

# COM222

## DESENVOLVIMENTO DE SISTEMAS WEB

Aula 14: Angular

# Angular

2

- Antes de mais nada, instalar angular cli
  - ▣ `npm install -g @angular/cli`

# Angular

## Definição

3

- Framework front-end Javascript que provê um método bem estruturado para criação de aplicações web
- Adota o modelo MVC ou MVVM
  - ▣ Separa a lógica de negócio da visão e do modelo, o que permite criar código fácil de compreender e de manter
  - ▣ Model é o data source
  - ▣ View é a página web renderizada
  - ▣ Controllers manipulam a interação entre a View e o Model
- Angular utiliza o conceito de two-way data binding, o qual permite que a view reflita automaticamente qualquer alteração no model e vice-versa

# Componentes e Funcionalidades

Module

Component

Data model

Template view

Diretiva

Expressão

Serviço

# Module

5

- Função: organizar componentes na aplicação
- Provê um namespace que permite referenciar diretivas e outros componentes
  - ▣ Facilita empacotar e reusar partes de uma aplicação
- Toda aplicação Angular tem um módulo principal (AppModule) e, opcionalmente, módulos adicionais
  - ▣ Módulos são identificados pelo decorator @NgModule
- Outros módulos podem ser adicionados ao módulo principal como dependências

# Module

6

- @NgModule tem quatro campos
  - ▣ declarations - especifica os componentes que são definidos neste módulo
    - Componentes precisam ser declarados no NgModule antes de serem usados
  - ▣ imports - descreve as dependências do módulo
  - ▣ providers - usado para injeção de dependência
  - ▣ bootstrap - indica qual componente deve ser carregado como o componente top-level quando o módulo é usado para disparar uma app

# Component

7

- Building block de uma aplicação Angular
- Controla uma ou mais seções (views) na tela
- Cada componente é composto de 3 partes
  - ▣ Component decorator (@Component)
  - ▣ View
  - ▣ Controller
- Para criar um componente
  - ▣ *ng generate component component-name*

# Data Model

8

- Representação Javascript de dados usados para popular uma view
- Dados podem vir de diferentes fontes
  - ▣ Base de dados, web service remoto, componentes web do back-end e até mesmo do código Angular do front-end
- Para criar um data model
  - ▣ `ng generate class model-or-class-name`

Exemplo:

```
export class Order {  
  constructor(  
    public name: string,  
    public email: string,  
    public phone: number,  
    public drink: string,  
    public tempPreference: string,  
    public sendText: boolean  
  ){}  
}
```



# Views com templates e diretivas

9

## □ Contexto

- ▣ Páginas HTML são baseadas no modelo DOM
- ▣ Navegadores renderizam um objeto HTML de acordo com suas propriedades DOM
- ▣ Aplicações web modernas usam Javascript para manipular o DOM e assim modificar o comportamento e a aparência dos elementos HTML renderizados pelo navegador
- ▣ Angular faz isso utilizando diretivas

# Views com templates e diretivas

10

- Diretivas possuem duas partes
  - ▣ Atributos, elementos e classes CSS extras que são adicionadas a um template HTML
    - Estendem as tags e atributos HTML
  - ▣ Código Javascript que estende o comportamento do DOM
    - Estendem as capacidades do HTML
- Utilizando diretivas, a lógica de funcionamento dos elementos passa a ser indicada pelo template HTML
  - ▣ Mais fácil de compreender
  - ▣ Lógica não fica escondida dentro do código Javascript

# Expressões

11

- Expressões são adicionadas a templates HTML utilizando `{{ }}`
- Expressões são avaliadas e seus resultados são dinamicamente adicionados à página web

# Data Binding

12

- Uma das funcionalidades mais relevantes do Angular
- Trata-se de um mecanismo para ligar dados do data model com o conteúdo que é exibido pelas páginas
- Angular provê “two-way” data binding
  - ▣ Model e view são vinculados (binding)
  - ▣ Quando o dado é modificado na página web (view), o model é automaticamente atualizado
  - ▣ Quando o dado é modificado no model, a página web (view) é automaticamente atualizada
- O model é a fonte de dados e a view é uma projeção desses dados

# Serviços

13

- Objetos que provêem funcionalidades para uma aplicação web
- Exemplo
  - ▣ Uma tarefa comum em aplicações web é fazer requisições assíncronas para um servidor web
  - ▣ Angular provê um serviço HTTP que fornece toda a funcionalidade necessária para acessar servidores web
- Angular provê diversos “built-in services”
  - ▣ Logging, parsing, requisições HTTP, ...

