Desenvolvimento Web com Angular
Do Zero a Quase Hero

1. Introdução ao Desenvolvimento Web e Angular

1.1. A Evolução do Desenvolvimento Web: Uma Perspectiva Histórica

O desenvolvimento web emergiu como um dos principais campos da tecnologia da informação, acompanhando a evolução da internet desde os anos 90. O lançamento da World Wide Web em 1991 por Tim Berners-Lee inaugurou uma nova era de comunicação e acesso à informação. Inicialmente, as páginas web eram simples, baseadas em **HTML estático**, com pouca ou nenhuma interatividade. Essas primeiras páginas serviam como documentos informativos, sem qualquer capacidade de responder a interações complexas dos usuários.

Com o crescimento da internet, surgiu a necessidade de maior dinamismo e interatividade nas páginas. Nos anos 2000, tecnologias como **JavaScript**, **CSS**, e posteriormente o surgimento do **AJAX** permitiram que sites se tornassem mais responsivos e próximos de aplicações desktop, consolidando o modelo de aplicações ricas na web (RIA – Rich Internet Applications). Essa fase trouxe a ascensão de **frameworks como jQuery**, que simplificaram a manipulação do DOM e a comunicação assíncrona com servidores.

Em paralelo, o backend também evoluiu significativamente. Linguagens e frameworks voltados ao desenvolvimento do lado servidor, como **Java com Spring**, **PHP**, **ASP.NET**, e posteriormente **Node.js**, começaram a estruturar a comunicação entre os navegadores e os servidores, formando o modelo **cliente-servidor** que conhecemos hoje. Nessa trajetória, surgiram arquiteturas mais robustas, como **RESTful APIs**, que viabilizam a comunicação entre sistemas por meio de protocolos padronizados, consolidando o conceito de separação entre frontend e backend.

1.2. A Necessidade de Frameworks Modernos no Frontend

À medida que a web se tornou uma plataforma essencial para negócios, a demanda por interfaces mais dinâmicas e escaláveis aumentou. Surgiram desafios relacionados à complexidade do desenvolvimento frontend, como a necessidade de gerenciar o estado da aplicação, otimizar o desempenho e garantir a responsividade em diferentes dispositivos. Nesse cenário, frameworks como **AngularJS** (lançado em 2010), **React** (2013) e **Vue.js** (2014) ganharam espaço, introduzindo novos paradigmas de desenvolvimento.

O **AngularJS** foi um dos primeiros frameworks amplamente adotados por sua abordagem declarativa, utilizando a estrutura **Model-View-Controller (MVC)**. Contudo, com o crescimento e a mudança das necessidades do mercado, o AngularJS enfrentou dificuldades de manutenção em projetos mais complexos. A fim de superar essas limitações, a equipe do Google lançou, em 2016, uma reescrita completa do framework: o **Angular 2**, posteriormente rebatizado apenas como **Angular**. Desde então, o Angular tem evoluído constantemente, até sua versão atual, o **Angular 15**, que oferece maior performance e uma experiência de desenvolvimento simplificada.

1.3. Características do Angular como Framework

O **Angular** é um framework de código aberto mantido pelo Google, projetado para facilitar a criação de **Single Page Applications (SPAs)**, um tipo de aplicação em que o conteúdo é carregado dinamicamente sem a necessidade de recarregar a página inteira. Angular é um framework baseado em **TypeScript**, uma superset do JavaScript que adiciona tipagem estática, tornando o código mais seguro e facilitando a manutenção em projetos de larga escala.

O **Angular CLI (Command Line Interface)** simplifica a criação e configuração de novos projetos, proporcionando um ambiente pronto para desenvolvimento. Além disso, o Angular é conhecido por seu uso de **componentes** e **módulos**, permitindo a modularização do código e a reutilização de funcionalidades.

Um dos aspectos mais poderosos do Angular é sua arquitetura orientada a serviços, que promove a separação entre a lógica de negócios e a interface. Por meio de **Services** e **Dependency Injection (DI)**, o Angular facilita a construção de aplicações escaláveis e manuteníveis, promovendo boas práticas de design.

Além disso, o framework possui um ecossistema robusto que inclui suporte nativo para **roteamento**, **formulários reativos**, e **teste automatizado** com ferramentas como Karma e Jasmine. A integração com bibliotecas de estilos como o **Bootstrap** amplia as possibilidades de personalização da interface, permitindo criar aplicações ricas e responsivas.

1.4. Angular no Contexto Atual do Desenvolvimento Web

O Angular se destaca entre os frameworks modernos por oferecer uma solução completa para o desenvolvimento de SPAs. Enquanto frameworks como **React** se concentram apenas na camada de visualização (View), o Angular adota uma abordagem de “bateria incluída”, fornecendo uma estrutura completa que contempla roteamento, gerenciamento de estado e testes.

No contexto atual, a adoção de frameworks como Angular responde a uma necessidade crescente por soluções que sejam tanto **performáticas** quanto **fáceis de**

escalar. Empresas e desenvolvedores buscam frameworks que possibilitem a construção de aplicações complexas com equipes distribuídas, e o Angular tem demonstrado ser uma escolha sólida para projetos corporativos e governamentais devido à sua robustez e suporte a longo prazo.

A **versão 15 do Angular** introduziu melhorias significativas em termos de performance, simplificando a configuração de projetos e otimizando a renderização dos componentes. Além disso, a nova versão foca em garantir maior compatibilidade com bibliotecas externas e na redução do tempo de build, fatores essenciais para projetos de grande escala.

1.5. Considerações finais

O desenvolvimento web, desde seus primórdios até o estado atual, passou por uma transformação significativa, movendo-se de páginas estáticas para aplicações interativas e dinâmicas. Dentro desse contexto, o **Angular** consolidou-se como uma ferramenta essencial para o desenvolvimento de SPAs robustas e escaláveis. Seu ecossistema integrado, aliado à evolução contínua pela comunidade e pelo Google, faz do Angular uma escolha estratégica para desenvolvedores e empresas que buscam qualidade, desempenho e manutenção simplificada em projetos modernos.

