Minicurso
Introdução ao Angular

Semana de Tecnologia e Segurança da Informação
Centro de Tecnologia da Informação e Comunicação
Universidade Federal do Pará
27/09/2017

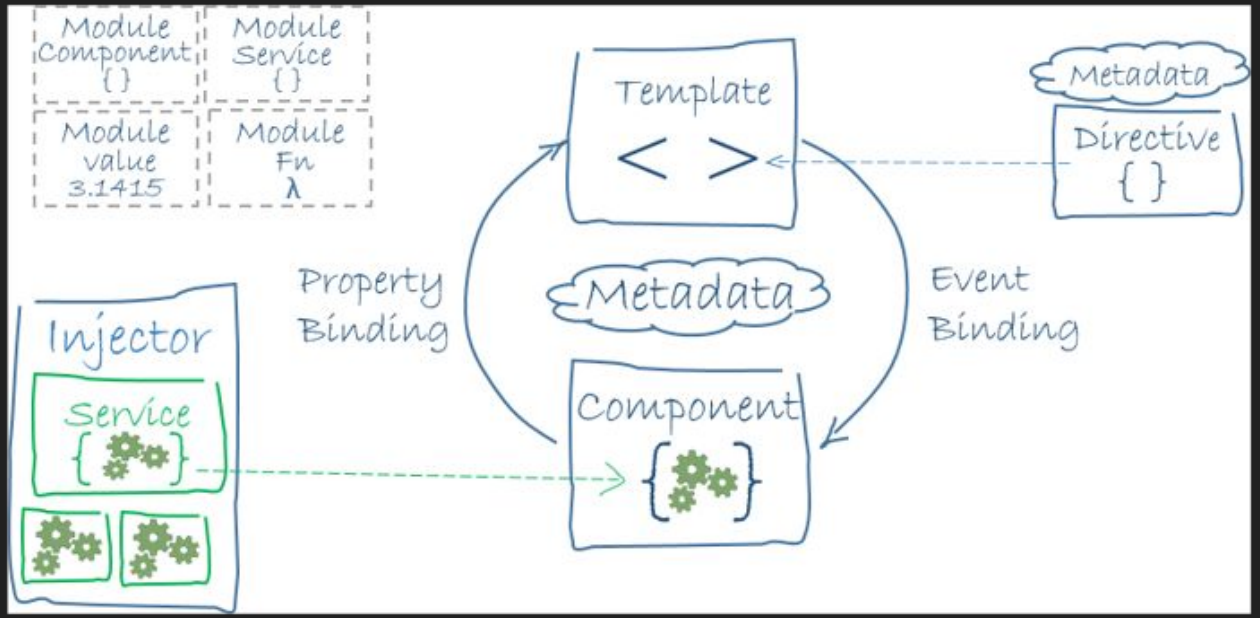
$ whoami

- Gustavo Lobato <gustavomaues@gmail.com>
- Contato no linkedin, gtalk, skype, medium, slideshare, github: gustavomaues
- Analista de TI - CTIC - UFPA
- Bacharel em Sistemas de Informação - UFPA
- Licenciado Pleno em Geografia - IFPA
- Especialista em Desenvolvimento de Aplicações para a Internet - UFPA
- Mestrando em Ciência da Computação - UFPA

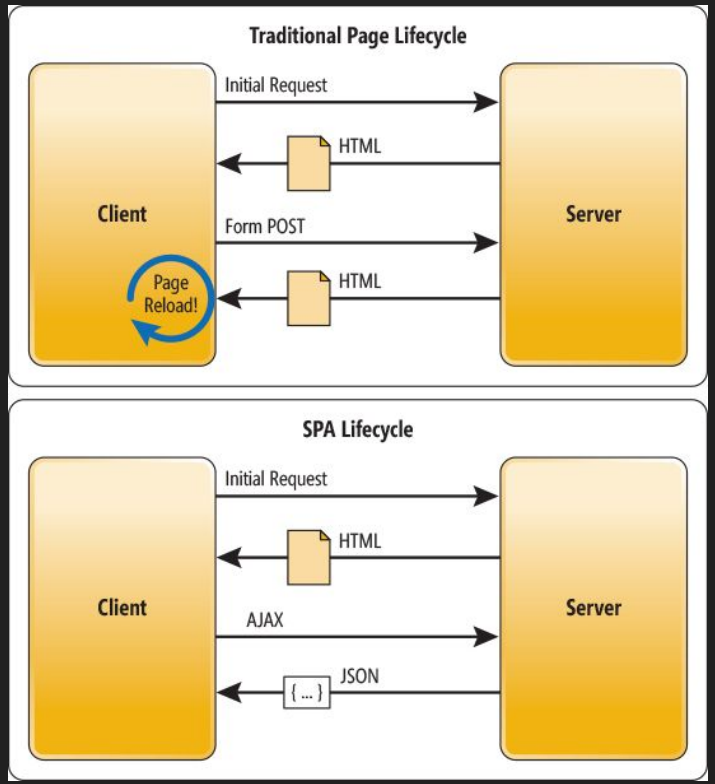
Instalação do ambiente de desenvolvimento