# Injeção de dependência

14

- Processo no qual o código de um componente define dependências de outros componentes
- Quando este código é inicializado, o componente do qual depende deve ser disponibilizado
- Exemplo
  - Se um módulo requer acesso a um servidor web via requisições HTTP, o serviço HTTP pode ser injetado dentro do módulo
    - Dessa forma, esta funcionalidade estará disponível para o módulo quando for necessária

# Ciclo de vida

15

- O ciclo de vida de um aplicação Angular tem 3 fases
  - ▣ Bootstrap
  - ▣ Compilação HTML
  - ▣ Runtime data binding

# Ciclo de vida

16

- Bootstrap
  - ▣ Ocorre quando o Angular é carregado no navegador
  - ▣ Angular inicializa seus componentes necessários, em seguida, o módulo app é inicializado e carregado
  - ▣ As dependências são injetadas (disponibilizadas)



# Ciclo de vida

17

## □ Compilação HTML

- ▣ Inicialmente, quando ocorre o carregamento da página, uma forma estática do DOM é carregada no navegador
- ▣ Em seguida, a versão estática do DOM é substituída por uma versão dinâmica que representa o Angular View. Este processo tem duas partes:
  - O static DOM é percorrido e todas as diretivas são coletadas
  - As diretivas são ligadas às funcionalidades Javascript existentes na biblioteca de diretivas do Angular, ou ao código de diretivas customizadas

# Ciclo de vida

18

- Runtime data binding
  - ▣ A fase de runtime dura até que o usuário recarregue ou saia da página
  - ▣ Se for usada two-way data binding
    - qualquer mudança no model reflete na view
    - qualquer mudança na view atualiza o model

# Separação de responsabilidades

19

- Ponto forte do Angular é a separação de responsabilidades
  - ▣ Assegura um código mais fácil de entender, dar manutenção e testar
- Deve-se respeitar a estrutura do Angular
  - ▣ Evitar “gambiarras” que violem essa estrutura

# Dicas

20

- A view provê a estrutura de apresentação da aplicação
  - ▣ Lógicas ligadas à apresentação de conteúdo devem estar contidas em diretivas no HTML template da view
- A manipulação do DOM deve sempre ser feita por código Javascript de diretivas (built-in ou criadas)
- Tarefas reutilizáveis devem ser implementadas como serviços, os quais devem ser adicionados aos módulos por meio de injeção de dependência
- Assegurar que a view reflete o estado atual do model
- Defina controladores dentro do namespace do módulo e não globalmente
  - ▣ Facilita o empacotamento e evita sobrecarregar o namespace global

21

# Diretivas

# O que são diretivas?

22

- São classes Javascript declaradas com *@directive*
- Descrevem atributos HTML específicos do angular que provêem comportamento dinâmico às páginas web
- Elas permitem que o Angular associe um comportamento específico a elementos DOM
- Três tipos
  - ▣ Diretivas de componente
  - ▣ Diretivas estruturais
  - ▣ Diretivas de atributo

# O que são diretivas?

23

- Diretivas de componente
  - ▣ Diretivas com templates
    - Templates descrevem views
  - ▣ Descrevem como os componentes devem ser processados, instanciados e usados em tempo de execução
- Diretivas estruturais
  - ▣ Manipulam a estrutura de um elemento no DOM
  - ▣ Começam com o sinal \*
    - \*ngIf, \*ngSwitch e \*ngFor
  - ▣ Afetam porções inteiras do DOM

