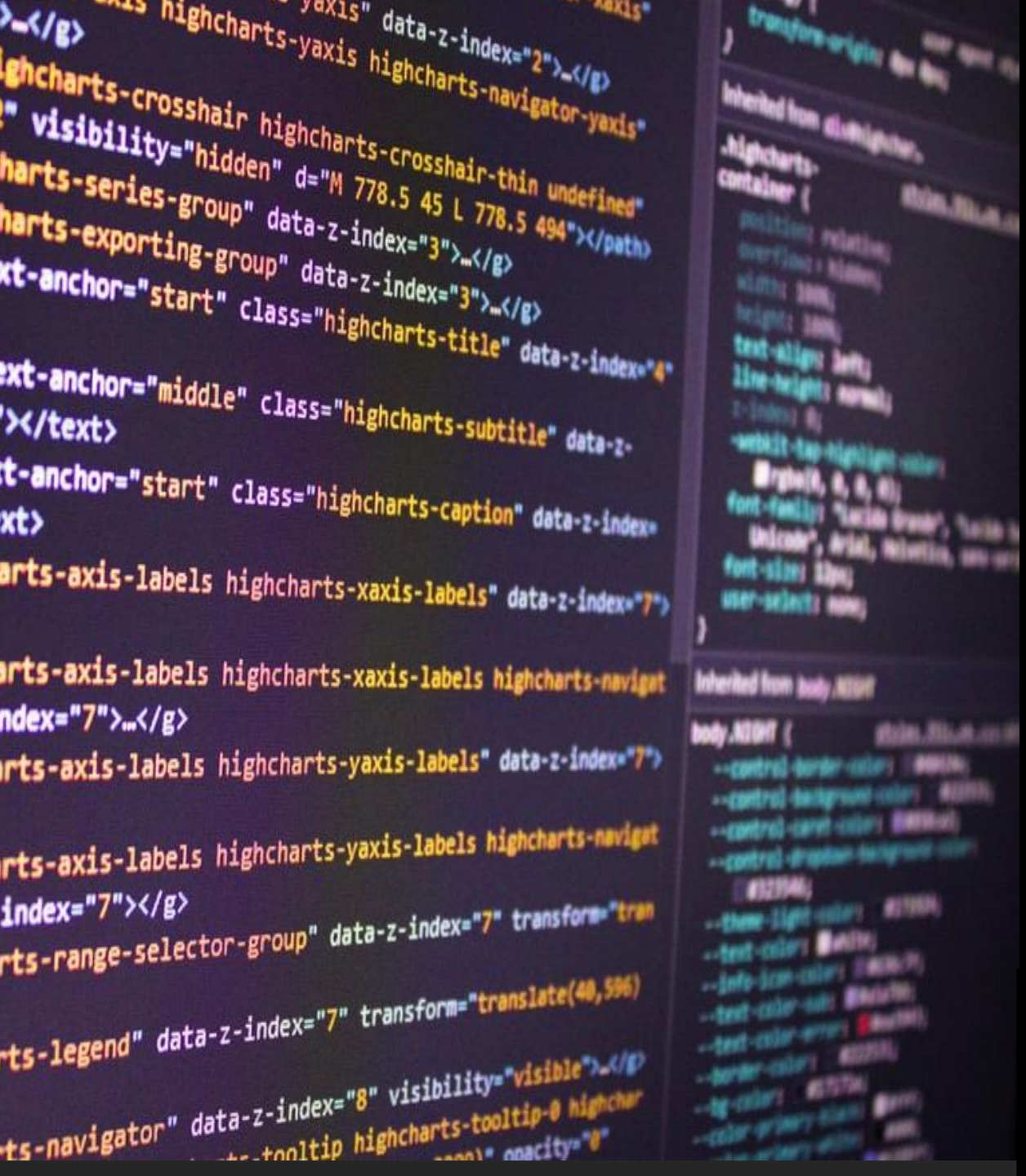


Projeto Integrador II

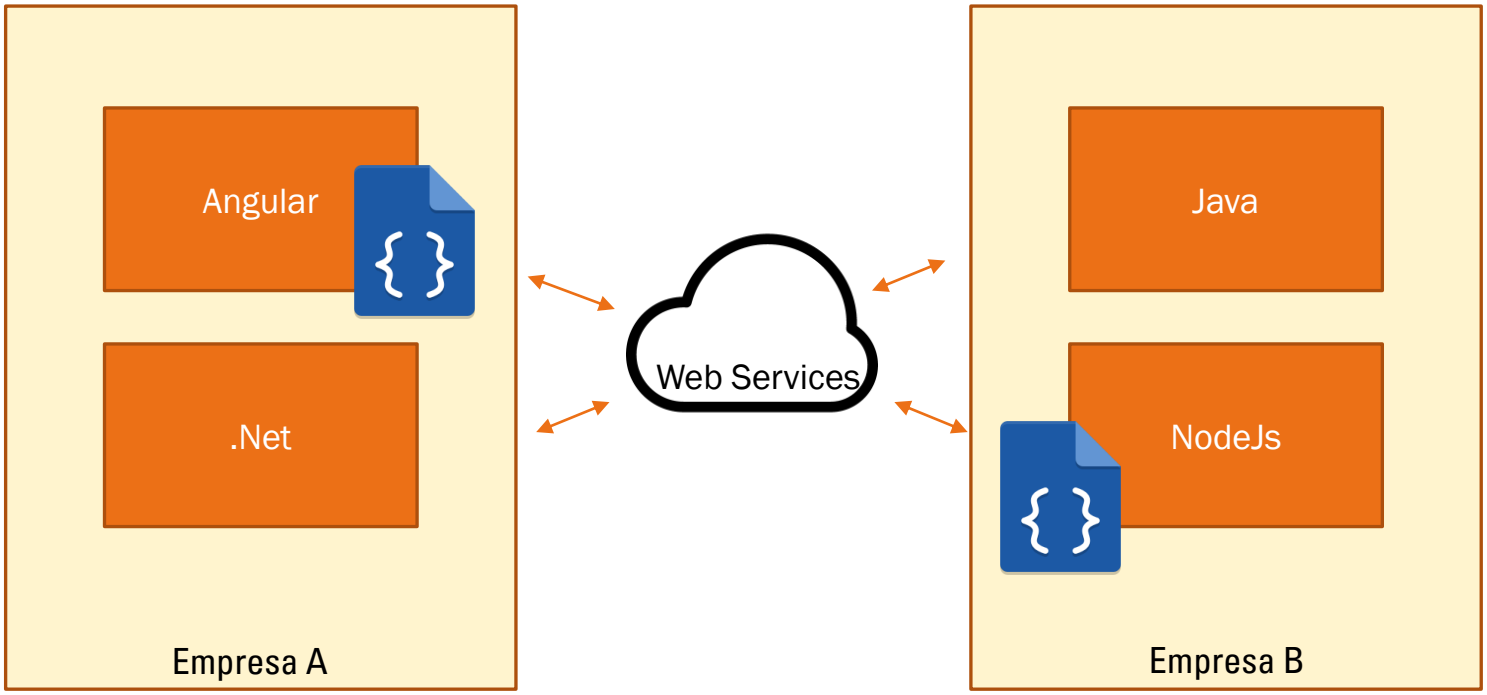
PROF. DR. FERNANDO T. FERNANDES

Agenda - CRUD

- ❑ Conceitos básicos de APIs
- ❑ Interagindo com uma API
- ❑ Implementação de API com JSON-SERVER

APIs
(Application Programmming Interface)

Integração usando APIs REST



APIs REST

- REST (*Representational State Transfer*)
  - Estilo de arquitetura criado em 2000 por **Roy Fielding**
  - Simplifica a construção de web services
  - Define **diretrizes** para criar serviços web (*web services*) simples e escaláveis
  - **Não mantém estado entre as operações**

  - APIs RESTful - Implementam conceitos REST para comunicação entre sistemas
  - **Representação** de formatos: Normalmente JSON e XML (Também HTML e YML)
  - **Operações**: Métodos GET/POST/PUT/DELETE do protocolo HTTP
  - **Recurso**: Uniform Resource Identifier (URI) – Arquivo ou recurso a ser buscado

