{Frontend - Angular}

...

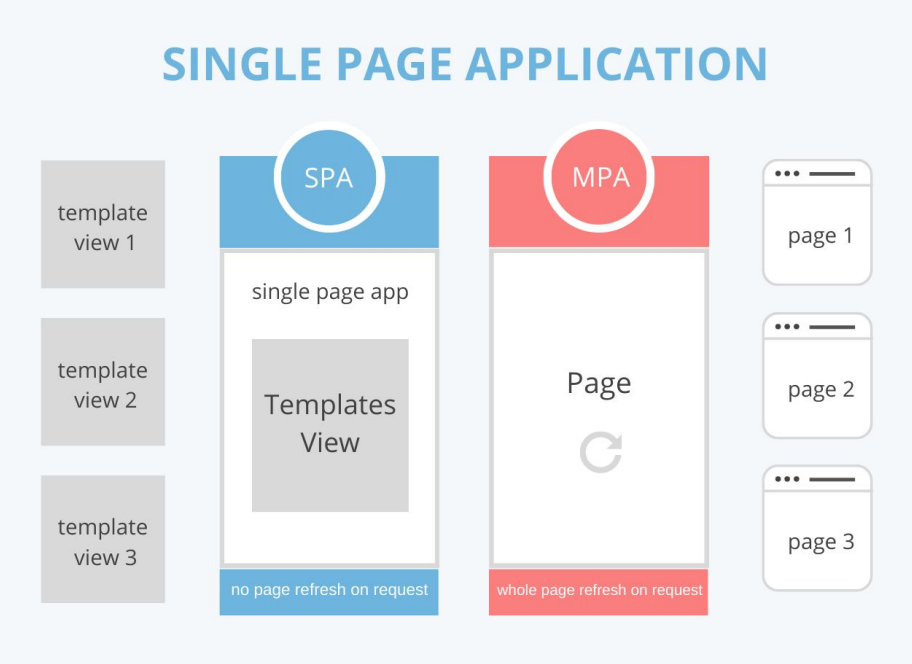
Ferramentas

https://nodejs.org/pt

https://angular.dev

Jovem Dev

O que é Angular?



- Angular é um framework de desenvolvimento web criado pelo Google.
- Construção de aplicações web single-page (SPA) de forma eficiente e escalável.

Jovem Dev

Principais Características

- Componentes Modulares: Facilita a reutilização e manutenção.
- Data Binding Bidirecional: Sincronização automática entre modelo e interface.
- Injeção de Dependências: Facilita o gerenciamento de serviços e recursos.
- TypeScript: Linguagem baseada em JavaScript, usada no Angular.

Porque utilizar?

- Desenvolvimento Rápido: Ferramentas e recursos prontos.
- Comunidade Ativa: Muitas bibliotecas e suporte.
- Escalabilidade: Ideal para aplicações de grande porte.
- Facilidade de Testes: Suporte integrado para testes unitários e de integração.

{ Jovem Dev }

Criando um projeto

Comando para criar um projeto:

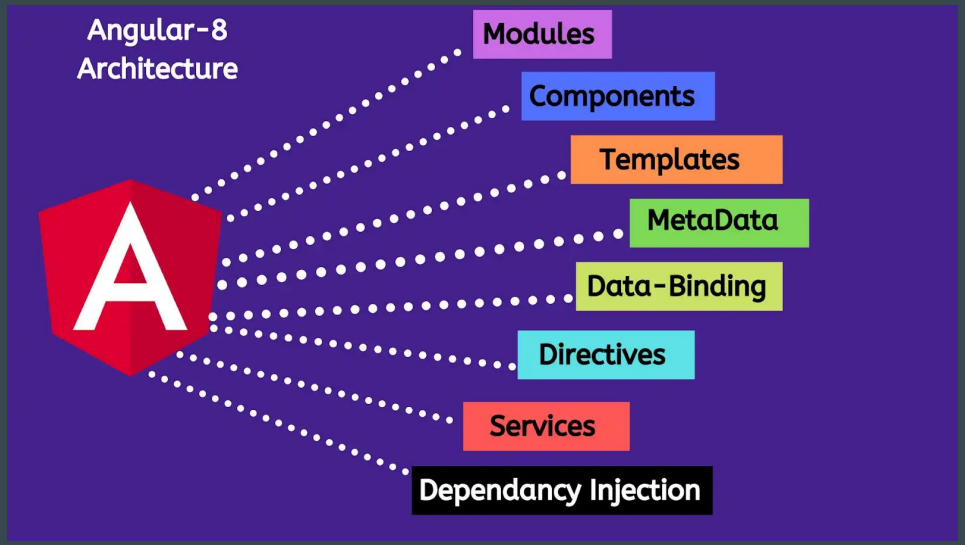
```
odair@odair-550XDA:~$ ng new meu-primeiro-projeto --routing --standalone=false
```

Comando para iniciar o projeto:

```
odair@odair-550XDA:~$ ng s
```

Jovem Dev

Estrutura de um projeto Angular



- Componentes: Blocos reutilizáveis de interface de usuário.
- Módulos: Agrupam componentes e serviços.
- Serviços: Lógica de negócios e comunicação com APIs.
- Templates: Definem a interface de usuário.