A compreensão dos conceitos históricos e técnicos do desenvolvimento web e do Angular é fundamental para que futuros profissionais possam tomar decisões informadas e criar soluções que atendam às demandas contemporâneas do mercado. Ao longo desta apostila, exploraremos como utilizar o Angular para construir aplicações funcionais, eficientes e alinhadas às melhores práticas da engenharia de software, preparando os alunos para atuarem de forma competitiva no mercado de trabalho.

2. Protótipo Inicial com HTML e Bootstrap

2.1. Introdução

Antes de integrar o frontend com o backend, é essencial construir um **protótipo inicial** que defina a estrutura visual da aplicação. O objetivo deste capítulo é explorar como utilizar **HTML** e **Bootstrap** para criar as primeiras páginas do projeto "Receitas da Mamãe". Um protótipo permite validar o layout e a experiência do usuário antes do desenvolvimento completo, evitando retrabalho nas fases seguintes. A aplicação terá um design simples e agradável, com uma **paleta de cores inspirada em ambientes culinários**, utilizando tons terrosos e suaves, remetendo a conforto e tradição.

2.2. Tecnologias Utilizadas

HTML (HyperText Markup Language)

O HTML é uma **linguagem de marcação** usada para estruturar o conteúdo de uma página web. Ele define elementos como cabeçalhos, parágrafos, botões, formulários e links. No protótipo, o HTML será utilizado para criar a base de cada página, organizando o conteúdo em seções lógicas.

Bootstrap

O **Bootstrap** é um framework CSS que facilita a criação de **layouts responsivos** e esteticamente agradáveis. Ele oferece uma variedade de componentes prontos, como formulários, botões, barras de navegação e grids, otimizando o tempo de desenvolvimento. Utilizaremos o Bootstrap para aplicar **responsividade** e **estilo** ao protótipo, garantindo uma boa apresentação em diversos dispositivos.

Paleta de Cores

Para remeter à temática culinária, a aplicação terá uma paleta baseada em:

- **Marrom claro** (#A0522D) e **bege** (#F5DEB3), sugerindo elementos como madeira e farinha.
- **Verde oliva** (#556B2F), que remete a frescor e alimentos saudáveis.
- **Laranja suave** (#FFB347), inspirando-se em alimentos assados e acolhedores.

2.3. Estrutura das Páginas HTML

A seguir, apresentamos o código de três páginas HTML que compõem o protótipo da aplicação. Elas incluem:

1. **Página Inicial**: Apresenta a aplicação e convida o usuário a explorar as receitas.

2. **Página de Cadastro/Atualização**: Permite adicionar ou atualizar uma receita (ainda sem backend).

3. **Página de Lista de Receitas**: Exibe as receitas cadastradas.

2.3.1. Página Inicial – index.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Receitas da Mamãe</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.c
ss">
</head>
<body style="background-color: #F5DEB3;">
    <nav class="navbar navbar-expand-lg navbar-dark" style="background-
color: #A0522D;">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Receitas da Mamãe</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ms-auto">
                    <li class="nav-item"><a class="nav-link active"
href="#">Início</a></li>
                    <li class="nav-item"><a class="nav-link"
href="cadastro.html">Cadastrar Receita</a></li>
                    <li class="nav-item"><a class="nav-link"
href="lista.html">Lista de Receitas</a></li>
                </ul>
            </div>
        </div>
    </nav>

    <div class="container text-center mt-5">
        <h1 class="display-4" style="color: #A0522D;">Bem-vindo às Receitas
da Mamãe</h1>
        <p class="lead" style="color: #556B2F;">Explore, cadastre e
compartilhe suas receitas favoritas!</p>
        <a href="lista.html" class="btn btn-lg" style="background-color:
#FFB347; color: white;">Ver Receitas</a>
    </div>
</body>
</html>
```

2.3.2. Página de Cadastro e Atualização – cadastro.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cadastrar Receita</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.c
ss">
</head>
<body style="background-color: #FFB347;">
    <div class="container mt-5">
        <h2 style="color: white; text-align: center;">Cadastrar ou
Atualizar Receita</h2>
        <form class="mt-4">
            <div class="mb-3">
                <label for="titulo" class="form-label">Título da
Receita</label>
                <input type="text" class="form-control" id="titulo"
placeholder="Ex: Bolo de Cenoura">
            </div>
            <div class="mb-3">
                <label for="ingredientes" class="form-
label">Ingredientes</label>
                <textarea class="form-control" id="ingredientes" rows="3"
placeholder="Ex: 2 xícaras de farinha..."></textarea>
            </div>
            <div class="mb-3">
                <label for="modoPreparo" class="form-label">Modo de
Preparo</label>
                <textarea class="form-control" id="modoPreparo" rows="3"
placeholder="Descreva o modo de preparo"></textarea>
            </div>
            <button type="submit" class="btn" style="background-color:
#556B2F; color: white;">Salvar Receita</button>
        </form>
    </div>
</body>
</html>
```