# O que são diretivas?

24

- Diretivas de atributo
  - ▣ Muda a aparência e o comportamento de elementos do DOM
    - `ngStyle` e `ngClass`
  - ▣ Usados nos mecanismos de data binding e event binding
  - ▣ Afetam apenas o elemento associado
- Além das diretivas internas do Angular, é possível criar diretivas customizadas com o comando:
  - ▣ `ng generate directive nome-da-diretiva`, ou
  - ▣ `ng g directive nome-da-diretiva`



# ngModule

25

- É uma classe Typescript anotada com o decorador @NgModule
  - ▣ Sua função é organizar o código
    - Componentes, diretivas, serviços, roteamento
- Toda aplicação Angular tem no mínimo um módulo, chamado módulo raiz ou AppModule (por convenção)
  - ▣ Novos módulos são criados com o comando
    - ng generate module nome-do-módulo
    - ng g module nome-do-módulo

# ngIf

26

- Exibe ou esconde um elemento com base numa condição
- A condição é determinada pelo resultado de uma expressão que é passada dentro da diretiva
- Se o resultado da expressão retornar falso, o elemento é removido do DOM
- Exemplos

```
<div *ngIf="orderForm.value.name=='demo'">exibe se name é'demo'</div>  
<div *ngIf="a > b">exibe se a for maior que b</div>  
<div *ngIf="str == 'yes'">exibe se str for a string 'yes'</div>  
<div *ngIf="myfunc()">exibe se myfunc() retornar true</div>
```

# ngSwitch

27

- Exibe diferentes elementos de acordo com o resultado de uma condição
  - ▣ Similar a usar ngIf várias vezes, por exemplo:

```
<div class="container">  
  <div *ngIf="myVar == 'A'">Var é A</div>  
  <div *ngIf="myVar == 'B'">Var é B</div>  
  <div *ngIf="myVar != 'A' && myVar != 'B'">Var é outra coisa</div>  
</div>
```

- ▣ pode ser transformado em:

```
<div class="container" [ngSwitch]="myVar">  
  <div *ngSwitchCase="'A'">Var é A</div>  
  <div *ngSwitchCase="'B'">Var é B</div>  
  <div *ngSwitchDefault>Var é outra coisa</div>  
</div>
```

# ngFor

28

- Repete um elemento DOM (ou uma coleção de elementos), passando um elemento do array em cada iteração
- Sintaxe: `*ngFor="let item of item-collection"`
  - ▣ *item* especifica uma variável que recebe em cada iteração um elemento de *item-collection*
- Exemplo: assumo um array de alunos em `app.component.ts`

```
this.alunos = [  
  { nome: 'Joao', email: 'joao@gmail.com', curso: 'CCO', ano: 'segundo' },  
  { nome: 'Maria', email: 'maria@gmail.com', curso: 'SIN', ano: 'terceiro' },  
  { nome: 'Jose', email: 'jose@gmail.com', curso: 'SIN', ano: 'terceiro' },  
  { nome: 'Ana', email: 'ana@gmail.com', curso: 'CCO', ano: 'segundo' }  
];
```

- Como podemos exibir esses dados no formato de uma tabela?

## app.component.html

```
<h4>Lista de alunos</h4>
<table>
  <thead>
    <tr>
      <th>Nome</th>
      <th>Email</th>
      <th>Curso</th>
      <th>Ano</th>
    </tr>
  </thead>
  <tr *ngFor="let a of alunos">
    <td>{{ a.nome }}</td>
    <td>{{ a.email }}</td>
    <td>{{ a.curso }}</td>
    <td>{{ a.ano }}</td>
  </tr>
</table>
```

## Outro exemplo: alunos por curso

app.component.ts

```
this.alunosPorCurso = [  
  {  
    curso: 'CCO',  
    alunos: [  
      { nome: 'Joao', email: 'joao@gmail.com', ano: 'segundo' },  
      { nome: 'Ana', email: 'ana@gmail.com', ano: 'segundo' }  
    ]  
  },  
  {  
    curso: 'SIN',  
    alunos: [  
      { nome: 'Maria', email: 'maria@gmail.com', ano: 'terceiro' },  
      { nome: 'Jose', email: 'jose@gmail.com', ano: 'terceiro' }  
    ]  
  }  
];
```