Introdução ao Angular

- Framework para construir aplicações clientes em html, css e javascript;
- Pode ser desenvolvido utilizando as linguagens javascript, ***typescript*** e dart;



Single Page Applications - SPA's



Projeto de exemplo: inscricao

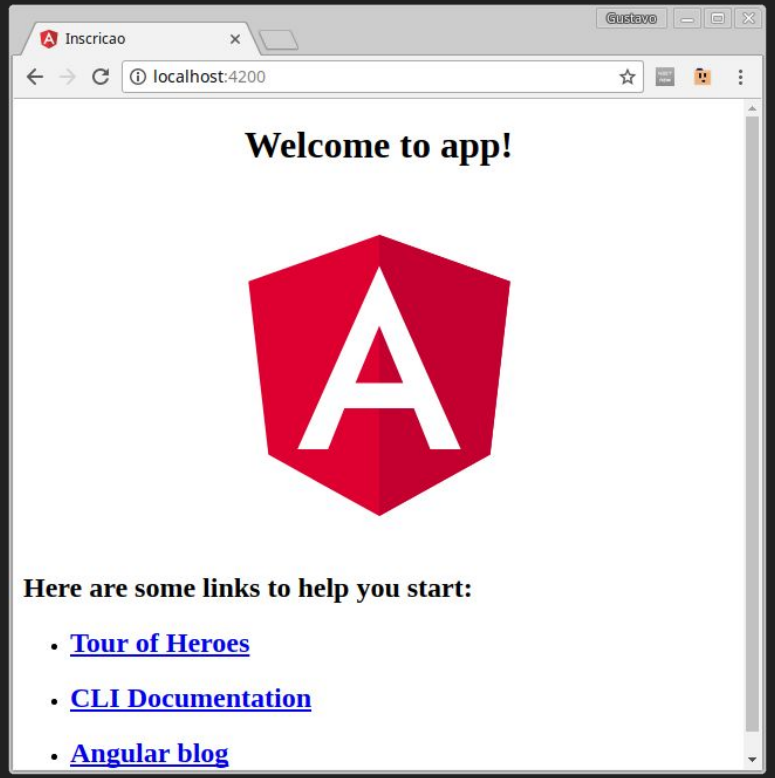
- O projeto de exemplo será um projeto de inscrição em um evento;
- DESCRIÇÃO

Criando o projeto através do Angular-CLI

- Na sua workspace, crie um novo projeto chamado “inscricao”:
- ng new inscricao --skip-tests --routing=true
  - inscricao: nome do projeto
  - --skip-tests: não gerar automaticamente classes de teste
  - --routing=true: criar automaticamente módulo de roteamento
- O comando acima cria um projeto chamado “inscricao” já com um repositório local git iniciado.
- Abra o diretório do projeto através do Visual Studio Code, navegue nos arquivos criados e conheça a estrutura básica do projeto.

Verifique se a aplicação inicial está funcionando

- ng serve
- Confira o resultado em:
- http://localhost:4200
- Pare e servidor:
- ctrl + c