2.3.3. Página de Lista de Receitas – lista.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista de Receitas</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.c
ss">
</head>
<body style="background-color: #F5DEB3;">
    <div class="container mt-5">
        <h2 class="text-center" style="color: #A0522D;">Receitas
Cadastradas</h2>
        <ul class="list-group mt-4">
            <li class="list-group-item">Bolo de Cenoura <span class="badge
bg-secondary">Doce</span></li>
            <li class="list-group-item">Pão de Queijo <span class="badge
bg-secondary">Salgado</span></li>
            <li class="list-group-item">Suco de Laranja <span class="badge
bg-secondary">Bebida</span></li>
        </ul>
    </div>
</body>
</html>
```

2.4. Considerações finais

Neste capítulo, desenvolvemos um **protótipo inicial** da aplicação "Receitas da Mamãe", utilizando **HTML e Bootstrap** para criar uma interface simples e intuitiva. Este protótipo é essencial para validar o design e a estrutura das páginas antes da integração com o backend em Java. A paleta de cores culinária reforça o tema e cria uma atmosfera aconchegante para os usuários. Nas próximas etapas, trabalharemos na integração desse frontend com o backend, transformando o protótipo em uma aplicação funcional e completa.

3. Configuração do Projeto Angular

3.1. Introdução ao Angular

O **Angular** é um framework desenvolvido pelo Google, utilizado para criar **Single Page Applications (SPA)**, onde o conteúdo é carregado dinamicamente, proporcionando uma experiência de navegação rápida e fluida. Escrito em **TypeScript**, uma superset de JavaScript que adiciona tipagem estática ao código, o Angular oferece uma estrutura modular e componentes reutilizáveis, facilitando a manutenção e escalabilidade de projetos.

Com uma arquitetura robusta baseada em componentes, serviços e injeção de dependências, o Angular se consolidou como uma escolha popular para o desenvolvimento de aplicações web. Além disso, ele possui um ecossistema abrangente, com ferramentas como **Angular CLI** para automação de tarefas, **RxJS** para programação reativa e suporte nativo para **roteamento** e **formulários reativos**.

3.2. Estrutura de Pastas no Angular

A organização da estrutura de pastas no Angular é essencial para manter o projeto bem estruturado, garantindo que cada funcionalidade esteja organizada de forma modular. Ao criar um projeto Angular, você encontrará a seguinte estrutura básica:

```
/src
  ├── /app               # Código principal da aplicação (módulos e componentes)
  │    ├── /components   # Componentes da interface
  │    ├── /services     # Serviços (lógica de negócio e comunicação com backend)
  │    └── app.module.ts # Módulo principal da aplicação
  ├── /assets            # Arquivos estáticos (imagens, ícones, etc.)
  ├── index.html         # Página principal da aplicação
  └── styles.css         # Estilos globais
```

Pastas principais:

- **src/app**: Contém a lógica central da aplicação, organizada em **módulos** e **componentes**.
- **src/assets**: Armazena arquivos estáticos, como imagens e fontes.
- **app.module.ts**: O módulo principal, responsável por definir os componentes e módulos importados para a aplicação.

A modularização é fundamental no Angular. Cada funcionalidade é dividida em **módulos** e **componentes**, permitindo que as partes da aplicação sejam desenvolvidas e mantidas de forma independente.

3.3. Criação e Configuração do Projeto Angular

3.3.1. Instalação do Angular CLI

O Angular CLI é uma ferramenta oficial para criação e gerenciamento de projetos Angular, simplificando tarefas como a criação de componentes e a configuração de módulos.

Para instalar o Angular CLI, utilize o seguinte comando:

```
npm install -g @angular/cli
```

Verifique se a instalação foi bem-sucedida executando:

```
ng version
```

3.3.2. Criação do Projeto Angular

Para criar um novo projeto, execute o seguinte comando:

```
ng new receitas-mamae
```

Durante a criação, o CLI solicitará algumas configurações, como adicionar roteamento e escolher o formato de estilos (CSS, SCSS, etc.). Selecione:

- **Roteamento**: Sim
- **Estilo**: CSS

Navegue até a pasta do projeto:

```
cd receitas-mamae
```

Inicie o servidor de desenvolvimento com:

```
ng serve
```

A aplicação estará disponível em **http://localhost:4200**.

3.4. Instalação do Bootstrap no Projeto

Para adicionar o **Bootstrap** e utilizar seus componentes na interface, siga os passos abaixo.

1. Instale o Bootstrap via **npm**:

```
npm install bootstrap
```

2. No arquivo angular.json, adicione o CSS do Bootstrap em **"styles"**:

```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
]
```

3. Reinicie o servidor Angular:

```
ng serve
```

3.5. Criação dos Componentes

Com a estrutura básica pronta, o próximo passo é criar os **componentes** necessários para a aplicação, baseando-se no protótipo desenvolvido anteriormente.

Componente Inicial (Home)

Crie um componente para a página inicial:

```
ng generate component components/home
```

Componente de Cadastro/Atualização

Crie um componente para o formulário de receitas:

```
ng generate component components/cadastro-receita
```

Componente de Lista de Receitas

Crie um componente para exibir a lista de receitas:

```
ng generate component components/lista-receitas
```

3.6. Configuração dos Componentes

Modificando o Roteamento

No Angular, a navegação entre páginas é configurada utilizando o **RouterModule**. Edite o arquivo `app-routing.module.ts` para definir as rotas dos componentes criados:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './components/home/home.component';
import { CadastroReceitaComponent } from './components/cadastro-
receita/cadastro-receita.component';
import { ListaReceitasComponent } from './components/lista-receitas/lista-
receitas.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'cadastro', component: CadastroReceitaComponent },
  { path: 'lista', component: ListaReceitasComponent }
];

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

Modificando os Componentes

1. **HomeComponent** (`home.component.html`):

```
<div class="container text-center mt-5">
  <h1 class="display-4" style="color: #A0522D;">Bem-vindo às
Receitas da Mamãe</h1>
  <p class="lead" style="color: #556B2F;">Explore, cadastre e
compartilhe suas receitas favoritas!</p>
  <a routerLink="/lista" class="btn btn-lg" style="background-color:
#FFB347; color: white;">Ver Receitas</a>
</div>
```

2. **CadastroReceitaComponent** (`cadastro-receita.component.html`):