Métodos HTTP

- Usamos os seguintes métodos HTTP:
  - **GET** (Consultas)
  - **POST** (Inclusões)
  - **PUT** (Alterações)
  - **DELETE** (Exclusões)
    - outros – PATCH (Alterar parte do conteúdo)

- **Stateless** (Não mantém estado)
  - Cada requisição (*request*) é independente

- Permite o consumo por meio de URIs

Retorno de Status HTTP

- 1XX – Informativo
- 2XX – Sucesso
  - Ex: HTTP 200 – OK ; 201 – Criação de um registro; 204 – Exclusão bem sucedida
- 3XX – Redirecionamento para outra URL
  - Erro: 301 - Recurso Movido Permanentemente
- 4XX – Erros na requisição do cliente
  - Erro: 404 – página não encontrada
- 5XX – Erros do Servidor
  - Erro 500 – Internal Server Error

RxJS
(Reactive Extensions for Javascript)

RxJS (Reactive Extensions for JavaScript)

Biblioteca para lidar com **operações assíncronas**.

Observables – fluxo de dados assíncrono que pode emitir múltiplos valores ao longo do tempo.

Subscribe – Permite **consumir** um *Observable.* Usamos o método .subscribe()

Pode ter até três funções de ***callback:***

- **Sucesso – (next)** – Executada quando um valor é emitido
- **Erro – (error) –** Executada quando ocorre um erro
- **Complete (complete) -** Executado quando o Observable finaliza

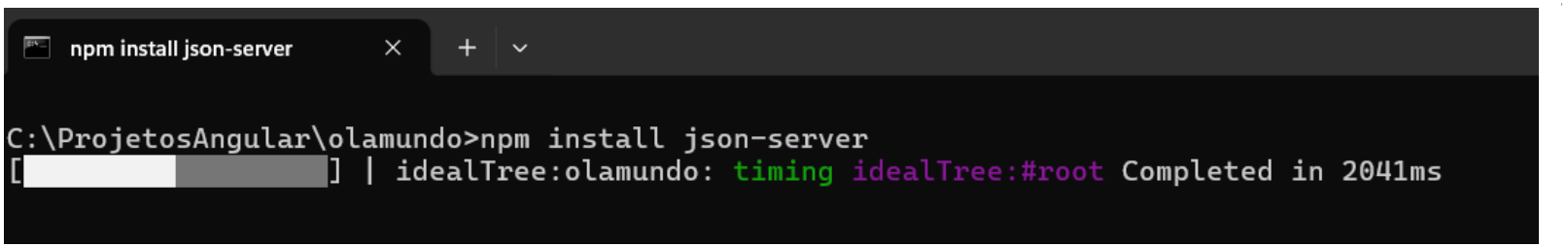
JSON-SERVER
(Preparação da nossa API simulada)

JSON-SERVER

- Ferramenta que permite simular uma **API REST**
- Permite simular requisições HTTP
  - Métodos: GET, POST, PUT, PATCH e DELETE
- Funciona com um arquivo JSON
- Rápido e fácil de configurar

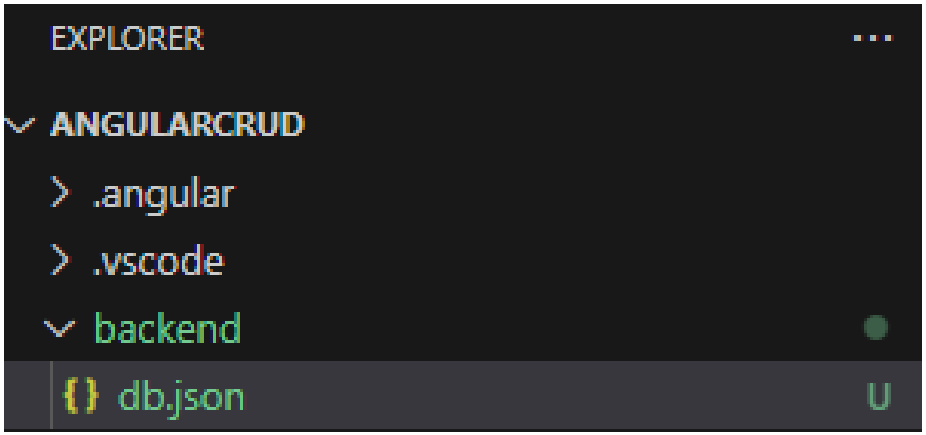
JSON-SERVER - instalação

```
npm install json-server
```