## app.component.html

```
<h4>List of students by major</h4>
<div *ngFor="let c of alunosPorCurso">
  <h2>{{ c.curso }}</h2>
  <table>
    <thead>
      <tr>
        <th>Nome</th>
        <th>Email</th>
        <th>Ano</th>
      </tr>
    </thead>
    <tr *ngFor="let a of alunos">
      <td>{{ a.nome }}</td>
      <td>{{ a.email }}</td>
      <td>{{ a.ano }}</td>
    </tr>
  </table>
</div>
```

# ngStyle e ngClass

32

- ngStyle
  - ▣ Ajusta as propriedades CSS de um elemento a partir de expressões do Angular
- ngClass
  - ▣ Ajusta e modifica classes CSS para um dado elemento em tempo de execução (dinamicamente)



33

# Data binding

# Data binding

34

- Funcionalidade usada para “amarrar” valores de duas variáveis Javascript
  - ▣ Quando o valor de uma variável muda, o da outra é automaticamente atualizado
- No Angular, data binding é usado para amarrar variáveis da view com variáveis da model
  - ▣ Na model, variáveis contêm valores primitivos Javascript
    - Strings, números, ...
  - ▣ A view define como renderizar a model
- **View** e **model** podem ser amarradas diretamente no HTML usando **diretivas**

# Data binding

35

- Dois tipos de data-binding
  - ▣ One-way binding
    - Renderiza dados do model na view
    - Altera dados do model em virtude de alterações na view
  - ▣ Two-way binding
    - Alterações no model refletem na view e alterações na view refletem no model

# One-way binding

36

- One-way binding
  - ▣ Interpolação de string
  - ▣ Binding de propriedade ou atributo
  - ▣ Binding de evento
  - ▣ Binding de classe
  - ▣ Binding de estilo

# One-way binding

## Interpolação de string

37

- Faz o binding do model com a view usando {{ }}

```
<h1>{{ title }}</h1>
```

binding-example.component.html

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

binding-example.component.ts

```
@Component({  
  selector: 'app-binding-example',  
  templateUrl: './binding-example.component.html',  
  styleUrls: ['./binding-example.component.css']  
})  
export class BindingExampleComponent implements OnInit {  
  constructor() { }  
  title = 'Exemplo de data binding';  
  ngOnInit() {  
  }  
}
```

```
<app-binding-example></app-binding-example>
```

app.component.html

# One-way binding

## Interpolação de string

38

### □ Outro exemplo usando um modelo de dados

```
export class Order {  
  constructor(  
    public name: string,  
    public email: string,  
    public phone: number,  
    public drink: string,  
    public tempPreference: string,  
    public sendText: boolean  
  ) {}  
}
```

order.ts

```
{{ orderModel.email }}
```

app.component.html

ou

```
{{ orderModel | json }}
```

app.component.html

Assumindo que uma instância do modelo de dados, chamada **orderModel**, tenha sido criada no app.component.ts

# One-way binding

Binding de propriedade ou atributo

39

- Amarra o model com a view usando a seguinte sintaxe:
  - ▣ [nome-atributo-HTML] = “nome-de-propriedade-do-model”

# One-way binding

## Binding de propriedade ou atributo

40

```
<input type="text" name="content" [value]="msg" />
```

binding-example.component.html

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

binding-example.component.ts

```
@Component({  
  selector: 'app-binding-example',  
  templateUrl: './binding-example.component.html',  
  styleUrls: ['./binding-example.component.css']  
})  
export class BindingExampleComponent implements OnInit {  
  constructor() { }  
  msg = 'Alguma mensagem';  
  ngOnInit() {  
  }  
}
```

```
<app-binding-example></app-binding-example>
```

app.component.html



# One-way binding

## Binding de evento

41

- Atualiza ou envia o valor de uma variável da view (DOM) para a model (classe de componente)
  - ▣ (event-name) = “função-disparada-pelo-evento”

# One-way binding

## Binding de evento

42

```
<h1>{{ title }}</h1>
```

binding-example.component.html

```
<button (click)="changeTitle()">Muda o título quando clicar neste botão</button>
```