```
<div class="container mt-5">
  <h2 class="text-center" style="color: white;">Cadastrar ou
Atualizar Receita</h2>
  <form>
    <div class="mb-3">
      <label for="titulo" class="form-label">Título da
Receita</label>
      <input type="text" class="form-control" id="titulo"
placeholder="Ex: Bolo de Cenoura">
    </div>
    <div class="mb-3">
      <label for="ingredientes" class="form-
label">Ingredientes</label>
      <textarea class="form-control" id="ingredientes"
rows="3"></textarea>
    </div>
    <div class="mb-3">
      <label for="modoPreparo" class="form-label">Modo de
Preparo</label>
      <textarea class="form-control" id="modoPreparo"
rows="3"></textarea>
    </div>
    <button type="submit" class="btn" style="background-color:
#556B2F; color: white;">Salvar</button>
  </form>
</div>
```

3. **ListaReceitasComponent** (`lista-receitas.component.html`):

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Receitas
Cadastradas</h2>
  <ul class="list-group mt-4">
    <li class="list-group-item">Bolo de Cenoura <span class="badge
bg-secondary">Doce</span></li>
    <li class="list-group-item">Pão de Queijo <span class="badge bg-
secondary">Salgado</span></li>
    <li class="list-group-item">Suco de Laranja <span class="badge
bg-secondary">Bebida</span></li>
  </ul>
</div>
```

3.7. Considerações finais

Com o projeto Angular configurado e os componentes essenciais criados, estamos prontos para desenvolver as funcionalidades completas da aplicação "Receitas da Mamãe". A organização modular facilita a manutenção e evolução do projeto, enquanto o uso do Bootstrap garante uma interface amigável e responsiva. No próximo capítulo, trabalharemos na integração com o backend, transformando o protótipo em uma aplicação completa e funcional.

4. Serviços Angular para Consumo da API

4.1. Introdução

No Angular, os **serviços** são fundamentais para lidar com a lógica de negócios e comunicação com APIs externas. Um serviço é uma classe que pode ser **injetada em outros componentes** utilizando o sistema de **injeção de dependência (DI)** do Angular, garantindo que a aplicação esteja bem organizada e mantenha o princípio de separação de responsabilidades. Neste capítulo, desenvolveremos um serviço em Angular para **consumir os endpoints** da API "Receitas da Mamãe", implementada em Java.

4.2. Criação do Serviço para Consumo da API

Vamos criar o **ReceitasService**, que será responsável por se comunicar com o backend Java por meio dos **endpoints REST** descritos anteriormente. Utilizaremos o Angular **HttpClient** para realizar as requisições HTTP.

4.2.1. Configuração do HttpClientModule

Primeiro, precisamos garantir que o **HttpClientModule** esteja importado no módulo principal da aplicação `app.module.ts`.

```
// src/app/app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http'; // Importação do
HttpClientModule
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

// Importação dos componentes
import { HomeComponent } from './components/home/home.component';
import { CadastroReceitaComponent } from './components/cadastro-receita/cadastro-
receita.component';
import { ListaReceitasComponent } from './components/lista-receitas/lista-
receitas.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    CadastroReceitaComponent,
    ListaReceitasComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule // HttpClientModule adicionado corretamente no imports
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

4.2.2. Criação do Serviço ReceitasService

Com o **HttpClientModule** configurado, vamos criar o serviço que irá realizar as operações de **CRUD** e o consumo dos dados.

```
ng generate service services/receitas
```

Código do Serviço – `receitas.service.ts`

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Receitas } from '../models/receitas';

@Injectable({
  providedIn: 'root'
})
export class ReceitasService {
  private apiUrl = 'http://localhost:8080/api/receitas';

  constructor(private http: HttpClient) {}

  // Headers padrão para as requisições
  private headers = new HttpHeaders({ 'Content-Type': 'application/json'
});

  // Criar uma nova receita
  saveReceita(receita: Receitas): Observable<Receitas> {
    return this.http.post<Receitas>(this.apiUrl, receita, { headers:
this.headers });
  }

  // Listar todas as receitas
  findAllReceitas(): Observable<Receitas[]> {
    return this.http.get<Receitas[]>(this.apiUrl);
  }

  // Buscar receita por ID
  findReceitaById(id: number): Observable<Receitas> {
    return this.http.get<Receitas>(`${this.apiUrl}/${id}`);
  }

  // Atualizar receita existente
  updateReceita(receita: Receitas): Observable<Receitas> {
    return this.http.put<Receitas>(this.apiUrl, receita, { headers:
this.headers });
  }

  // Deletar receita por ID
  deleteReceita(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }
```

```
  // Buscar receitas por tipo
  findReceitasByTipo(tipo: string): Observable<Receitas[]> {
    return this.http.get<Receitas[]>(`${this.apiUrl}/tipo/${tipo}`);
  }

  // Listar tipos de receitas disponíveis
  getTiposReceitas(): Observable<string[]> {
    return this.http.get<string[]>(`${this.apiUrl}/tipos`);
  }
}
```

4.3. Implementação nos Componentes

Agora que temos o serviço ReceitasService, vamos integrá-lo aos **componentes Angular** para que cada um realize a operação correspondente ao seu objetivo.