JSON-SERVER –db.json

1) Crie uma pasta chamada **backend a** partir da raiz do projeto



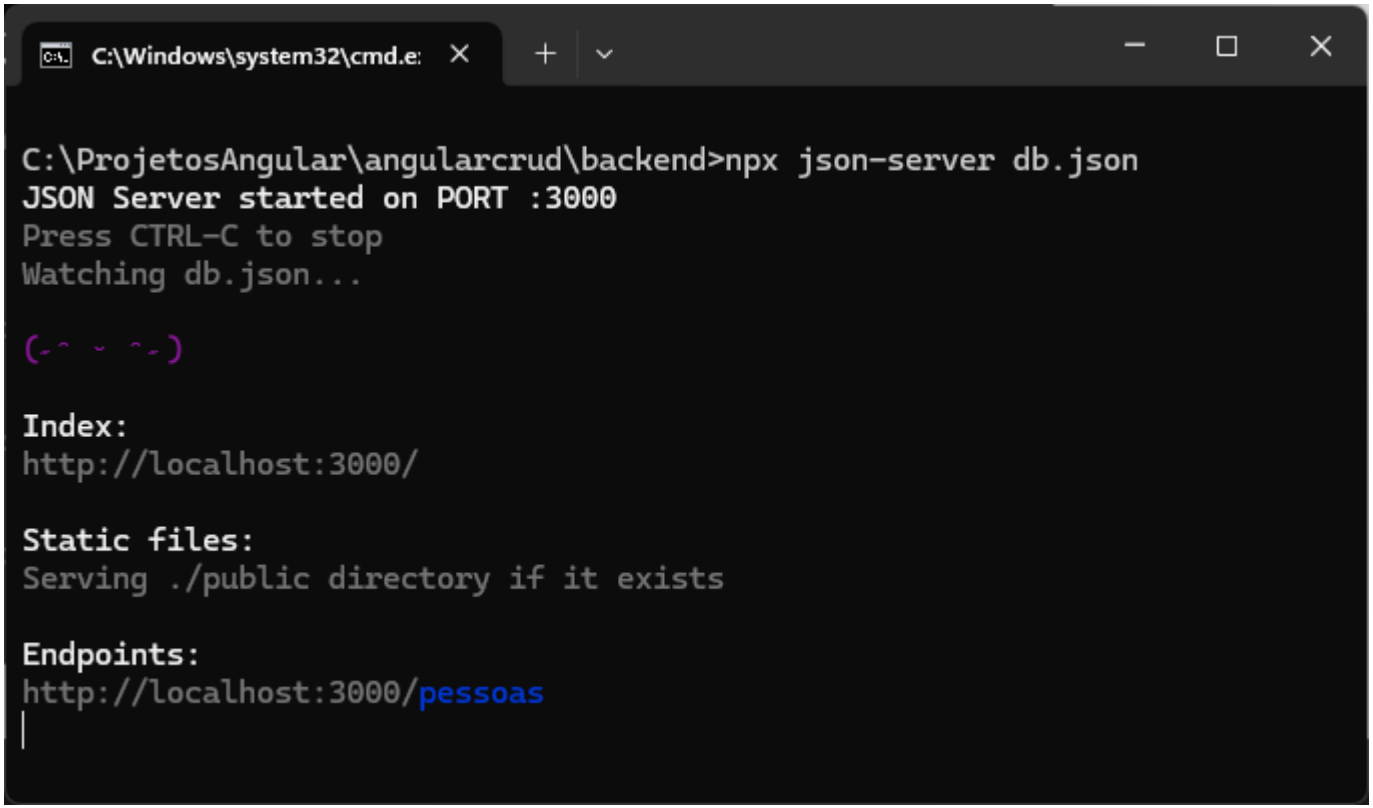
2) Adicione um arquivo **db.json** à pasta com o seguinte conteúdo:

```
{
    "pessoas": [
      {
        "id": 1,
        "nome": "Enzo",
        "sobrenome": "Silva",
        "dtNascimento": "2005-01-10"
      },
      {
        "id": 2,
        "nome": "Valentina",
        "sobrenome": "Guimaraes",
        "dtNascimento": "2007-08-14"
      }
    ]
}
```