Component

- Criado manualmente
- Criado por CLI: ng g c meu-novo-componente

Jovem Dev

Exercício 01

- Criar um componente para cabeçalho
- Criar um componente para conteúdo
- Adicionar ambos no componente principal

{ Jovem Dev }

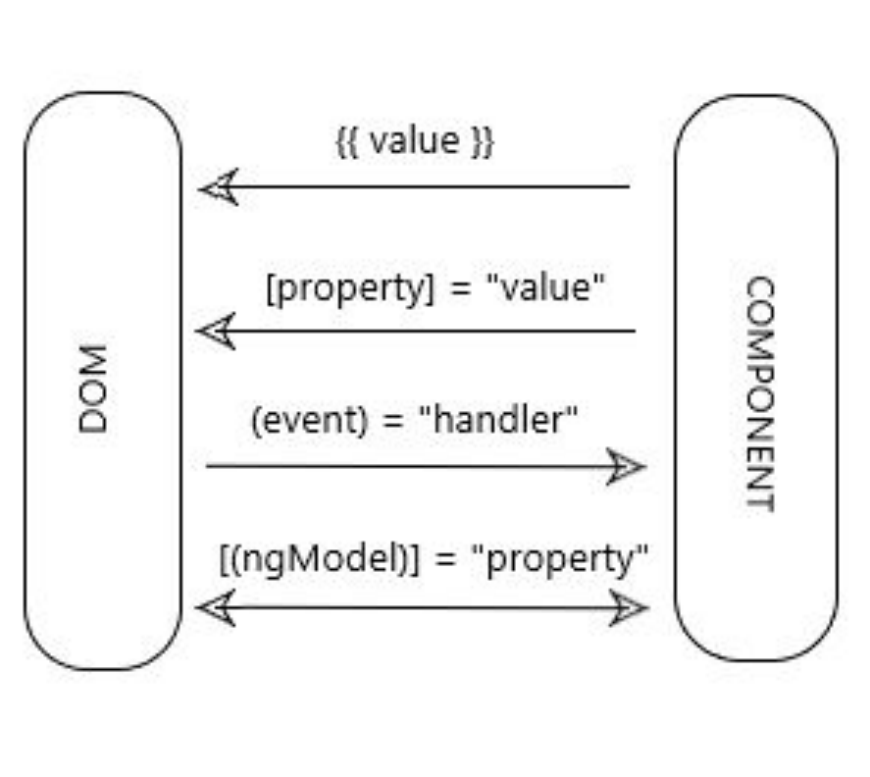
Typescript

```
export class AppComponent {
  label?: string;
  count?: number
  logical?: boolean;
  car?: {
    model: string
  };

  myPublicMethod() {

  }

  private myPrivateMethod() {

  }
}
```

- Tipos:
  - string
  - number
  - boolean
  - objetos
- Métodos
  - privados
  - públicos

{ Jovem Dev }

Data binding



- Interpolação
- Property Binding
- Event binding
- Two-way data binding

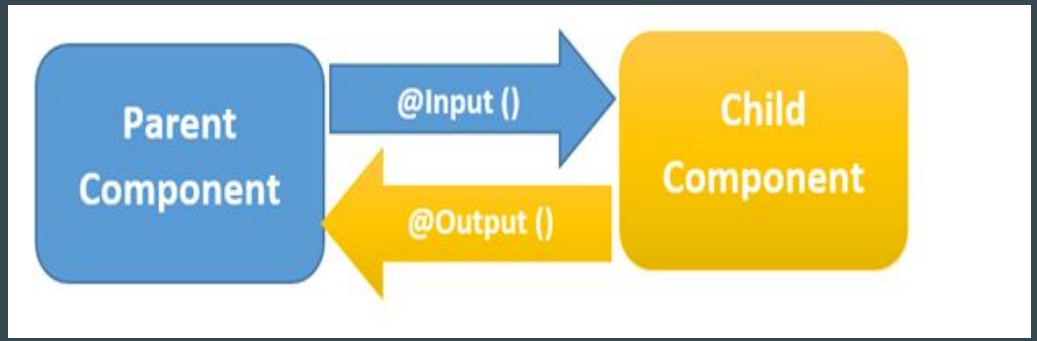
{ Jovem Dev }

Exercício 02

- Crie um componente de formulário
  - Adicione nome, email e fone
  - Imprima o valor que estiver digitando de cada input ao seu lado
  - emita um alert ao clicar no botão enviar

Jovem Dev

Comunicação entre componentes



- Input: Usado para passar dados de um componente pai para um componente filho.
- Output: Usado para enviar dados ou eventos de um componente filho de volta para o componente pai.