HomeComponent – Listar Receitas na Página Inicial

```
import { Component, OnInit } from '@angular/core';
import { ReceitasService } from '../../services/receitas.service';
import { Receitas } from '../../models/receitas';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
})
export class HomeComponent implements OnInit {
  receitas: Receitas[] = [];

  constructor(private receitasService: ReceitasService) {}

  ngOnInit(): void {
    this.receitasService.findAllReceitas().subscribe((data) => {
      this.receitas = data;
    });
  }
}
```

HTML do **HomeComponent** (`home.component.html`):

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Receitas Recentes</h2>
  <ul class="list-group mt-4">
    <li *ngFor="let receita of receitas" class="list-group-item">
      {{ receita.titulo }} <span class="badge bg-secondary">{{
receita.tipoReceita }}</span>
    </li>
  </ul>
</div>
```

CadastroReceitaComponent – Salvar e Atualizar Receitas

```
import { Component } from '@angular/core';
import { ReceitasService } from '../../services/receitas.service';
import { Receitas } from '../../models/receitas';

@Component({
  selector: 'app-cadastro-receita',
  templateUrl: './cadastro-receita.component.html',
})
export class CadastroReceitaComponent {
  receita: Receitas = new Receitas();

  constructor(private receitasService: ReceitasService) {}

  salvarReceita(): void {
    this.receitasService.saveReceita(this.receita).subscribe(() => {
      alert('Receita salva com sucesso!');
    });
  }
}
```

HTML do **CadastroReceitaComponent** (`cadastro-receita.component.html`):

```
<form class="container mt-5">
  <div class="mb-3">
    <label for="titulo" class="form-label">Título</label>
    <input [(ngModel)]="receita.titulo" type="text" class="form-control"
id="titulo" />
  </div>
  <div class="mb-3">
    <label for="tipo" class="form-label">Tipo</label>
    <input [(ngModel)]="receita.tipoReceita" type="text" class="form-
control" id="tipo" />
  </div>
  <button (click)="salvarReceita()" class="btn btn-success">Salvar</button>
</form>
```

ListaReceitasComponent – Exibir Receitas Cadastradas

```
import { Component, OnInit } from '@angular/core';
import { ReceitasService } from '../../services/receitas.service';
import { Receitas } from '../../models/receitas';

@Component({
  selector: 'app-lista-receitas',
  templateUrl: './lista-receitas.component.html',
})
export class ListaReceitasComponent implements OnInit {
  receitas: Receitas[] = [];

  constructor(private receitasService: ReceitasService) {}

  ngOnInit(): void {
    this.receitasService.findAllReceitas().subscribe((data) => {
      this.receitas = data;
    });
  }

  deletarReceita(id: number): void {
    this.receitasService.deleteReceita(id).subscribe(() => {
      this.receitas = this.receitas.filter((r) => r.id !== id);
    });
  }
}
```

HTML do **ListaReceitasComponent** (`lista-receitas.component.html`):

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Receitas Cadastradas</h2>
  <ul class="list-group mt-4">
    <li *ngFor="let receita of receitas" class="list-group-item">
      {{ receita.titulo }}
      <button (click)="deletarReceita(receita.id)" class="btn btn-danger
btn-sm float-end">Excluir</button>
    </li>
  </ul>
</div>
```

4.4. Considerações finais

Neste capítulo, configuramos o **serviço Angular** para consumir a API "Receitas da Mamãe" e implementamos a integração dos serviços com os componentes. A separação entre **serviços e componentes** garante uma arquitetura limpa e organizada, seguindo as melhores práticas do desenvolvimento Angular. O **ReceitasService** centraliza toda a lógica de comunicação com o backend, enquanto os componentes são responsáveis pela exibição dos dados e pela interação com o usuário.

Com essa estrutura, a aplicação está pronta para se comunicar com o backend Java, consumindo dados de maneira eficiente e responsiva. No próximo capítulo, exploraremos como **validar formulários** e **lidar com erros** nas requisições HTTP, aprimorando ainda mais a experiência do usuário.

5. Implementando a Interface de Usuário

5.1. Introdução

A interface de usuário (UI) é a porta de entrada para a interação do usuário com a aplicação. No desenvolvimento de aplicações web modernas, garantir que a interface seja **amistosa, intuitiva** e **responsiva** é fundamental para oferecer uma boa experiência ao usuário. Neste capítulo, implementaremos a interface da aplicação "Receitas da Mamãe" utilizando **Angular**, **Bootstrap** e **Font Awesome** para incluir ícones nos menus. A paleta de cores culinária será mantida, utilizando tons terrosos e suaves para reforçar a temática acolhedora.

5.2. Preparação do Projeto

5.2.1. Instalação do Font Awesome

Para adicionar ícones à interface, utilizaremos a biblioteca **Font Awesome**. Execute o seguinte comando para instalar a biblioteca no projeto:

```
npm install @fortawesome/fontawesome-free
```

Em seguida, adicione a referência ao CSS do Font Awesome no arquivo **angular.json**:

```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "node_modules/@fortawesome/fontawesome-free/css/all.min.css"
]
```

5.3. Ajuste da Interface com Componentes Angular

5.3.1. Menu de Navegação Global

Criamos um **menu de navegação** com links para as páginas principais: Home, Cadastro e Lista de Receitas. Utilizamos ícones do **Font Awesome** para tornar o menu mais amigável.

Código HTML – Menu de Navegação (`app.component.html`)

```
<nav class="navbar navbar-expand-lg" style="background-color: #A0522D;">
  <div class="container-fluid">
    <a class="navbar-brand text-white" href="#">Receitas da Mamãe</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav ms-auto">
        <li class="nav-item">
          <a class="nav-link text-white" routerLink="/">
            <i class="fas fa-home"></i> Início
          </a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" routerLink="/cadastro">
            <i class="fas fa-edit"></i> Cadastrar Receita
          </a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" routerLink="/lista">
            <i class="fas fa-list"></i> Lista de Receitas
          </a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

5.3.2. Página Inicial (HomeComponent)

Código HTML – HomeComponent (`home.component.html`)