Subir a API com o json-server

Abra um terminal e navegue até a pasta **backend** do projeto

Execute o comando: `npx json-server db.json`



Testando a API

`http://localhost:3000/pessoas`


```
[
  {
    "id": "1",
    "nome": "Enzo",
    "sobrenome": "Silva",
    "dtNascimento": "2005-01-10"
  },
  {
    "id": "2",
    "nome": "Valentina",
    "sobrenome": "Guimaraes",
    "dtNascimento": "2007-08-14"
  }
]
```

Preparar o consumo da API (Camada de Serviços)

pessoa.service.ts - listagem de pessoas de forma assíncrona

```
import { Injectable } from '@angular/core';
import { Pessoa } from '../types/types';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

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

  private readonly API = 'http://localhost:3000/pessoas'

  constructor(private http: HttpClient) { }

  listar(): Observable<Pessoa[]>{
    return this.http.get<Pessoa[]>(this.API)
  }

}
```

Permitir chamadas http

Endereço da API

Injeção de dependência

Chamada assíncrona via método **GET**

pessoa.service.ts - demais métodos do CRUD

```
export class PessoaService {

  private readonly API = 'http://localhost:3000/pessoas'

  constructor(private http: HttpClient) { }

  listar(): Observable<Pessoa[]>{
    return this.http.get<Pessoa[]>(this.API)
  }

  buscarPorId(id: number): Observable<Pessoa | undefined> {
    return this.http.get<Pessoa>(this.API + `/${id}`);
  }

  incluir(pessoa: Pessoa): Observable<Pessoa>{
    return this.http.post<Pessoa>(this.API, pessoa)
  }

  editar(pessoa: Pessoa): Observable<Pessoa>{
    const url = `${this.API}/${pessoa.id}`
    return this.http.put<Pessoa>(url, pessoa)
  }

  excluir(id: number): Observable<Pessoa>{
    return this.http.delete<Pessoa>(this.API + `/${id}`);
  }
}
```