Adicionando o ***PrimeNg***

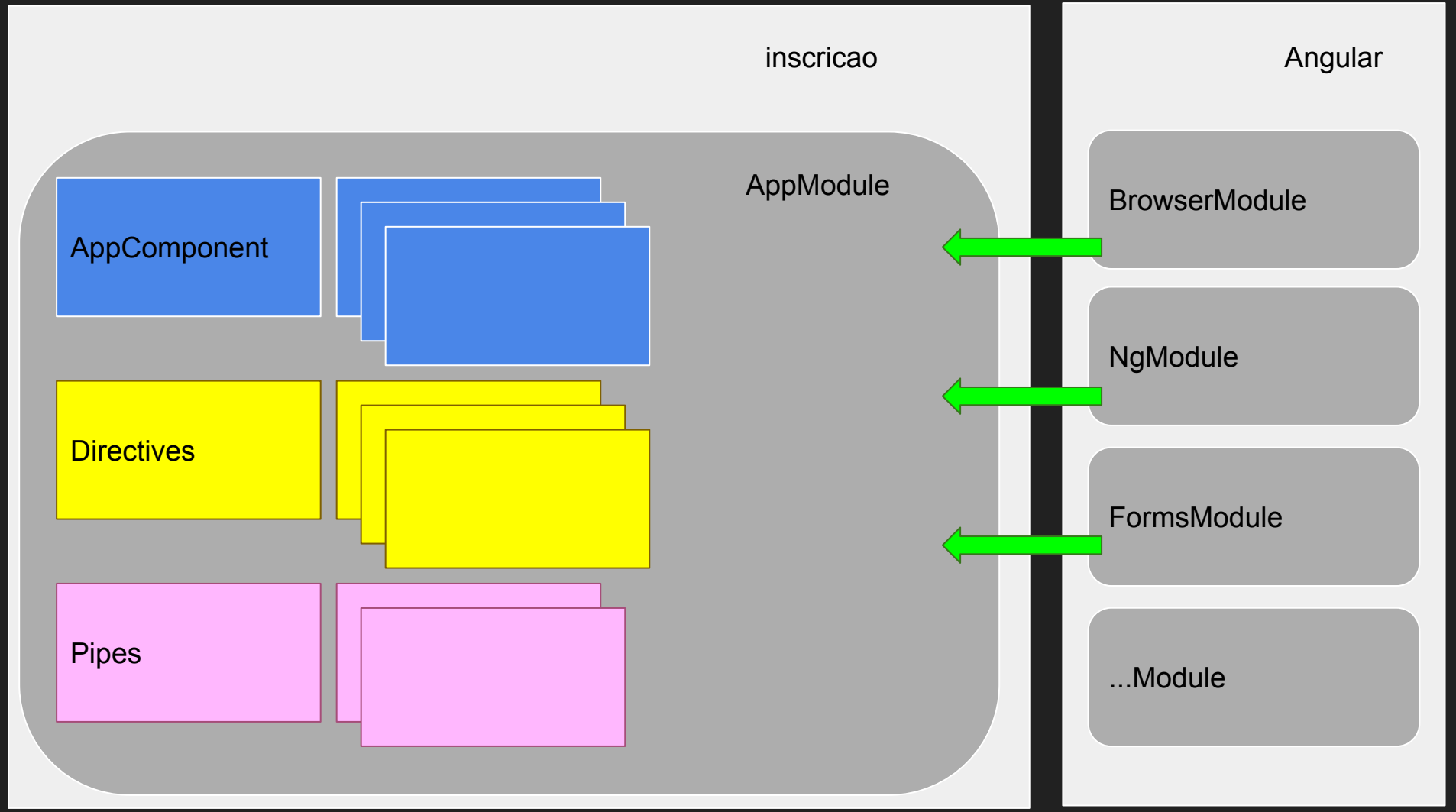
- Saiba mais em: https://www.primefaces.org/primeng/
- npm install primeng --save
- npm install font-awesome --save
- Acrescentar na propriedade styles do angular-cli.json:

```
"styles": [
        "styles.css",
        "../node_modules/font-awesome/css/font-awesome.min.css",
        "../node_modules/primeng/resources/primeng.min.css",
        "../node_modules/primeng/resources/themes/omega/theme.css"
],
```

- Execute novamente o ng serve

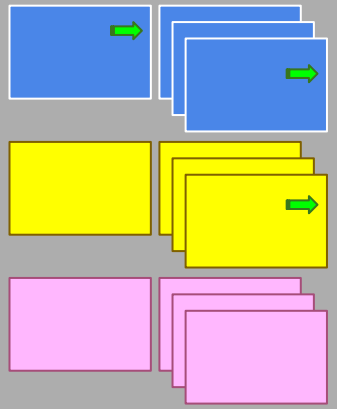
Módulos

https://angular.io/guide/ngmodule

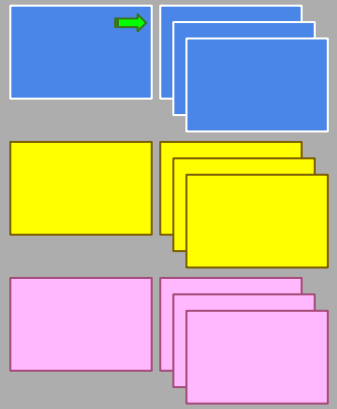

inscricao
Angular
AppModule
AppComponent
Directives
Pipes
BrowserModule
NgModule
FormsModule
...Module

inscricao

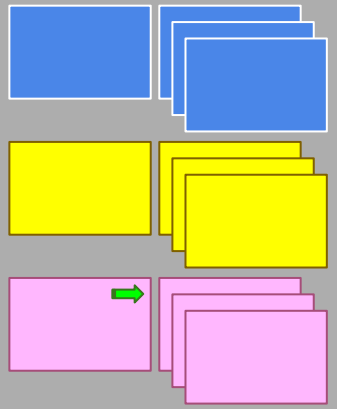
AppModule



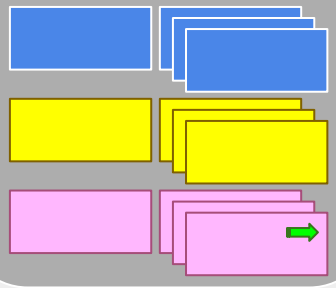
InscricaoModule



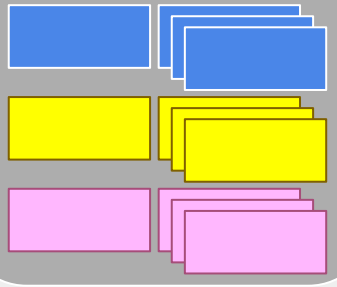
SharedModule



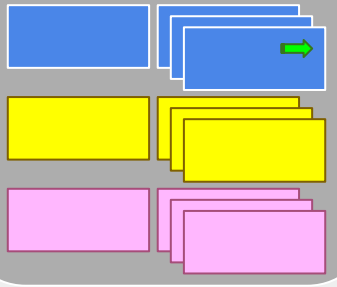
CoreModule



fulanoModule



beltranoModule



Angular

BrowserModule

NgModule

FormsModule

...Module

O que são Módulos?