```
<div class="container text-center mt-5">
  <h1 class="display-4" style="color: #A0522D;">Bem-vindo às Receitas da
Mamãe</h1>
  <p class="lead" style="color: #556B2F;">
    Descubra receitas deliciosas e cadastre as suas próprias!
  </p>
  <a routerLink="/lista" class="btn btn-lg" style="background-color:
#FFB347; color: white;">
    <i class="fas fa-utensils"></i> Ver Receitas
  </a>
</div>
```

5.3.3. Página de Cadastro de Receitas (CadastroReceitaComponent)

HTML – CadastroReceitaComponent (`cadastro-receita.component.html`)

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Cadastrar Nova
Receita</h2>
  <form class="mt-4">
    <div class="mb-3">
      <label for="titulo" class="form-label">Título da Receita</label>
      <input [(ngModel)]="receita.titulo" type="text" class="form-control"
id="titulo" placeholder="Ex: Bolo de Cenoura" />
    </div>
    <div class="mb-3">
      <label for="tipo" class="form-label">Tipo de Receita</label>
      <select [(ngModel)]="receita.tipoReceita" class="form-select"
id="tipo">
        <option value="Doce">Doce</option>
        <option value="Salgado">Salgado</option>
        <option value="Bebida">Bebida</option>
        <option value="Outros">Outros</option>
      </select>
    </div>
    <div class="mb-3">
      <label for="ingredientes" class="form-label">Ingredientes</label>
      <textarea [(ngModel)]="receita.ingredientes" class="form-control"
id="ingredientes" rows="3"></textarea>
    </div>
    <button (click)="salvarReceita()" class="btn" style="background-color:
#556B2F; color: white;">
      <i class="fas fa-save"></i> Salvar Receita
    </button>
  </form>
</div>
```

TypeScript – CadastroReceitaComponent (`cadastro-receita.component.ts`)

```
import { Component } from '@angular/core';
import { ReceitasService } from '../../services/receitas.service';
import { Receitas } from '../../models/receitas';

@Component({
  selector: 'app-cadastro-receita',
  templateUrl: './cadastro-receita.component.html',
})
export class CadastroReceitaComponent {
  receita: Receitas = new Receitas();

  constructor(private receitasService: ReceitasService) {}

  salvarReceita(): void {
    this.receitasService.saveReceita(this.receita).subscribe(() => {
      alert('Receita salva com sucesso!');
```

```
    });
  }
}
```

5.3.4. Página de Lista de Receitas (ListaReceitasComponent)

HTML – ListaReceitasComponent (`lista-receitas.component.html`)

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Lista de Receitas</h2>
  <ul class="list-group mt-4">
    <li *ngFor="let receita of receitas" class="list-group-item d-flex
justify-content-between align-items-center">
      {{ receita.titulo }}
      <span class="badge bg-secondary">{{ receita.tipoReceita }}</span>
      <button (click)="deletarReceita(receita.id)" class="btn btn-sm btn-
danger">
        <i class="fas fa-trash-alt"></i>
      </button>
    </li>
  </ul>
</div>
```

TypeScript – ListaReceitasComponent (`lista-receitas.component.ts`)

```
import { Component, OnInit } from '@angular/core';
import { ReceitasService } from '../../services/receitas.service';
import { Receitas } from '../../models/receitas';

@Component({
  selector: 'app-lista-receitas',
  templateUrl: './lista-receitas.component.html',
})
export class ListaReceitasComponent implements OnInit {
  receitas: Receitas[] = [];

  constructor(private receitasService: ReceitasService) {}

  ngOnInit(): void {
    this.receitasService.findAllReceitas().subscribe((data) => {
      this.receitas = data;
    });
  }

  deletarReceita(id: number): void {
    this.receitasService.deleteReceita(id).subscribe(() => {
      this.receitas = this.receitas.filter((r) => r.id !== id);
    });
  }
}
```

5.4. Considerações finais

Com a implementação da interface de usuário concluída, garantimos que a aplicação "Receitas da Mamãe" tenha uma **navegação intuitiva** e um **design consistente** com a temática culinária. O uso de **Bootstrap** e **Font Awesome** aprimorou a usabilidade, tornando a aplicação mais agradável e moderna. No próximo capítulo, abordaremos como validar os formulários e lidar com erros nas requisições para oferecer uma experiência ainda melhor ao usuário.

6. Estilizando com Bootstrap: Layout de Receitas

6.1. Introdução

O **design de interface** é essencial para proporcionar uma experiência agradável e intuitiva ao usuário. Neste capítulo, aplicaremos o **Bootstrap**, um framework CSS popular, para garantir que a interface da aplicação "Receitas da Mamãe" seja responsiva, esteticamente agradável e consistente com a **paleta de cores culinária** apresentada. Adicionaremos **ícones do Font Awesome** para enriquecer o layout e melhorar a navegação, garantindo que as funcionalidades fiquem claras e fáceis de acessar.

6.2. Estrutura Responsiva com Bootstrap

O Bootstrap oferece um sistema de **grid** que facilita a construção de layouts responsivos, adaptando-se automaticamente para diferentes tamanhos de tela. Utilizaremos esse recurso para garantir que a aplicação funcione bem em **dispositivos móveis e desktops**.

Adição de Container e Grid

```
<div class="container mt-5">
  <div class="row">
    <div class="col-md-6">
      <h1 class="display-5" style="color: #A0522D;">Receitas da Mamãe</h1>
      <p class="lead" style="color: #556B2F;">Explore receitas deliciosas e
inspire-se na cozinha!</p>
    </div>
    <div class="col-md-6 text-center">
      <img src="assets/cooking.png" class="img-fluid" alt="Cozinhando" />
    </div>
  </div>
</div>
```

Explicação

- `container`: Centraliza o conteúdo e define margens automáticas.
- `row` e `col-md-6`: Utilizamos o sistema de grid para dividir a área em duas colunas de igual tamanho em telas médias e maiores.

6.3. Estilizando o Menu de Navegação

Aqui adicionaremos **ícones no menu de navegação** para torná-lo mais claro e moderno.

Código HTML – Menu de Navegação (`app.component.html`)

```
<nav class="navbar navbar-expand-lg" style="background-color: #A0522D;">
  <div class="container-fluid">
    <a class="navbar-brand text-white" href="#">
      <i class="fas fa-book-open"></i> Receitas da Mamãe
    </a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav ms-auto">
        <li class="nav-item">
          <a class="nav-link text-white" routerLink="/">
            <i class="fas fa-home"></i> Início
          </a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" routerLink="/cadastro">
            <i class="fas fa-plus-circle"></i> Nova Receita
          </a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" routerLink="/lista">
            <i class="fas fa-list"></i> Lista de Receitas
          </a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

6.4. Estilizando a Página de Lista de Receitas

Adicionaremos **cartões estilizados** para exibir as receitas de forma organizada e agradável.