Preencher uma pessoa a partir do ID

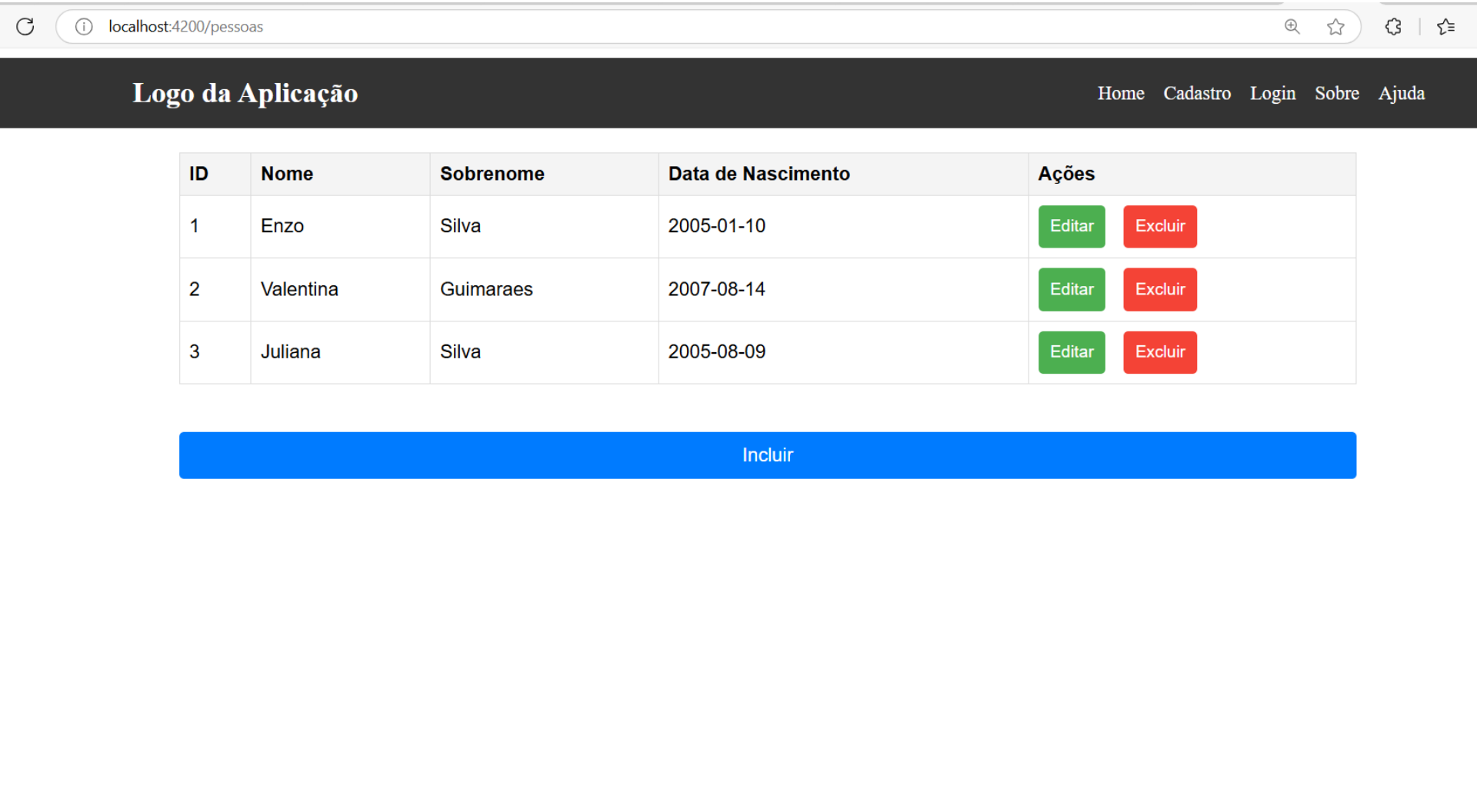
Incluir um objeto via método **POST**

Alterar um objeto via método **PUT**

Excluir um objeto via método **DELETE**

Consulta e Exclusão de Pessoas

Formulário de Listagem
pessoa-listagem.component.html



pessoa-listagem.component.ts – Habilitar **listagem** e **exclusão** de pessoas

```
export class PessoaListagemComponent implements OnInit {

  listaPessoas: Pessoa[] = [];

  constructor(
      private service: PessoaService,
      private router: Router
    ){}

  ngOnInit(): void {

    this.service.listar().subscribe((pessoas)=>{
      this.listaPessoas = pessoas;
    });

  }
  excluir(id:number){
    if(id){
      this.service.excluir(id).subscribe(() => {
          window.location.reload()
      })
    }
  }
}
```

Permitir o roteamento após a exclusão

Subscrição à API

Chama a camada de serviço para excluir a pessoa a partir do id

pessoa-listagem.component.html – Adicionar rotas de **alteração** e **exclusão**

```
<div class="content-container">

    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nome</th>
                <th>Sobrenome</th>
                <th>Data de Nascimento</th>
                <th>Ações</th>
            </tr>
        </thead>
        <tbody>
            @for (pessoa of listaPessoas; track pessoa){
                <tr>
                    <td>{{pessoa.id}}</td>
                    <td>{{pessoa.nome}}</td>
                    <td>{{pessoa.sobrenome}}</td>
                    <td>{{pessoa.dtNascimento}}</td>
                    <td>
                        <button class="btn-editar" [routerLink]="['alterar', pessoa.id]" >Editar</button>
                        <button class="btn-excluir" (click)="excluir(pessoa.id!)">Excluir</button>
                    </td>
                </tr>
            }
        </tbody>
    </table>
    <button class="btn-incluir" router-link="/pessoas/incluir">Incluir</button>

</div>
```