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

binding-example.component.ts

```
@Component({
```

```
  selector: 'app-binding-example',
```

```
  templateUrl: './binding-example.component.html',
```

```
  styleUrls: ['./binding-example.component.css']
```

```
})
```

```
export class BindingExampleComponent implements OnInit {
```

```
  constructor() { }
```

```
  title = 'Data binding example';
```

```
  ngOnInit() {
```

```
  }
```

```
  changeTitle() {
```

```
    this.title = 'Event binding';
```

```
  }
```

```
}
```

```
<app-binding-example></app-binding-example>
```

app.component.html

# Two-way binding

43

- Combina binding de *propriedade* e de *evento* usando a diretiva **ngModel**
  - ▣ [(ngModel)] = “nome-da-propriedade-no-model”
- Mudanças no componente (model) refletem na view; mudanças na view alteram o model
- Muito usado em formulários de entrada de dados

# Two-way binding

44

```
<input type="text" name="content" [(ngModel)]="msg" />
```

binding-example.component.html

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

binding-example.component.ts

```
@Component({  
  selector: 'app-binding-example',  
  templateUrl: './binding-example.component.html',  
  styleUrls: ['./binding-example.component.css']  
})  
export class BindingExampleComponent implements OnInit {  
  constructor() { }  
  msg = 'Alguma mensagem';  
  ngOnInit() {  
  }  
}
```

```
<app-binding-example></app-binding-example>
```

app.component.html

# Two-way binding

45

## □ Outro exemplo usando um modelo de dados

```
export class Order {  
  constructor(  
    public name: string,  
    public email: string,  
    public phone: number,  
    public drink: string,  
    public tempPreference: string,  
    public sendText: boolean  
  ) {}  
}
```

order.ts

Acessando/alterando a propriedade email do data model:

```
<input [(ngModel)]="orderModel.email" type="email"  
class="form-control" name="email">
```

app.component.html

Assumindo que uma instância do modelo de dados, chamada **orderModel**, tenha sido criada no app.component.ts