HTML – ListaReceitasComponent (`lista-receitas.component.html`)

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Receitas Cadastradas</h2>
  <div class="row mt-4">
    <div *ngFor="let receita of receitas" class="col-md-4">
      <div class="card mb-4" style="border-color: #FFB347;">
        <div class="card-body">
          <h5 class="card-title">{{ receita.titulo }}</h5>
          <p class="card-text">
            <strong>Tipo:</strong> {{ receita.tipoReceita }}
          </p>
          <button (click)="deletarReceita(receita.id)" class="btn btn-sm
btn-danger">
```

```
            <i class="fas fa-trash-alt"></i> Excluir
          </button>
        </div>
      </div>
    </div>
  </div>
</div>
```

Explicação

- `card`: Utilizamos o componente **card do Bootstrap** para exibir as receitas.
- `col-md-4`: Cada receita é exibida em uma coluna, e em telas menores, os cartões são empilhados.

6.5. Estilizando a Página de Cadastro de Receitas

Utilizaremos **formularios do Bootstrap** com **feedbacks visuais** para melhorar a usabilidade.

HTML – CadastroReceitaComponent (`cadastro-receita.component.html`)

```
<div class="container mt-5">
  <h2 class="text-center" style="color: #A0522D;">Cadastrar Nova
Receita</h2>
  <form class="mt-4">
    <div class="mb-3">
      <label for="titulo" class="form-label">Título da Receita</label>
      <input [(ngModel)]="receita.titulo" type="text" class="form-control"
id="titulo" placeholder="Ex: Bolo de Cenoura" />
    </div>
    <div class="mb-3">
      <label for="tipo" class="form-label">Tipo de Receita</label>
      <select [(ngModel)]="receita.tipoReceita" class="form-select"
id="tipo">
        <option value="Doce">Doce</option>
        <option value="Salgado">Salgado</option>
        <option value="Bebida">Bebida</option>
        <option value="Outros">Outros</option>
      </select>
    </div>
    <div class="mb-3">
      <label for="ingredientes" class="form-label">Ingredientes</label>
      <textarea [(ngModel)]="receita.ingredientes" class="form-control"
id="ingredientes" rows="3"></textarea>
    </div>
    <button (click)="salvarReceita()" class="btn btn-success">
      <i class="fas fa-save"></i> Salvar Receita
    </button>
  </form>
```

```
</div>
```

6.6. Considerações finais

A aplicação "Receitas da Mamãe" foi estilizada utilizando **Bootstrap** e **Font Awesome**, proporcionando um layout moderno, funcional e agradável. As páginas estão organizadas de maneira responsiva, com **cartões** e **menus intuitivos** que facilitam a navegação e a interação do usuário. Essa estrutura visual consistente é essencial para garantir uma boa experiência, independentemente do dispositivo utilizado.

7. Testes e Execução Final

7.1. Introdução

Nesta etapa, realizaremos **testes** na aplicação "Receitas da Mamãe" para garantir que todas as funcionalidades implementadas estão funcionando corretamente e que não há erros nas interações entre o frontend Angular e o backend em Java. A fase de testes é essencial para validar a lógica de negócios, garantir que os **endpoints REST** respondam como esperado e assegurar uma **boa experiência de usuário**. Além disso, sugeriremos melhorias e abordaremos possíveis evoluções para o projeto em um subcapítulo final.

7.2. Tipos de Testes Realizados

1. **Testes de Unidade**: Verificam o funcionamento de partes isoladas da aplicação, como serviços e métodos.
2. **Testes de Integração**: Validam a comunicação entre os componentes e a API.
3. **Testes de Interface (UI)**: Avaliam a aparência e a usabilidade do sistema, garantindo que a interface esteja funcional e amigável.

7.3. Configuração do Ambiente de Testes

Para realizar testes em Angular, utilizaremos as ferramentas **Karma** e **Jasmine**, que já vêm pré-configuradas ao criar um projeto Angular com o **Angular CLI**.

7.3.1. Comando para Rodar os Testes

Para rodar todos os testes na aplicação, basta executar o seguinte comando:

```
ng test
```

Isso abrirá o navegador com o **Karma Test Runner**, mostrando os resultados dos testes em tempo real.

7.4. Implementação dos Testes

7.4.1. Teste de Unidade – ReceitasService

Este teste verifica se o serviço **ReceitasService** consegue buscar todas as receitas com sucesso.

Código – `receitas.service.spec.ts`