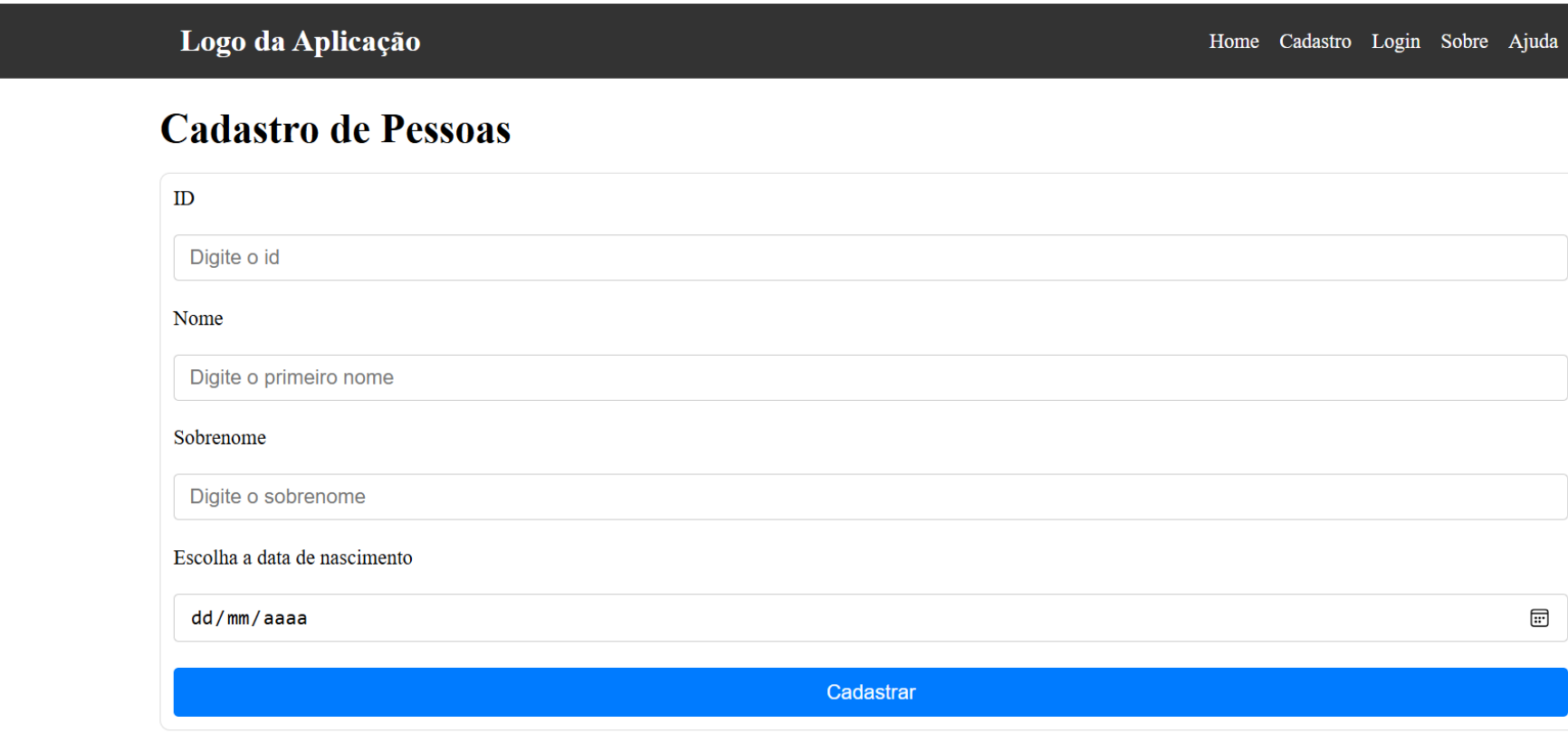
Inclusão e Alteração

Inclusão e Alteração

- Usaremos o formulário de cadastro tanto para inclusão de novas pessoas, quanto para alteração.

- Para isso, precisaremos verificar se na requisição constam o parâmetro com o id da pessoa.
  - Se constar, alteramos, senão incluímos um novo registro.