- Classes que ajudam a organizar a aplicação em blocos de funcionalidades;
- Classes decoradas com ***@NgModule*** com alguns metadados que instruem como o Angular deve compilar e executar o código do módulo;
- Cada aplicação Angular deve declarar pelo menos um módulo, chamado convencionalmente de ***AppModule***, considerado o módulo raiz;
- O módulo AppModule é o único módulo que deve possuir o metadado ***bootstrap***, responsável por indicar os componentes responsáveis pela inicialização da aplicação;

ANGULAR

Principais metadados do @NgModule

- ***declarations*** - as classes de visualização pertencentes ao módulo. São três tipos de classes de visualização: *components*, *directives*, e *pipes*.
- ***exports*** - um subconjunto de *declarations* que deve estar visível e utilizável para templates de componentes de outros módulos;
- ***imports*** - outros módulos que exportam classes que são necessárias para os templates de componentes declarados deste módulo;
- ***providers*** - provedor de serviços (*services*) deste módulo que devem estar disponíveis para toda a aplicação;

Criando o módulo “Core”

- ng g m core --routing
  - g: generate
  - m: module
  - core: nome do módulo
  - --routing: cria um módulo de rotas para o módulo core

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { CoreRoutingModule } from './core-routing.module';
@NgModule({
 imports: [ CommonModule, CoreRoutingModule ],
 declarations: []
})
export class CoreModule { }
```

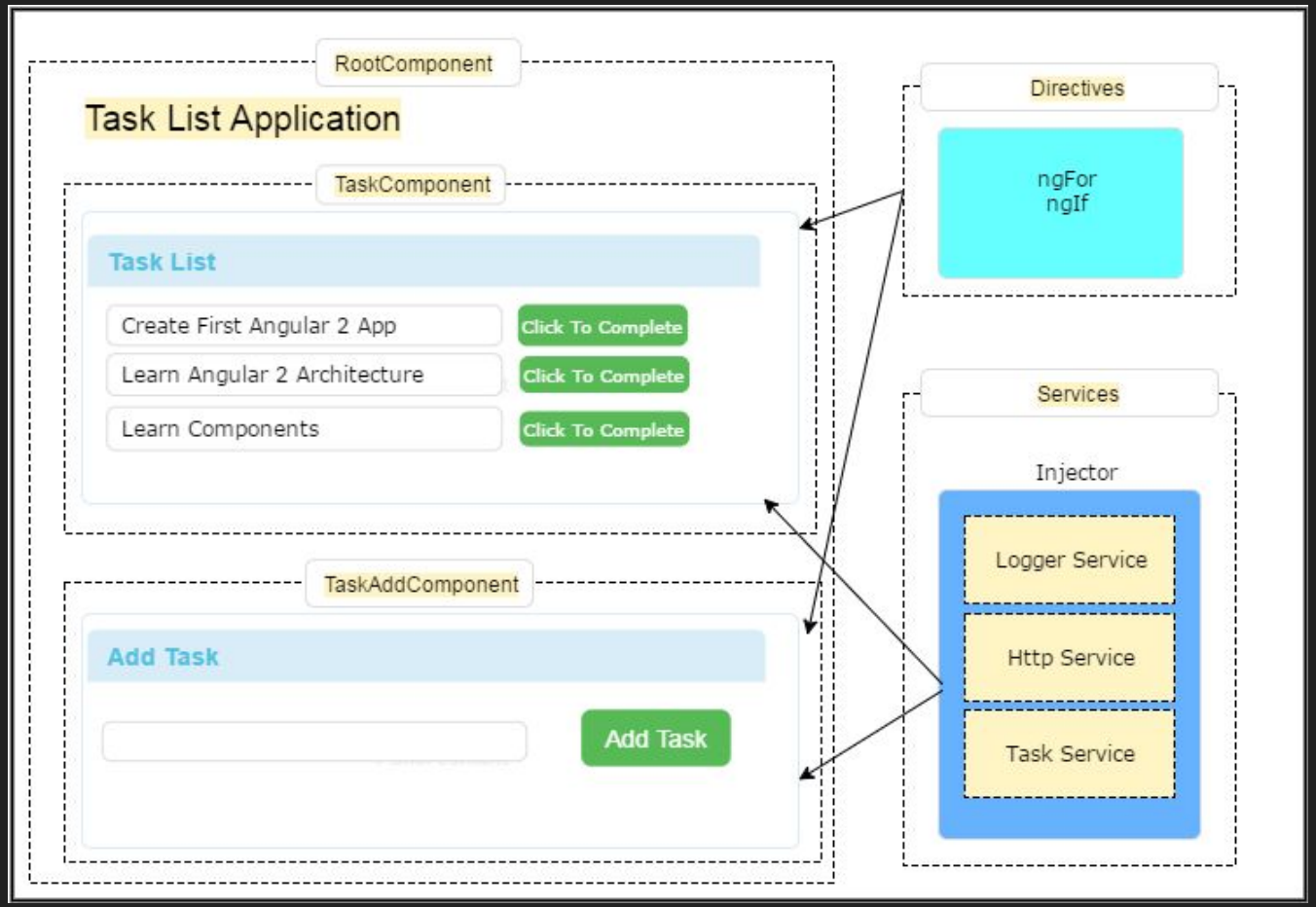
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

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

Criando o módulo “inscrição”

- Siga os mesmos passos da criação do módulo Core

Componentes

https://angular.io/api/core/Component



Fonte da imagem: https://www.tektutorialshub.com/angular2-architecture-overview-concepts/

Componentes: breve introdução

- Os componentes são classes decoradas com ***@Component*** e alguns metadados;
- São responsáveis por controlar trechos de tela denominados ***view***;
- A *view* de um componente é definida através de ***templates***;
- No *template* é declarado o código HTML que renderiza o componente;
