com/angular2/angular2_overview.htm
- http://www.angular2.com/