Formulário de Cadastro (com id)



pessoa-form.component.ts – Preparação do formulário para **cadastro** ou **alteração**

```
export class PessoaFormComponent {

  titulo = 'Cadastro de Pessoas';

  pessoaId?: number;

  //Defino um objeto pessoa que será incluído ou alterado.
  pessoa: Pessoa = {} as Pessoa;

  constructor(
    private service: PessoaService,
    private router: Router,
    private route: ActivatedRoute
  ){
      //Verifico se é alteração
      this.pessoaId = Number(this.route.snapshot.params['id']);

      if (this.pessoaId) {
        service.buscarPorId(this.pessoaId).subscribe(pessoa => {
          if (pessoa) {
            this.pessoa.id = pessoa.id;
            this.pessoa.nome = pessoa.nome;
            this.pessoa.sobrenome = pessoa.sobrenome;
            this.pessoa.dtNascimento = pessoa.dtNascimento;
          }
        })
      }
  } //Fim do contrutor
```

Injeto as dependências da camada de **serviços**, **leitura de parâmetros e redirecionamento**

Se estiver em modo de **alteração**, preencho o objeto **pessoa**

pessoa-form.component.ts – Preparação do método submeter para **cadastro** ou **alteração**

```
submeter(){

  if(this.pessoaId){

    this.service.editar(this.pessoa).subscribe(() =>{
      this.router.navigate(['/pessoas'])
    })

  }else{
    this.service.incluir(this.pessoa).subscribe(()=>{
      this.router.navigate(['/pessoas'])
    })
  }
} //Fim do método submeter
```

Caso seja alteração, chamo o método **editar**

Caso seja inclusão, chamo o método **incluir**

pessoa-form.component.html – Ativar evento de **submissão** e vínculo bi-direcional

```
<form>
        <label for="input-id">ID</label><br>
        <input class="input-simples" id="input-id" name="input-id"
placeholder="Digite o id" type="text" [(ngModel)]="pessoa.id" />

        <label for="input-nome">Nome</label><br>
        <input class="input-simples" id="input-nome" name="input-nome"
placeholder="Digite o primeiro nome" type="text" [(ngModel)]="pessoa.nome"/>

        <label for="input-sobrenome">Sobrenome</label><br>
        <input class="input-simples" id="input-sobrenome" name="input-
sobrenome" placeholder="Digite o sobrenome" type="text"
[(ngModel)]="pessoa.sobrenome"/>

        <label for="input-nascimento">Escolha a data de
nascimento</label><br>
        <input class="input-simples" id="input-nascimento" name="input-
nascimento" type="date" [(ngModel)]="pessoa.dtNascimento"/>

        <button class="input-submeter" (click)="submeter()" >
            Cadastrar
        </button>
    </form>
```

Adicione os campos label e input para o ID

Adicione o **ngModel** para comunicação bi-direcional.

Adicione o evento **click** ao botão cadastrar que aciona o método **submeter** do ts.

Lab – Simulação de CRUD

1. Faça uma inclusão pelo formulário de cadastro e consulte a API

2. Faça uma alteração de um registro

3. Exclua o registro adicionado

Tipo de tarefa: **Individual**

Tempo Estimado: **5 minutos**

Conclusão - CRUD

- ✓ Conceitos básicos de APIs
- ✓ Interagindo com uma API
- ✓ Implementação de API com JSON-SERVER

ADO 2 – CRUD com JSON-SERVER

1. Crie um projeto **usando Angular** no padrão “<SeuGrupo>ADO2”.

   Ex: LivrariaADO2

2. Faça um CRUD de produtos para produtos de seu site**.**

   a) Permitir a inclusão, alteração, exclusão e exclusão de produtos

   b) Use o JSON-SERVER para simular a API

3. Insira o código do formulário de cadastro e o formulário de listagem criados no documento WORD (incluindo CSS, HTML e **TS**)

4. Inserir uma **capa** com o **NOME COMPLETO** dos integrantes.

5. **Alternativa de entrega:** Vídeo demonstrando o uso da aplicação.

Tempo Estimado: **2 semanas**

Obrigado!

proffernandotfernandes