- Além de elementos HTML, *templates* contém sintaxes próprias do Angular;
- Nas classes de componentes são definidas lógicas da aplicação que interagem com a *view* através da API de propriedades e métodos;
- O ***ciclo de vida*** de um componente é gerenciado pelo Angular;

Componentes: principais metadados

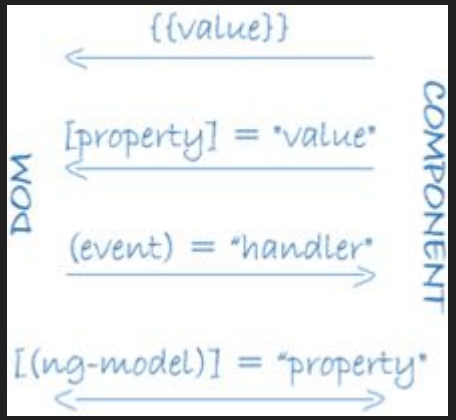
- ***selector***: metadado que define uma tag (seletor css) para o componente. Quando o angular encontra uma tag com o seletor informado (Ex. <app-rodape><app-rodape>) insere uma instância do componente em seu lugar;
- ***templateUrl***: caminho do arquivo de *template* que define o componente;
- ***styleUrls***: array com os caminhos dos arquivos de estilo que definem o componente;
- ***providers***: array de provedores de injeção de dependência necessários ao componente;

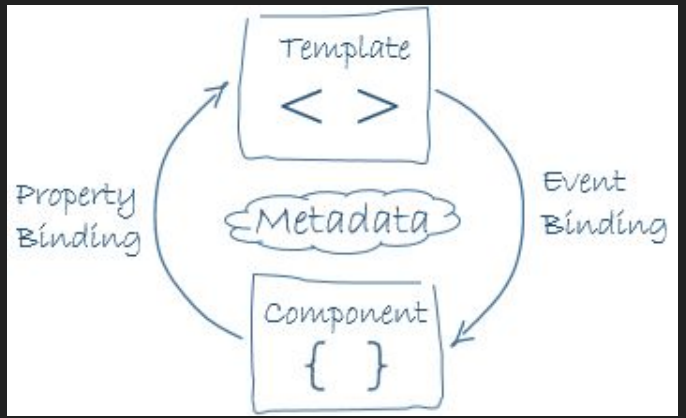
Criando o componente “topo” no módulo “core”

- ng g c core/topo --spec=false
  - g: generate
  - c: component
  - core/topo: nome do módulo / nome do componente
  - --spec=false: não criar classes de teste automaticamente
- Observe que o *TopoComponent* foi automaticamente **declarado** no módulo;
- O conteúdo html do topo ficará no arquivo app/topo/topo.component.html
- O conteúdo css do topo ficará no arquivo app/topo/topo.component.css
- Para que este componente possa ser utilizado em outros módulos, não basta declará-lo, é necessário **exportá-lo**:
  - exports: [TopoComponent]
- No AppModule, importe o CoreModule:
  - imports: [BrowserModule,CoreModule, AppRoutingModule]

Componentes: data binding

- Mecanismo para coordenar partes do template com partes de um template com partes do componente;
- Adiciona marcações de ligação no template para dizer ao Angular como ligar ambos os lados;
- Existem quatro formas da sintaxe de data binding, cada forma com uma direção:
- {{ }} *interpolation*
- [ ] *property binding*
- ( ) *event binding*
- [( )] *ngModel*





{{ interpolation }}

- No TopoComponent, crie uma propriedade chamada titulo;
- Atribua um valor ao titulo, como: “Sistema de Inscrição”
- No template, referencie essa propriedade usando {{ titulo }}; com um css interessante;
- Adicione o <app-topo></app-topo> no topo do app.component.html