Jovem Dev

Decorador @Input

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.sass'
})
export class AppComponent {
  @Input() nome?: string;
```

O decorador @Input permite que um componente filho receba valores de seu componente pai.

Jovem Dev

Decorador @Output

```
10    @Output() mensagemEnviada = new EventEmitter<string>();
11
12    enviarMensagem() {
13      this.mensagemEnviada.emit('Olá, Pai!');
14    }
15
```

O decorador @Output permite que um componente filho envie eventos ou dados para seu componente pai.

Jovem Dev

Exercício 03

- Adicionar no componente de cabeçalho um título dinâmico recebido pelo pai
- Adicionar no componente de cabeçalho um campo de pesquisa onde ao sair do campo ou clicar em pesquisar o valor é emitido e printado pelo AppComponent

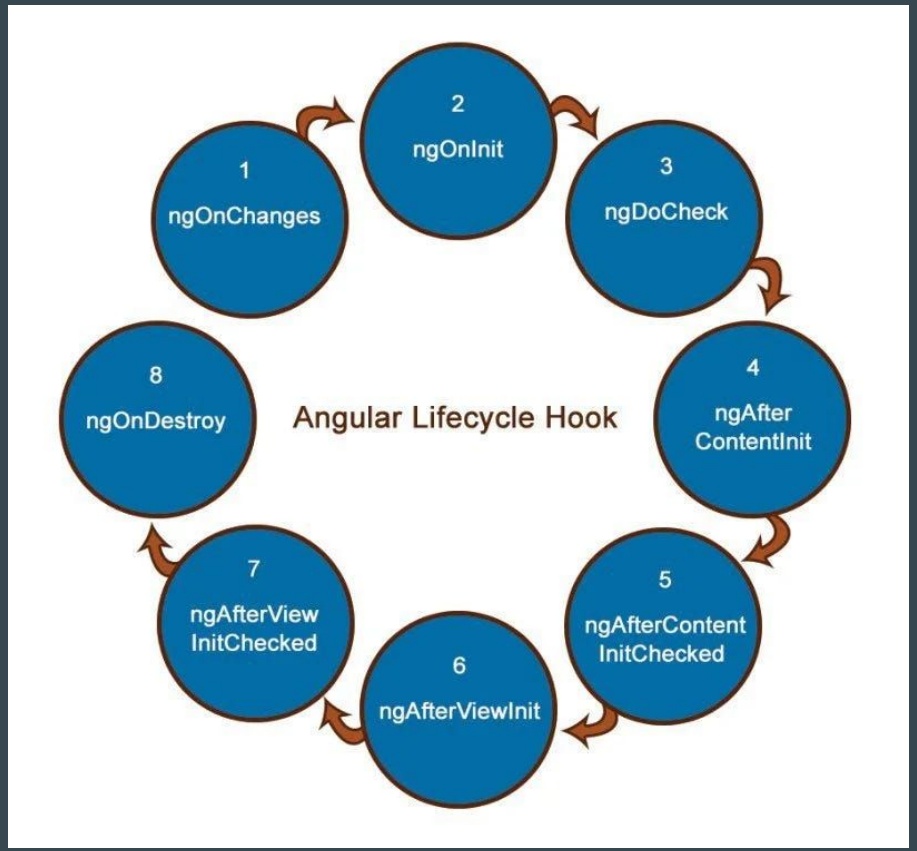
Jovem Dev

Exercício 04

- Criar um componente de formulário de cliente
- Criar um componente de listagem de cliente
  - mostrar os clientes cadastrados em tempo real
  - permitir editar e deletar cliente

Jovem Dev

Ciclo de vida dos componentes



- Definição: Os ciclos de vida são momentos específicos em que o Angular executa métodos no componente, do momento de sua criação até sua destruição.
- Permite controlar o comportamento de componentes em diferentes estágios.

Jovem Dev

Estrutura de Rotas no Angular

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LoginComponent } from './pages/login/login.component';
import { SignupComponent } from './pages/signup/signup.component';
import { MasterComponent } from './pages/master/master.component';
import { HomeComponent } from './pages/home/home.component';
import { ReportsComponent } from './pages/reports/reports.component';

const routes: Routes = [
    { path: 'login', component: LoginComponent },
    { path: 'signup', component: SignupComponent },
    { path: 'master', component: MasterComponent },
    { path: 'home', component: HomeComponent },
    { path: 'reports', component: ReportsComponent },
];

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

- Roteamento é a navegação entre diferentes componentes
- Utiliza o módulo RouterModule do Angular

{ Jovem Dev }

Exercício 05

- Crie uma rota para o formulário de cadastro criado anteriormente
- Crie uma rota para a listagem criada anteriormente
- Faça um menu capaz de navegar entre as duas rotas

{ Jovem Dev }

Rotas filhas

```
const routes: Routes = [
  {
    path: 'admin',
    component: AdminComponent,
    children: [
      { path: 'users', component: UsersComponent },
      { path: 'settings', component: SettingsComponent }
    ]
  }
];
```

Jovem Dev

Rotas com parâmetros

```
4  const routes: Routes = [
5    { path: 'product/:id', component: ProductComponent }
6  ];
7
```

```
12
13   constructor(private route: ActivatedRoute) {
14     this.route.params.subscribe(params => {
15       const productId = params['id'];
16     });
17   }
18
```

Rota padrão e rota coringa

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

Jovem Dev

Navegando entre rotas

```
<nav>
  <a routerLink="/home">Home</a>
  <a routerLink="/about">About</a>
</nav>
<router-outlet></router-outlet>
```

- Usando diretiva routerLink para navegação

Jovem Dev

Exercício 06

- Utilizando o mesmo projeto crie uma rota pai de funcionário para o cadastro e para a listagem
- Faça um botão de editar funcionário que redireciona para a tela do cadastro passando o id do usuário
- Adicione uma rota coringa informando que a página não existe
- Adicione um dashboard e coloque ele como rota padrão

Jovem Dev

Módulos

- NgModule define o escopo de componentes e serviços
- Controla como as diferentes partes da aplicação Angular se conectam
- declarations: Declara os componentes, diretivas e pipes
- imports: Importa outros módulos
- providers: Registra serviços

Jovem Dev

Benefícios da Modularização

- Reutilização de código
- Melhoria na manutenção e escalabilidade
- Melhor organização de código

Jovem Dev

Lazy Loading de Módulos

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'loja' },
  {
    path: 'client',
    canActivate: [AuthGuard],
    loadChildren: () => import('./pages/client/client.module').then((m) => m.ClientModule),
  },
  {
    path: 'loja',
    canActivate: [AuthGuard],
    loadChildren: () => import('./pages/loja/loja.module').then((m) => m.LojaModule),
  },
  {
    path: 'transacao',
    canActivate: [AuthGuard],
    loadChildren: () => import('./pages/transacao/transacao.module').then((m) => m.TransacaoModule),
  },
  {
    path: 'fechamento-loja',
    canActivate: [AuthGuard],
    loadChildren: () => import('./pages/fechamento-loja/fechamento-loja.module').then((m) => m.FechamentoLojaModule),
  },
];
```

- Carregar módulos sob demanda (Lazy Loading)
- Otimiza o desempenho da aplicação, carregando apenas o que é necessário

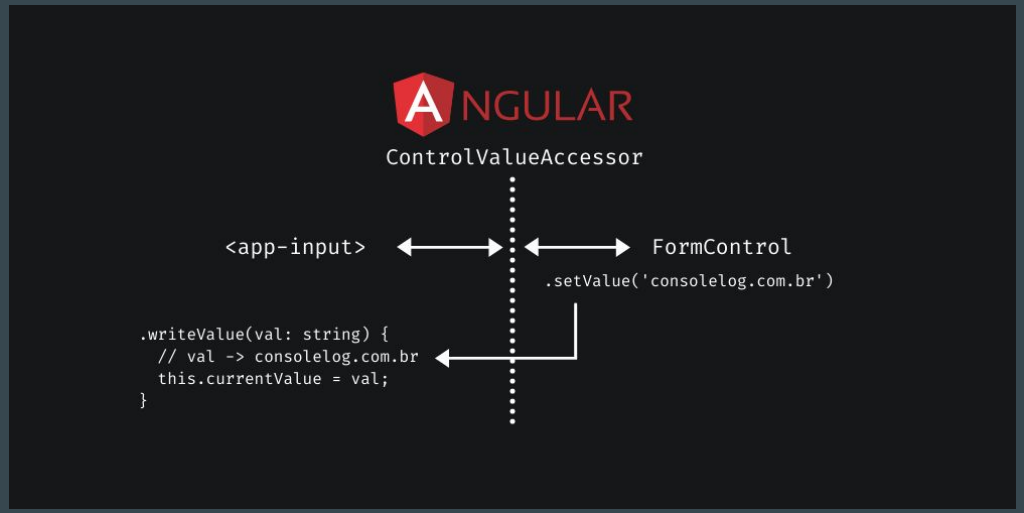
Jovem Dev

Exercício 07

- Crie um novo projeto com o nome de gerenciamento-usuario-produto
- Crie um módulo de layout onde a tela será dividida em header menu e conteúdo
- Crie um módulo de usuário e um módulo de produto
- Implemente componentes de listagem e cadastro para cada um
- Faça o menu alternar entre as telas, mostrando a tela ativa

Jovem Dev

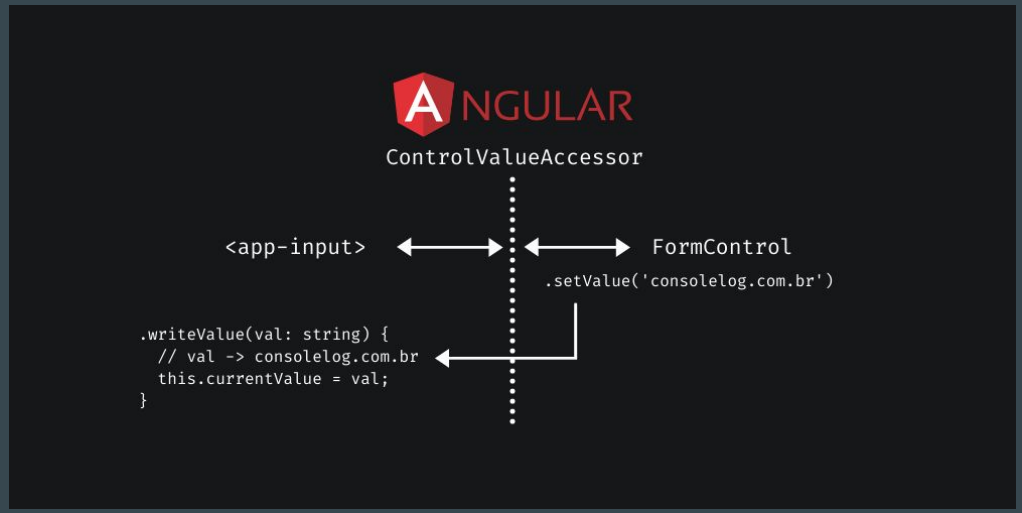
Formulários Reativos



- Abordagem baseada em programação reativa para criar e gerenciar formulários em Angular.
- Permite maior controle e flexibilidade sobre o estado e a validação dos formulários.
- Baseado em observables.
- Formulários gerados e gerenciados através de código TypeScript.

{ Jovem Dev }

Vantagens dos formulários Reativos



- Maior Controle: Controle total sobre o valor, estado e validação dos campos.
- Validação Personalizada: Possibilidade de criar validadores síncronos e assíncronos.
- Testabilidade: Testes unitários facilitados para a lógica dos formulários.
- Flexibilidade: Dinamicamente adicione/remova controles e grupos de controles.

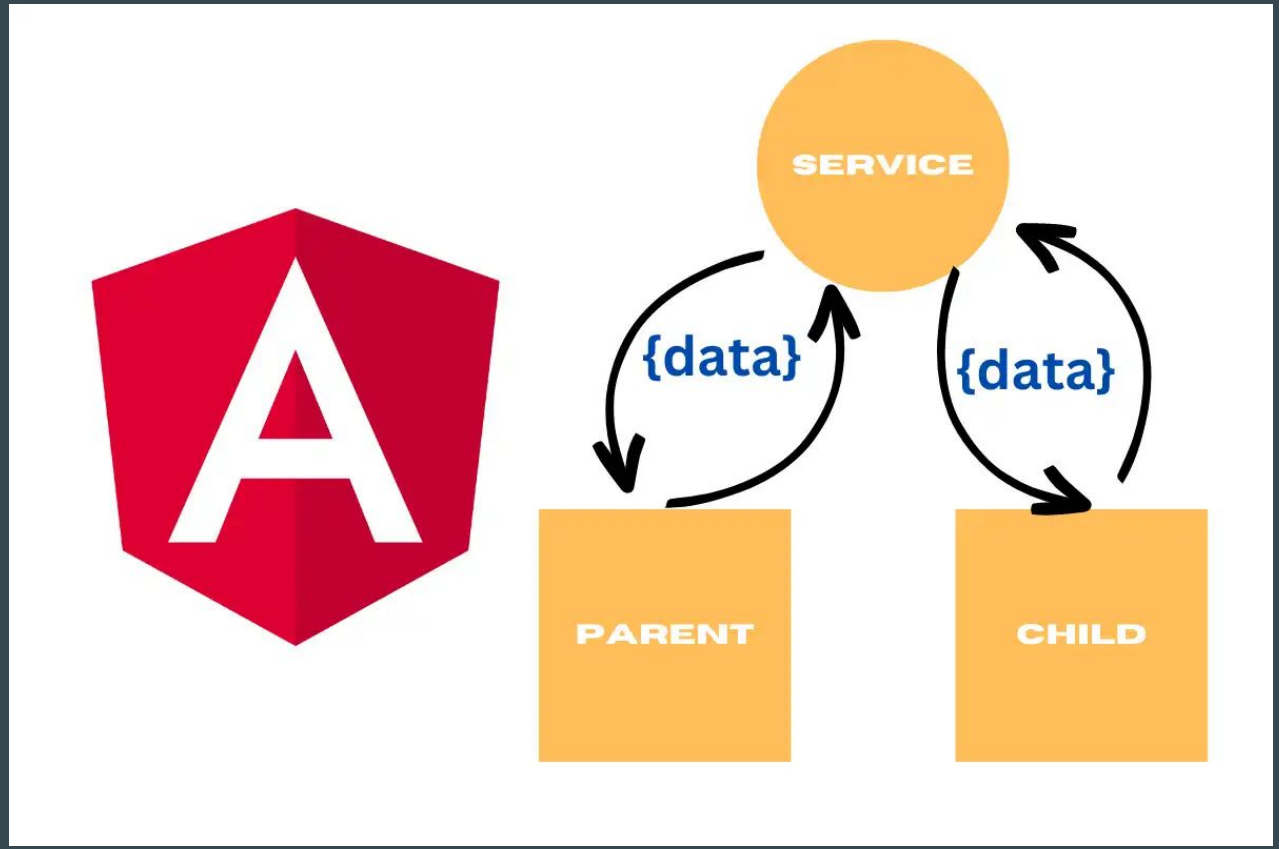
{ Jovem Dev }

Exercício 08

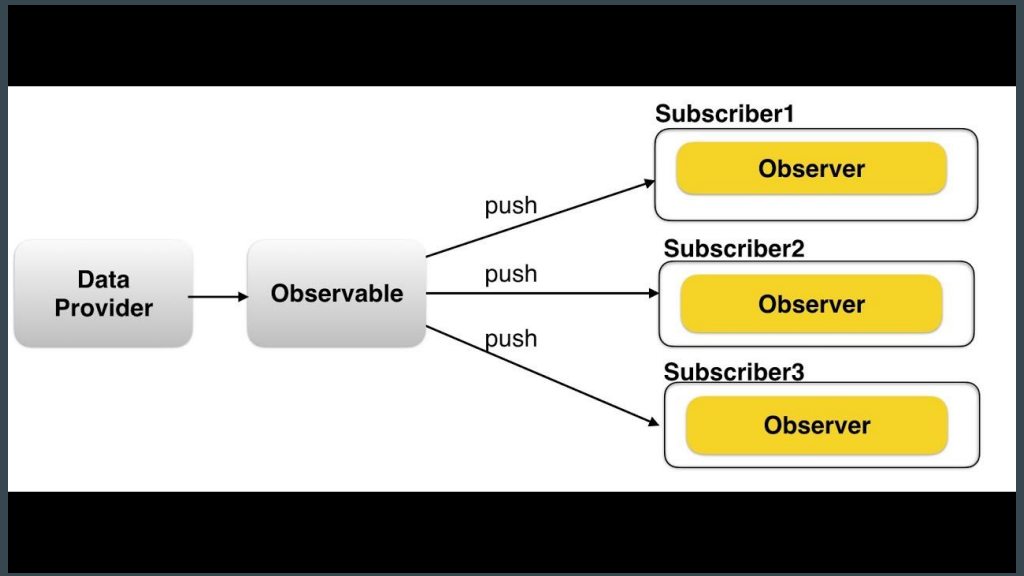
- Adapte o exercício anterior para utilizar formulário reativo
- Faça ao menos uma validação personalizada e informe o erro na tela

Jovem Dev

Services

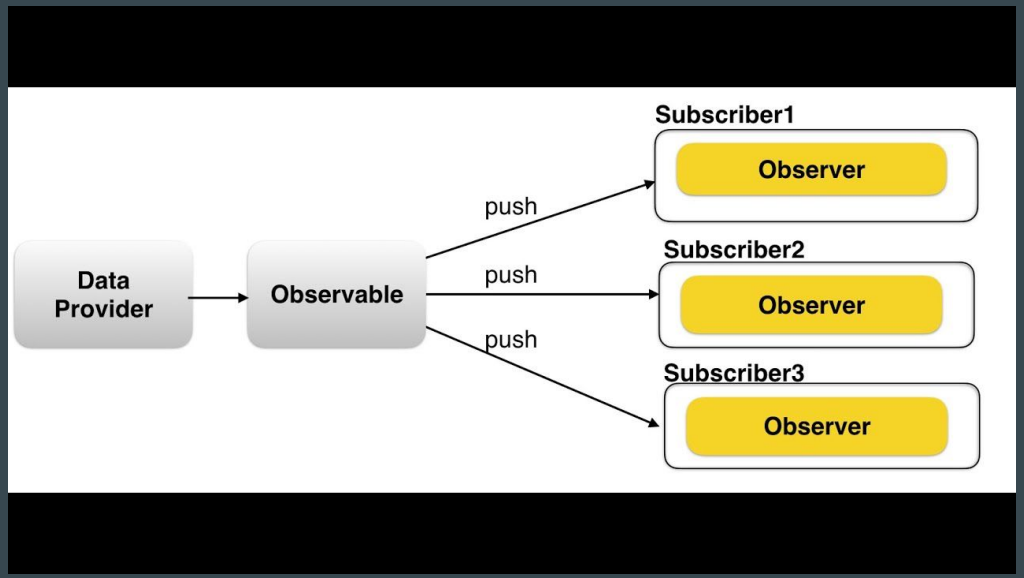


Observables



- Objetos que representam fluxos de dados que podem ser observados.
- Diferente de promessas, eles podem emitir múltiplos valores ao longo do tempo.
- Assíncronos
- Suporte a múltiplos valores
- Podem ser cancelados (unsubscribe)

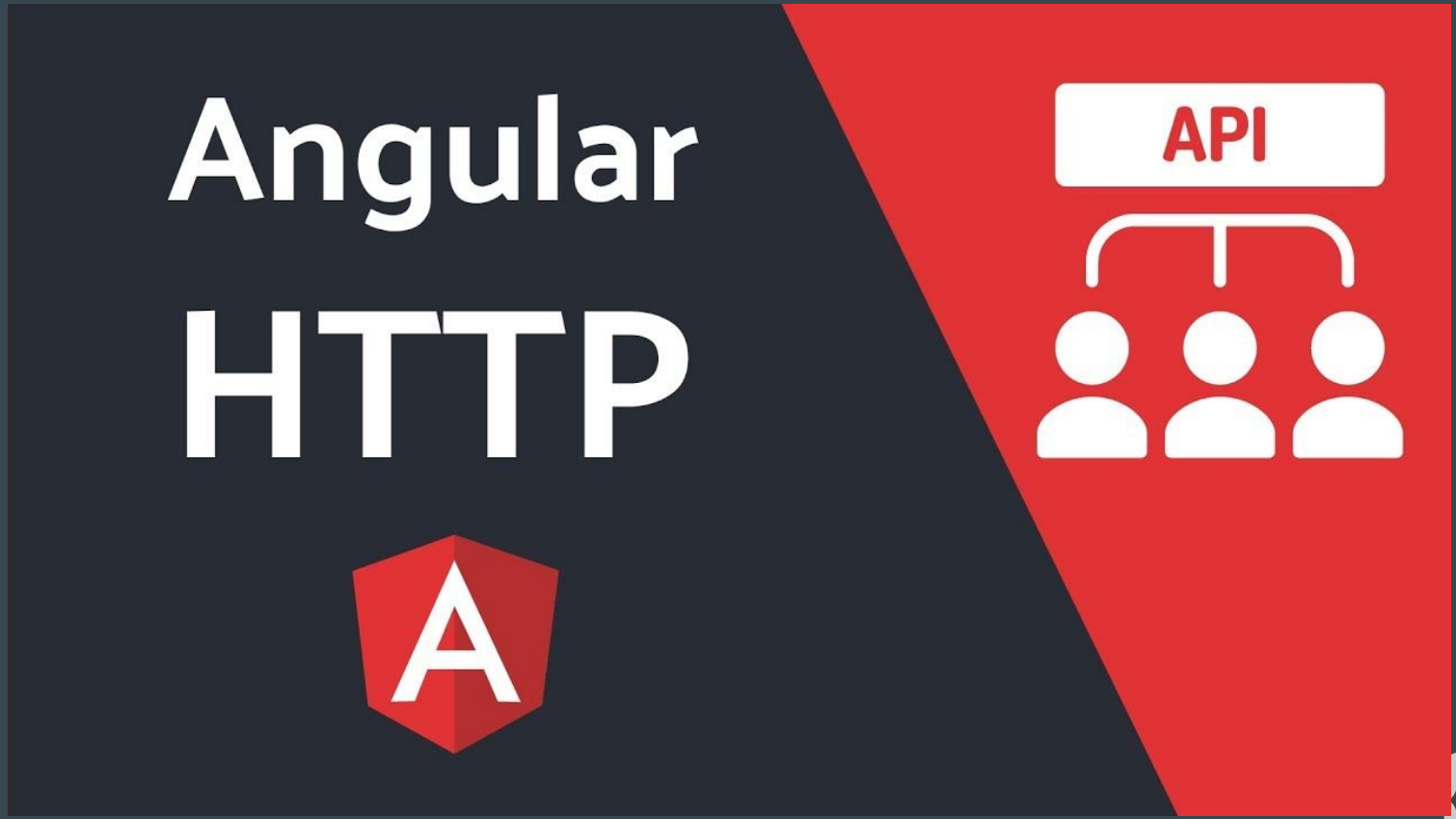
Operadores em Observables



- map: transforma valores.
- filter: filtra valores.
- merge: combina observables.

Jovem Dev

Requisições Http



Jovem Dev

Adicionando PrimeNg

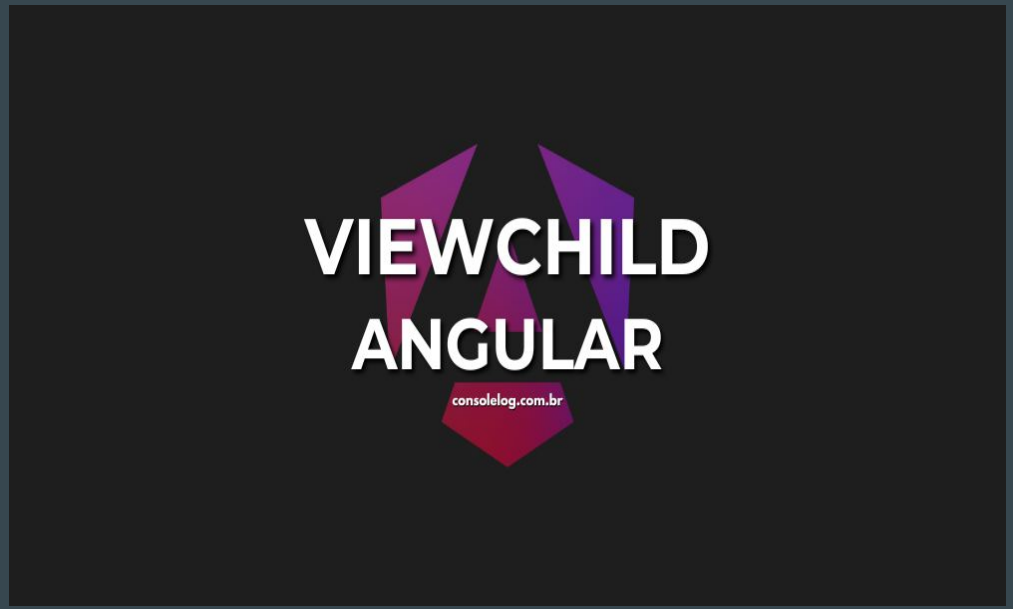
- npm install primeng primeicons
- npm install primeflex (Temas (Opcional))
- Imports no styles.scss
  - @import '../node_modules/primeng/resources/themes/saga-blue/theme.css';
  - @import '../node_modules/primeng/resources/primeng.min.css';
  - @import '../node_modules/primeicons/primeicons.css';

Desafio

- Desenvolver um album de fotos utilizando a api do JsonPlaceHolder
  - Deve listar todos os albuns com uma thumbnail carregada
  - Ao clicar em um album deve listar todas as suas fotos

Jovem Dev

@ViewChild

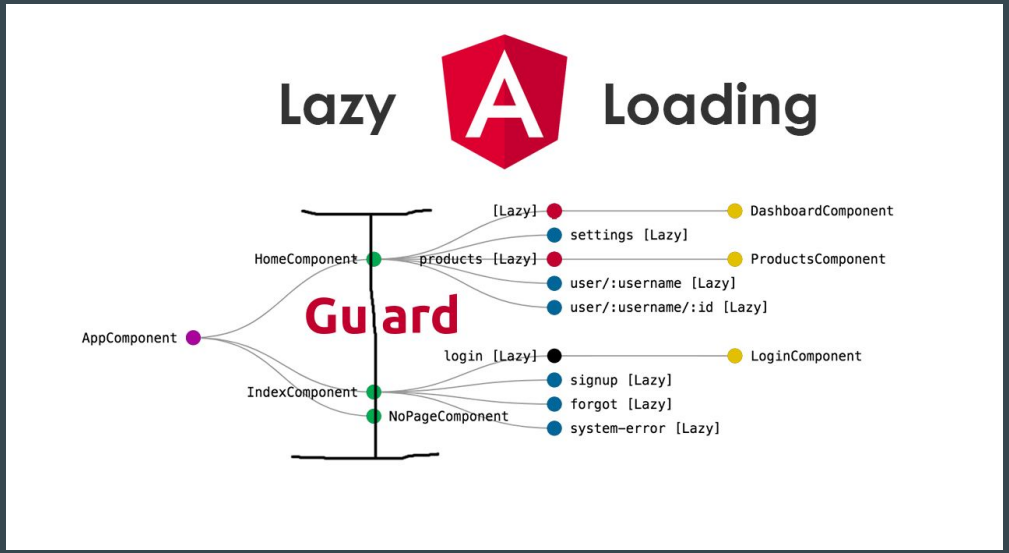


- @ViewChild é um decorador que permite selecionar e manipular elementos ou componentes filhos no template de um componente Angular.

- Ele pode ser usado para acessar:
  - Elementos nativos do DOM.
  - Componentes filhos (child components).
  - Diretivas aplicadas a elementos.

Jovem Dev

Guards



- Guards são serviços no Angular que controlam o acesso a rotas.
- Usados para verificar condições antes de ativar, desativar, carregar ou descarregar rotas.
- Tipos principais:
  - CanActivate
  - CanActivateChild
  - CanDeactivate
  - Resolve
  - CanLoad

{ Jovem Dev }

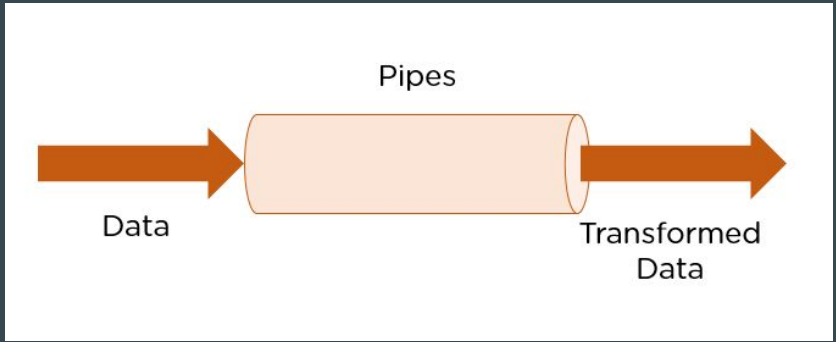
Diretivas customizadas

Angular Custom Directives

- Diretivas no Angular são classes que podem modificar o comportamento ou o layout de um elemento no DOM.

Jovem Dev

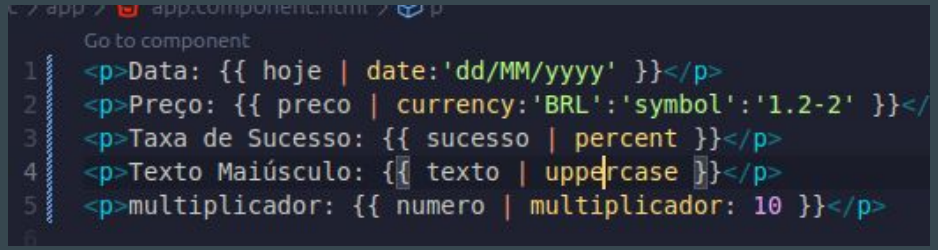
Pipes



- Pipes são usados para transformar dados em templates Angular.
- Úteis para formatar strings, números, datas, entre outros, diretamente na view.
- Sintaxe simples

Jovem Dev

Pipes Comuns



- Angular oferece vários pipes embutidos:
- date: Formata datas.
- currency: Formata valores monetários.
- percent: Formata números como porcentagem.
- uppercase / lowercase: Converte para maiúsculas/minúsculas
- json: Exibe objetos em formato JSON.

{ Jovem Dev }