```
import { TestBed } from '@angular/core/testing';
import { HttpClientTestingModule, HttpTestingController } from
'@angular/common/http/testing';
import { ReceitasService } from './receitas.service';
import { Receitas } from '../models/receitas';

describe('ReceitasService', () => {
  let service: ReceitasService;
  let httpMock: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [ReceitasService]
    });

    service = TestBed.inject(ReceitasService);
    httpMock = TestBed.inject(HttpTestingController);
  });

  it('deve retornar uma lista de receitas', () => {
    const mockReceitas: Receitas[] = [
      { id: 1, titulo: 'Bolo de Cenoura', tipoReceita: 'Doce',
ingredientes: [] },
      { id: 2, titulo: 'Pão de Queijo', tipoReceita: 'Salgado',
ingredientes: [] }
    ];

    service.findAllReceitas().subscribe((receitas) => {
      expect(receitas.length).toBe(2);
      expect(receitas).toEqual(mockReceitas);
    });

    const req = httpMock.expectOne('http://localhost:8080/api/receitas');
    expect(req.request.method).toBe('GET');
    req.flush(mockReceitas);
  });

  afterEach(() => {
    httpMock.verify();
  });
});
```

7.4.2. Teste de Integração – Componente de Lista de Receitas

Este teste verifica se o componente **ListaReceitasComponent** é carregado corretamente e se o método de busca é executado.

Código – `lista-receitas.component.spec.ts`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ListaReceitasComponent } from './lista-receitas.component';
import { ReceitasService } from '../../services/receitas.service';
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { of } from 'rxjs';

describe('ListaReceitasComponent', () => {
  let component: ListaReceitasComponent;
  let fixture: ComponentFixture<ListaReceitasComponent>;
  let receitasService: ReceitasService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      declarations: [ListaReceitasComponent],
      providers: [ReceitasService]
    });

    fixture = TestBed.createComponent(ListaReceitasComponent);
    component = fixture.componentInstance;
    receitasService = TestBed.inject(ReceitasService);

    spyOn(receitasService, 'findAllReceitas').and.returnValue(of([
      { id: 1, titulo: 'Bolo de Cenoura', tipoReceita: 'Doce',
ingredientes: [] }
    ]));
  });

  it('deve carregar receitas na inicialização', () => {
    component.ngOnInit();
    expect(component.receitas.length).toBe(1);
  });
});
```

7.5. Execução Final

Após a implementação e os testes, executamos a aplicação para garantir que tudo está funcionando corretamente. Para iniciar o backend e o frontend:

7.5.1. Executando o Backend

No diretório do projeto Java, execute:

```
mvn spring-boot:run
```

O backend estará disponível em **`http://localhost:8080`**

7.5.2. Executando o Frontend Angular

No diretório do projeto Angular, execute:

```
ng serve
```

A aplicação estará disponível em **`http://localhost:4200`**

7.6. Sugestões de Melhorias

1. **Validação de Formulários**: Adicionar validações nos campos de cadastro para garantir que o usuário preencha todas as informações necessárias corretamente.
2. **Tratamento de Erros**: Implementar interceptadores HTTP para exibir mensagens de erro claras ao usuário em caso de falha nas requisições.
3. **Paginação**: Adicionar paginação na lista de receitas para melhorar a navegação quando houver muitos itens.
4. **Autenticação e Autorização**: Incluir autenticação para permitir que apenas usuários cadastrados possam adicionar ou excluir receitas.

7.7. Evolução da Aplicação

7.7.1. Integração com Banco de Dados Relacional

Atualmente, estamos utilizando uma estrutura básica. Uma possível evolução seria **integrar a aplicação com um banco de dados relacional** (como MySQL ou PostgreSQL) para armazenar as receitas de forma persistente.

7.7.2. Implementação de Autenticação

Podemos integrar a aplicação com uma plataforma de autenticação, como o **OAuth2**, para permitir que os usuários se autentiquem usando suas contas de Google ou Facebook.

7.7.3. Implementação de Avaliação de Receitas

Adicionar um sistema de **avaliação com estrelas** para que os usuários possam avaliar e comentar nas receitas.

7.7.4. Mobile First

Melhorar o design da interface para seguir o princípio **mobile-first**, garantindo uma experiência de usuário ainda melhor em dispositivos móveis.

7.8. Conclusão

Neste capítulo, realizamos os **testes necessários para validar o funcionamento da aplicação** e sugerimos melhorias para garantir a qualidade e a evolução contínua do projeto. A aplicação "Receitas da Mamãe" está pronta para ser utilizada, proporcionando uma experiência agradável e intuitiva ao usuário. As sugestões de evolução permitirão que o projeto cresça de forma sustentável, garantindo que novas funcionalidades possam ser adicionadas no futuro.

Conclusão

A apostila **"Desenvolvimento Web com Angular - Do Zero a Quase Hero"** apresentou todas as etapas essenciais para o desenvolvimento de uma aplicação web moderna e funcional, utilizando Angular para o frontend e uma API em Java para o backend. Através de exemplos práticos, estruturamos uma aplicação chamada "Receitas da Mamãe", explorando desde a criação do protótipo até a implementação final com foco na experiência do usuário e na integração eficiente entre as camadas do sistema. Ao longo do processo, enfatizamos boas práticas de design, modularização do código e uso de ferramentas como **Bootstrap** e **Font Awesome** para garantir uma interface agradável e responsiva.

Os capítulos abordaram de forma didática a configuração do ambiente de desenvolvimento, a criação de componentes e serviços, a integração com o backend e a execução de testes para validar o comportamento do sistema. Além disso, a utilização de conceitos como **injeção de dependências** e **arquitetura REST** reforçou a importância de construir uma aplicação desacoplada e escalável, preparada para evoluir conforme as necessidades do projeto. As sugestões de melhorias apresentadas ao longo da apostila, como a inclusão de validações, paginação e autenticação, demonstram a preocupação em alinhar a aplicação com as melhores práticas de desenvolvimento web.

Por fim, a construção dessa aplicação não apenas oferece uma introdução sólida ao desenvolvimento com Angular e Java, mas também serve como uma base para projetos futuros, incentivando a prática contínua e o aprimoramento de habilidades. Com a aplicação dos conceitos apresentados, espera-se que os alunos e desenvolvedores possam desenvolver soluções robustas e modernas, prontas para atender às demandas do mercado. A partir deste ponto, o conhecimento adquirido pode ser expandido, possibilitando a inclusão de novas funcionalidades e integração com tecnologias mais avançadas no futuro.