```
import { Component, OnInit } from '@angular/core';
@Component({
 selector: 'app-topo',
 templateUrl: './topo.component.html',
 styleUrls: ['./topo.component.css'] })
export class TopoComponent implements OnInit {

 titulo: string = 'Sistema de Inscrição';

 constructor() { }
 ngOnInit() { }
}
```

{{ o que é possível fazer com interpolação? }}

- O título é: {{ titulo }}
- A soma de 1 + 1 é {{ 1 + 1 }}
- {{ 1 + 1 + getValor() }}
- <img src="{{ imagemUrl }}" style="height:30px">
- <div *ngFor="let participante of inscritos">{{ participante.nome }}
- ...

Use a mesma lógica do topo e crie o rodapé

- Não esqueça que o <app-rodape></app-rodape> ficará no fim do arquivo app.component.html;

Criando o componente da página de boas vindas

1. Crie um componente no módulo “core” chamado “home”;
2. Crie o conteúdo html e css desejando boas vindas;
3. Exporte o HomeComponent no CoreModule;
4. Declare o <app-home></app-home> no app.component.html;
5. Veja o resultado no navegador;

Criando o componente da página de boas vindas

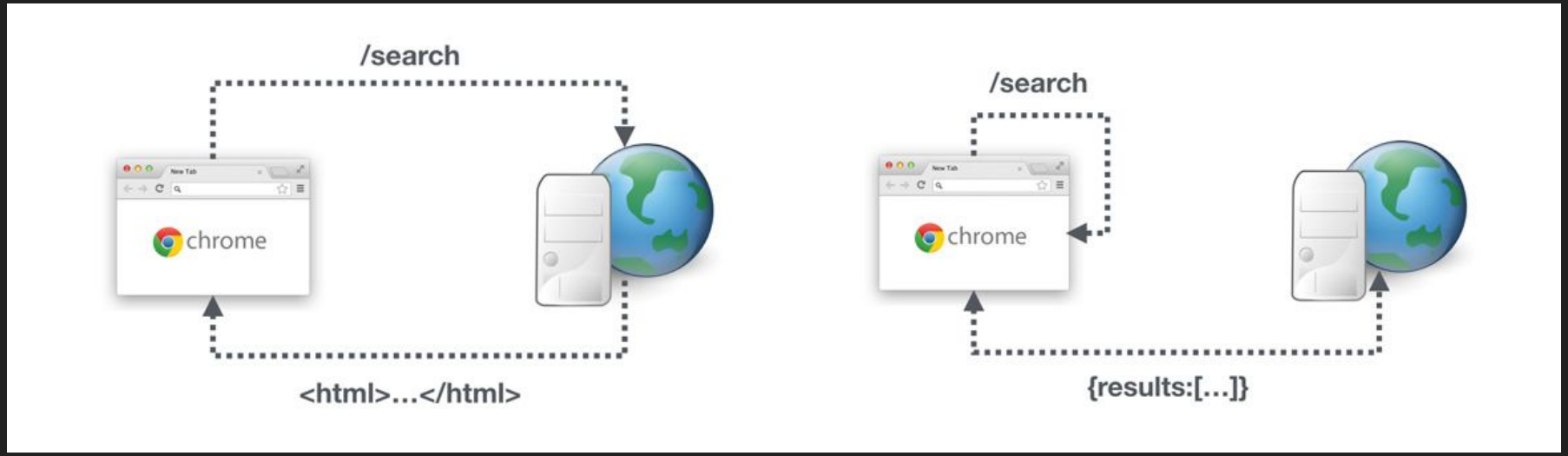
1. Crie um componente no módulo “core” chamado “home”;
2. Crie o conteúdo html e css desejando boas vindas;
3. Exporte o HomeComponent no CoreModule;
4. ~~Declare o <app-home></app-home> no app.component.html~~

- O conteúdo do HomeComponent não deve ser exibido sempre, só deve ser exibido quando o usuário acessar a raiz da aplicação ou /home;
- Vamos conhecer e usar ***ROTAS***

Routing & Navigation

https://angular.io/guide/router

Diferença entre aplicações Server Side e Client Side



Rotas: introdução

- O *router* é um *singleton*, portanto, há apenas uma instância para toda a aplicação;
- Quando uma URL é alterada no navegador, o *router* busca um componente correspondente para exibi-lo;
- É possível declarar todas as rotas em um único módulo, como no módulo raiz, porém...
- Recomenda-se criar um módulo de rota para cada módulo da aplicação;
- As rotas são mapeadas na sequência em que são declaradas, então **a ordem importa**;
- Neste caso, declaramos além de rotas raízes (root), diversas rotas filhas (child)
- Nas rotas é possível enviar e receber parâmetros, realizar redirecionamentos, controlar o acesso, entre outras funções;

Declarando a rota para o HomeComponent

- No core-routing.module

```
const routes: Routes = [
 { path: '', redirectTo: 'home', pathMatch: 'full' },
 { path: 'home', component: HomeComponent }
];
```

- Use patchMatch: ‘full’ para paths vazios;
- ***Não*** coloque “/” no path
- Entenda o funcionamento do redirectTo e do component;
- No navegador, acesse localhost:4200/ e localhost:4200/home

ANGULAR

Com calma, entenda o papel do elemento...

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

E se alguém digitar uma URL que não existe?

Error: Cannot match any routes.

- Crie um componente core/pagina-nao-encontrada;
- Crie um conteúdo amigável no html e css do componente;
- Exporte o componente no CoreModule;
- Adicione a nova rota no AppRoutingModule usando *wildcard*;