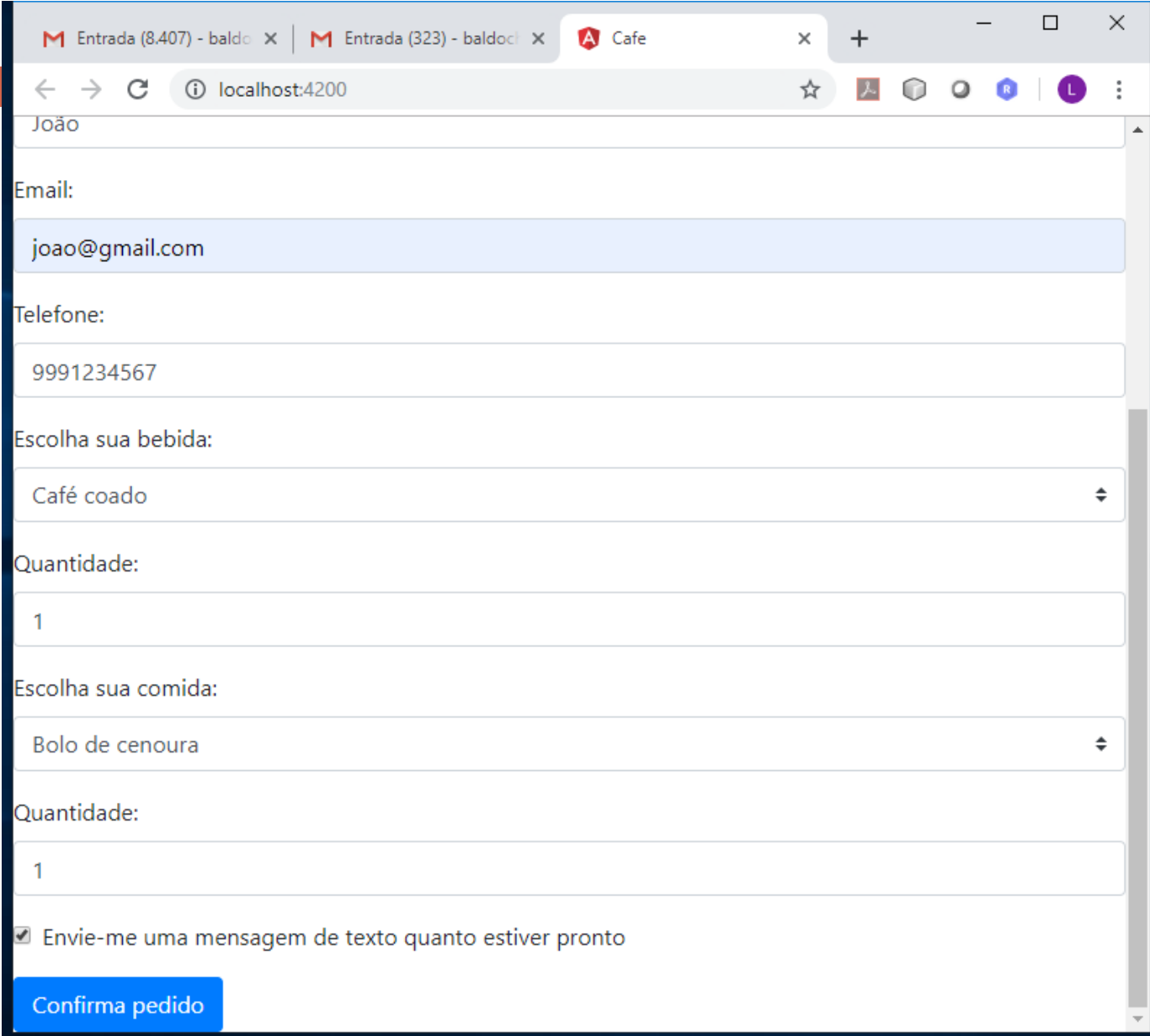
# Exercício 1

46

- Altere o exercício da Cafeteria
  - ▣ Retire a opção: quente/frio da bebida
  - ▣ Inclua o campo quantidade para a bebida
  - ▣ Inclua o campo comida
    - Bolo, pão de queijo, etc.
  - ▣ Inclua o campo quantidade para a comida
  - ▣ Ao pressionar o botão submit, informe o nome e o pedido do cliente no topo do formulário
- ▣ Importante: alterar o order.ts