```
const routes: Routes = [
    { path: '**', component: PaginaNaoEncontradaComponent }
];
```

Mais sobre rotas e navegação… aguarde!

- Precisamos passar parâmetros na URL;
- Precisamos receber parâmetros da URL;
- Precisamos criar rotas filhas;
- Precisamos implementar segurança / permissão de acesso a rotas;

Componentes: [*Input Property Binding*]

- Exemplo com HTML:
- <p hidden>Seja bem vindo, Fulano</p>
- Exemplo com Angular e ***Input Property Binding***:

```
<p [hidden]="!isUsuarioAutenticado()">
    Bem vindo ao nosso sistema.
</p>
<p [hidden]="isUsuarioAutenticado()">
    Visitante, faça login.
</p>
```

Componentes: (*Output Event Binding*)

```
<button (click)="logado=!logado">
    {{ logado ? 'Logout' : 'Login' }}
</button>
```

```
<button (click) = "alteraAutenticacao()">
    {{ logado ? "Logout" : "Login" }}
</button>
```

ANGULAR

Partindo para exemplos mais interessantes...

- A seguir criaremos nossa página de inscrição;
- Criaremos uma classe de modelo chamada Curso;
- Criaremos um componente para exibição da lista de cursos;
- Criaremos um componente para exibição de um curso específico com a opção de inscrição;
- Veremos como um componente pai se comunica com um componente filho;
- Veremos como criar rotas filhas;

Criando a classe do modelo Curso no módulo Inscrição

- ng g cl inscricao/curso
  - cl: classe

```
export class Curso {
     id: number;
     nome: string;
}
constructor(id: number, nome: string) {
   this.id = id;
   this.nome = nome;
 }
```

Crie um componente chamado CursoDetalhe no módulo Inscrição

- ng g c inscricao/curso-detalhe --spec=false
- Crie uma propriedade “curso” no componente criado;

```
curso: Curso;
```

ANGULAR

Crie um componente chamado InscricaoForm no módulo Inscrição

- ng g c inscricao/inscricao-form --spec=false

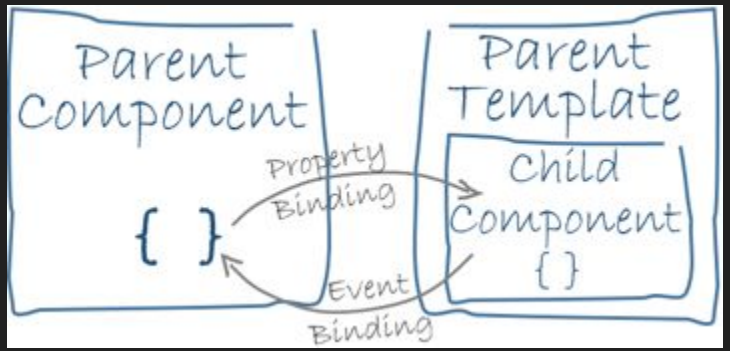
Rotas: criando rotas filhas...

- No AppModule, importe o InscricaoModule;
- No InscricaoRoutingModule, configure as rotas:
- Teste todos os caminhos: /curso /curso/lista /curso/inscricao/1

```
const routes: Routes = [
 { path: 'curso',
     children: [
       { path: '', redirectTo: 'lista', pathMatch: 'full'},
       { path: 'inscricao/:cursoId', component: InscricaoFormComponent},
       { path: 'lista', component: CursoListaComponent},
     ]
 },
];
```

Componentes: data binding, voltando...

- Data binding não ocorre somente entre o componente e o seu *template*, ou o DOM;
- mas também entre os próprios componentes;
- estes são chamados componentes *parents* e componentes *childs*;
- Já vimos que é possível usar o [property binding] em atributos do DOM, como no hidden;
- e também usamos o (event binding) em eventos do DOM, como no click;
- Como podemos criar atributos para os nossos componentes se ligarem com outros?
- Como posso fazer algo do tipo <app-curso-detalhe [curso]=”cursoX” /> ?????
- Ou algo do tipo <app-curso (onCursoSelecionado)=”acessarCurso($event)” /> ?????



Componentes: @Input property

- Editando o componente CursoDetalheComponent

```
import { Curso } from './../curso';
import { Component, Input, OnInit } from '@angular/core';
@Component({
 selector: 'app-curso-detalhe',
 templateUrl: './curso-detalhe.component.html',
 styleUrls: ['./curso-detalhe.component.css']
})
export class CursoDetalheComponent implements OnInit {

 @Input() curso: Curso;

 constructor() { }
 ngOnInit() { }
}
```

Componentes: @Input property

- Implementando o template do CursoDetalheComponent

```
<div class="ui-g curso-detalhe">
    <div class="ui-g-1">{{ curso.id }}</div>
    <div class="ui-g-9">{{ curso.nome }}</div>
    <div class="ui-g-2">Inscrição</div>
</div>
```

Componentes: @Input property

- Implementando o template do CursoListaComponent
- Observe o Property Binding sendo aplicado ao componente CursoDetalhe a partir do template do CursoLista;
- Não se preocupe com o *ngFor, ainda iremos conhecer as Diretivas;

```
<h2>Cursos Disponíveis</h2>
<app-curso-detalhe *ngFor="let cursox of cursos" [curso]="cursox">
</app-curso-detalhe>
```

Componentes: @Output

- E se quiséssemos contar quantas vezes clicaram no botão “Inscrever” para cada curso? Parece claro que iremos acrescentar um contador no CursoLista, porém, se o método “inscrever” está no CursoDetalhe, como o CursoLista poderá ser notificado?
- Na classe Curso, crie uma propriedade clicks: number = 0;
- No template do CursoDetalhe, coloque o total de clicks ao lado do nome do curso e
- altere o texto inscrição para: <a (click)=”inscrever()”>inscrever-se
- No CursoDetalhe, implemente o método “inscrever” apenas para escrever no console;
- Acesse o sistema, clique em inscrever-se e veja se o console está como esperado;

```
inscrever(): void {
    console.log(`Alguém querendo se inscrever no curso ${this.curso.nome}`);
}
```

Componentes: @Output - Emitindo um evento

- No CursoDetalheComponente, Importe:

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

- Crie um emissor de eventos do tipo Curso:

```
@Output() cursoSelecionado = new EventEmitter<Curso>();
```

- No método inscrever, emita um evento:

```
inscrever(): void {
    console.log(`Alguém querendo se inscrever no curso${this.curso.nome}`);
    this.cursoSelecionado.emit(this.curso);
}
```

- A missão do CursoDetalhe termina aqui, ele apenas emite um evento, não deve ter conhecimento de quem receberá e nem o que irão fazer com o objeto recebido;

Componentes: @Output - Recebendo um evento

- No CursoListaComponente, crie um método que incrementa os clicks em um determinado curso:

```
incrementaClicks(curso: Curso) {
    this.cursos.find(c => c.id === curso.id).incrementa();
}
```

- No template do CursoListaComponente, faça o binding do evento:

```
<app-curso-detalhe *ngFor="let curso of cursos" [curso]="curso"
(cursoSelecionado)="incrementaClicks($event)"></app-curso-detalhe>
```

- Clique novamente em inscrever-se e veja se os cliques estão incrementando;

Rotas: Como acessar uma view a partir de um link?

- Para um link chamar uma determinada rota, usa-se a propriedade *routerLink*;
- Exemplo:
- <a routerLink="/curso" routerLinkActive="link_ativo">Listar Cursos
- <a routerLink="/curso/cadastrar" routerLinkActive="link_ativo">Novo Curso
- A propriedade *routerLinkActive* adiciona uma determinada classe CSS quando o link estiver ativo;
- Implementando… vamos criar linkar link “inscrever-se” a uma rota:

```
<a [routerLink]="['/curso', 'inscricao', curso.id]"
   (click)="inscrever()">inscrever-se</a>
```

Dependency Injection & Providers

Criando o InscricaoService no módulo inscrição

- ng g s inscricao/inscricao --spec=false
  - s: módulo/nome do serviço
- Crie o método que busca todas as inscrições usando HttpClient (Não esqueça de importar o módulo HttpClientModule no InscricaoModule)

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
...
@Injectable()
export class InscricaoService {

 constructor(protected httpClient: HttpClient) { }
 URL: string = 'http://localhost:8080/api/inscricao/';

 getInscricoes(): Observable<Array<Inscricao>> {
   return this.httpClient.get<Array<Inscricao>>(`${this.URL}`);
 } ...
```

Injetando o InscricaoService no InscricaoFormComponent

```
inscricoes: Array<Inscricao>;

 constructor(private service: InscricaoService) { }

 ngOnInit() {
   this.service.getInscricoes().subscribe(
     data => { this.inscricoes = data; }
   );
 }
```

Exibindo a lista de inscrições - usando pipes

- No inscricao-form.componente.html:

```
<div class="ui-g inscricao" *ngFor="let insc of inscricoes">
 <div class="ui-g-4">{{ insc.nome }}</div>
 <div class="ui-g-4">{{ insc.curso.nome }}</div>
 <div class="ui-g-4">{{ insc.dataInscricao | date:'dd/MM/yyyy' }}</div>
</div>
```

- Observe a formatação da data usando “|” PIPE
- O Angular fornece diversos Pipes, como:
- {{ 1234.56 | currency : ‘BRL’ }}
- {{ 1234.56 | currency: ‘BRL’ | lowercase }}
- {{ 3.14159265 | number: '3.1-2' }}
- {{ jsonVal | json }}
- {{ 0.123456 | percent: '2.1-2' }}
- etc...

Exemplo de um pipe personalizado

- ng g p core/string-inversa --spec=false --export=true
  - p: pipe
  - --export=true: exporta o pipe no módulo core

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'stringInversa' })
export class StringInversaPipe implements PipeTransform {

 transform(value: any, args?: any): any {
   let novaString: string = '';
   for (let i = value.length - 1; i >= 0; i--) {
     novaString += value.charAt(i);
   }
   return novaString;
 }
}
```

Forms:

- Prática com formulários