# Exercício 1

47

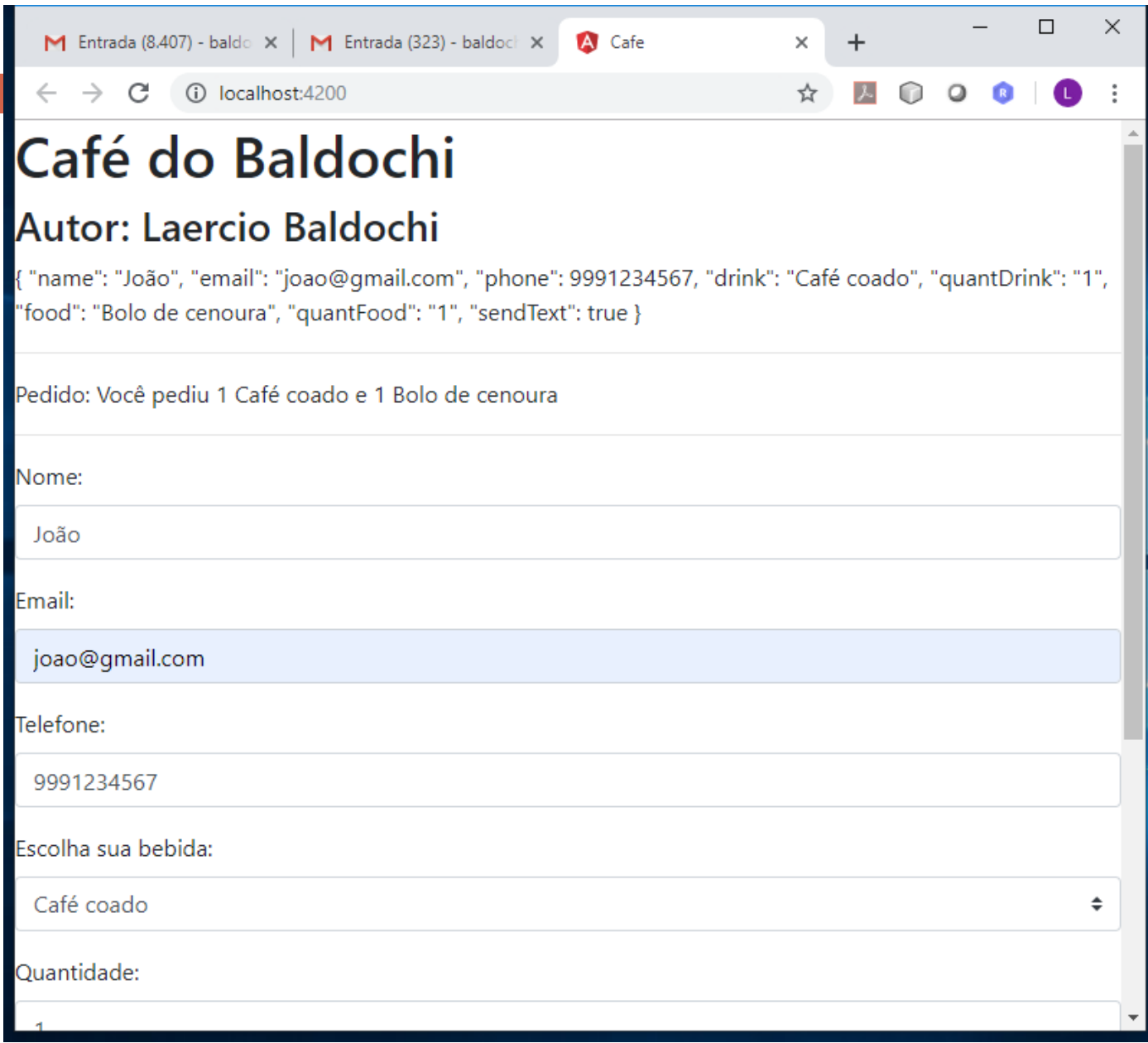


The screenshot shows a web browser window with three tabs: 'Entrada (8.407) - baldoc', 'Entrada (323) - baldoc', and 'Cafe'. The address bar shows 'localhost:4200'. The form contains the following fields and elements:

- Nome:** João
- Email:** joao@gmail.com
- Telefone:** 9991234567
- Escolha sua bebida:** Café coado
- Quantidade:** 1
- Escolha sua comida:** Bolo de cenoura
- Quantidade:** 1
- ☒ Envie-me uma mensagem de texto quando estiver pronto
- Botão:** Confirma pedido

# Exercício 1

48



The screenshot shows a web browser window with three tabs: 'Entrada (8.407) - baldoc...', 'Entrada (323) - baldoc...', and 'Cafe'. The address bar shows 'localhost:4200'. The page title is 'Café do Baldochi' and the author is 'Laercio Baldochi'. Below the title, there is a JSON object: 

```
{ "name": "João", "email": "joao@gmail.com", "phone": 9991234567, "drink": "Café coado", "quantDrink": "1", "food": "Bolo de cenoura", "quantFood": "1", "sendText": true }
```

. The form contains the following fields: 'Pedido: Você pediu 1 Café coado e 1 Bolo de cenoura', 'Nome:' with a text input containing 'João', 'Email:' with a text input containing 'joao@gmail.com', 'Telefone:' with a text input containing '9991234567', 'Escolha sua bebida:' with a dropdown menu showing 'Café coado', and 'Quantidade:' with a text input containing '1'.

Entrada (8.407) - baldoc x | Entrada (323) - baldoc x | Cafe x + - □ ×

← → ↻ ⓘ localhost:4200 ☆ 📄 🏠 🌐 📧

## Café do Baldochi

Autor: Laercio Baldochi

```
{ "name": "João", "email": "joao@gmail.com", "phone": 9991234567, "drink": "Café coado", "quantDrink": "1", "food": "Bolo de cenoura", "quantFood": "1", "sendText": true }
```

Pedido: Você pediu 1 Café coado e 1 Bolo de cenoura

Nome:

Email:

Telefone:

Escolha sua bebida:

Quantidade: