



CHALLENGE idwall

O Poder transformador da tecnologia em
tornar o mundo um lugar mais seguro

KeepCalm

FASE 03

Victor José Bueno Araújo - 94295

Iris Magno de Azevedo - 94900

SUMÁRIO

SISTEMA DO FBI E INTERPOL	4
MER - MODELAGEM DE ENTIDADES E RELACIONAMENTO	4
PROTÓTIPO FBI E INTERPOL.....	6
BAIXANDO OS DADOS DA API DO FBI - PYHTON.....	6
APLICAÇÃO JAVA	11
FRONT-END.....	23
SYSTEM PRACTICE - PROTOTIPAGEM	26
DOCUMENTAÇÃO DA API.....	28
CONCLUSÃO	32

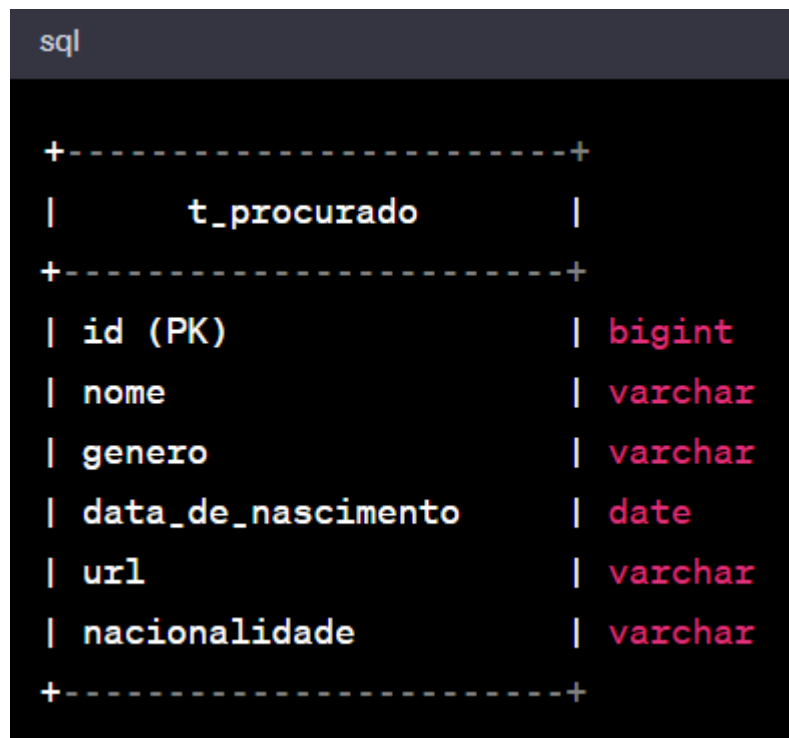
TABELA DE FIGURAS

Figura 1: Modelo Lógico da tabela t_procurados	4
Figura 2: Modelo físico criação tabela t_procurados	5
Figura 3: Algoritmo para baixar os dados da API 1	7
Figura 4: Algoritmo para baixar os dados da API 2	9
Figura 5: Página com resultado positivo para pesquisa de procurado	26
Figura 6: Página com resultado negativo para pesquisa de procurado.....	26

SISTEMA DO FBI E INTERPOL

Criação de um sistema para *idwall* que gerencie as pessoas mais procuradas do FBI e Interpol.

MER - MODELAGEM DE ENTIDADES E RELACIONAMENTO



```
sql

+-----+
|      t_procurado      |
+-----+
| id (PK)                | bigint
| nome                   | varchar
| genero                  | varchar
| data_de_nascimento     | date
| url                    | varchar
| nacionalidade           | varchar
+-----+
```

Figura 1: Modelo Lógico da tabela t_procurados

O banco de dados "Procurados" é composto por uma tabela chamada "*t_procurado*" que armazena informações sobre indivíduos procurados. A tabela possui as seguintes colunas:

- **id:** Identificador único para cada registro na tabela. É um valor numérico do tipo "bigint".
- **nome:** Armazena o nome do procurado. É um valor de texto do tipo "varchar".
- **genero:** Armazena o gênero do procurado. É um valor de texto do tipo "varchar".
- **data_de_nascimento:** Armazena a data de nascimento do procurado. É um valor de data do tipo "date".

- **url:** Armazena uma URL relacionada ao procurado. É um valor de texto do tipo "varchar".
- **nacionalidade:** Armazena a nacionalidade do procurado. É um valor de texto do tipo "varchar".

O banco de dados "**Procurados**" é utilizado para armazenar informações relevantes sobre pessoas procuradas. Cada registro na tabela representa um indivíduo procurado e contém dados como nome, gênero, data de nascimento, URL relacionada e nacionalidade.

O banco de dados pode ser utilizado para diversas finalidades, como pesquisa e localização de pessoas procuradas pelas autoridades, acompanhamento e registro de informações atualizadas sobre os procurados, e apoio em investigações e ações de segurança.

O modelo lógico do banco de dados "**Procurados**" fornece uma estrutura organizada e eficiente para armazenar e recuperar informações sobre pessoas procuradas. Ele pode ser adaptado e refinado de acordo com as necessidades e requisitos específicos do sistema.

```
SCSS

CREATE TABLE t_procurados (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY,
    nome VARCHAR2(200),
    nacionalidade VARCHAR2(100),
    data_de_nascimento_utilizada VARCHAR2(500),
    genero VARCHAR2(20),
    url VARCHAR2(500),
    PRIMARY KEY (id)
);
```

Figura 2: Modelo físico criação tabela t_procurados

Esse SQL é um comando de criação de tabela chamada "*t_procurados*". Está é a explicação dos elementos presentes na declaração:

- **id:** É uma coluna do tipo *NUMBER* que é gerada automaticamente com um valor único para cada registro inserido na tabela. A cláusula *GENERATED BY DEFAULT AS IDENTITY* indica que os valores para essa coluna serão gerados automaticamente em ordem crescente.
- **nome:** É uma coluna do tipo *VARCHAR2(200)* que armazena o nome do procurado. O tamanho máximo é de 200 caracteres.
- **nacionalidade:** É uma coluna do tipo *VARCHAR2(100)* que armazena a nacionalidade do procurado. O tamanho máximo é de 100 caracteres.
- **data_de_nascimento_utilizada:** É uma coluna do tipo *VARCHAR2(500)* que armazena as datas de nascimento utilizadas para identificar o procurado. O tamanho máximo é de 500 caracteres.
- **genero:** É uma coluna do tipo *VARCHAR2(20)* que armazena o gênero do procurado. O tamanho máximo é de 20 caracteres.
- **url:** É uma coluna do tipo *VARCHAR2(500)* que armazena a URL associada ao procurado. O tamanho máximo é de 500 caracteres.
- **PRIMARY KEY (id):** Define a coluna id como chave primária da tabela, garantindo que cada registro tenha um valor único para essa coluna.

Em resumo, essa declaração cria uma tabela chamada "*t_procurados*" com colunas para armazenar informações sobre os procurados, como nome, nacionalidade, data de nascimento, gênero e URL. A coluna id é a chave primária da tabela e é gerada automaticamente.

PROTÓTIPO FBI E INTERPOL

BAIXANDO OS DADOS DA API DO FBI - PYHTON

CÓDIGO

```
import requests
import json

# Itera sobre as 500 páginas
for page in range(1, 501):
    response = requests.get('https://api.fbi.gov/wanted/v1/list',
    params={
```

```
        'page': str(page)
    })
    data = json.loads(response.content)

    # Itera sobre a lista de procurados e gera um comando INSERT para
    # cada um
    for item in data['items']:
        titulo = item['title'].replace('"', "'")
        genero = item['sex'].replace('"', "'") if item.get('sex')
    else ''
        datas_de_nascimento_utilizadas =
    str(item['dates_of_birth_used']).replace('"', "'") if item.get(
        'dates_of_birth_used') else ''
        url = item['url']
        nacionalidade = item['nationality'].replace('"', "'") if
    item.get('nationality') else ''

        insert = f"INSERT INTO t_procurado (id, nome, genero,
    data_de_nascimento_utilizada, url, nacionalidade) VALUES
    (sq_t_proc.NEXTVAL, '{titulo}', '{genero}',
    '{datas_de_nascimento_utilizadas}', '{url}', '{nacionalidade}');"
        print(insert)
```

Figura 3: Algoritmo para baixar os dados da API 1

Este algoritmo utiliza a biblioteca *requests* para fazer requisições *HTTP* e a biblioteca *json* para manipular dados em formato *JSON*. O objetivo do algoritmo é baixar dados de uma *API* e facilitar a inserção desses dados em um banco de dados.

O ALGORITMO SEGUE OS SEGUINTE PASSOS

1. **Iteração sobre as 500 páginas:** O algoritmo realiza um loop de 1 a 500 para iterar sobre as páginas da *API* de onde os dados serão obtidos.
2. **Requisição HTTP:** Para cada página, o algoritmo faz uma requisição *GET* para a URL '*https://api.fbi.gov/wanted/v1/list*' passando o número da página como parâmetro. Isso permite obter os dados da página específica.
3. **Conversão dos dados:** O conteúdo da resposta *HTTP* é convertido de *JSON* para um objeto *Python* usando a função *json.loads()*. Isso permite manipular e acessar os dados de forma mais conveniente.

4. **Iteração sobre a lista de procurados:** O algoritmo itera sobre a lista de procurados obtida na resposta da **API**. Para cada procurado, são extraídas as informações relevantes, como título, gênero, datas de nascimento utilizadas, **URL** e nacionalidade.
5. **Manipulação dos dados:** As informações extraídas são tratadas para garantir que estejam no formato adequado para inserção no banco de dados. Por exemplo, caracteres especiais como aspas simples (') são substituídos por duas aspas simples (") para evitar erros de sintaxe SQL.
6. **Comando INSERT:** Com base nas informações extraídas, o algoritmo gera um comando *SQL INSERT* para cada procurado. O comando *INSERT* é uma string formatada que contém os valores a serem inseridos nas colunas da tabela "*t_procurado*".
7. **Impressão do comando INSERT:** O comando *INSERT* gerado para cada procurado é impresso no console. Isso facilita a visualização dos comandos e pode ser útil para depurar ou verificar os dados antes da inserção no banco de dados.

O algoritmo facilita a inserção dos dados no banco de dados ao gerar os comandos *INSERT* prontos para uso. Os comandos podem ser copiados e executados no banco de dados para inserir os registros correspondentes à lista de procurados baixados da *API*.

É importante ressaltar que o algoritmo apenas gera os comandos *INSERT*, mas não executa diretamente a inserção no banco de dados. Isso deve ser feito separadamente utilizando uma conexão com o banco de dados e executando os comandos gerados.

BAIXANDO OS DADOS DA API DA INTERPOL

```
import requests
import json

# URL da API
url = 'https://ws-public.interpol.int/notices/v1/red?nationality=us&page=1&resultPerPage=200'

# Enviar solicitação HTTP
response = requests.get(url)
```



```
# Verificar se a solicitação HTTP foi bem sucedida
if response.status_code == 200:
    # Acessar o JSON
    json_data = response.json()

    # Iterar sobre cada aviso vermelho (red notice)
    for notice in json_data['_embedded']['notices']:
        # Concatenar os campos forename e name
        full_name = f"{notice['forename']} {notice['name']}"

        # Obter a data de nascimento no formato correto
        date_of_birth = notice['date_of_birth'].replace('/', '-')

        # Converter a lista de nacionalidades em uma string separada
        # por vírgulas
        nationalities_str = ', '.join(notice['nationalities'])

        # Obter o gênero
        gender = 'Desconhecido' # Valor padrão caso não haja
        # informação de gênero
        if 'sex' in notice:
            if notice['sex'] == 'M':
                gender = 'Masculino'
            elif notice['sex'] == 'F':
                gender = 'Feminino'

        # Obter a URL
        url = notice['_links']['self']['href']

        # Imprimir o comando SQL de inserção
        print(f"INSERT INTO t_procurado (id, nome, genero,
        data_de_nascimento_utilizada, url, nacionalidade) VALUES
        (sq_t_proc.NEXTVAL, '{full_name}', '{gender}', '{date_of_birth}',
        '{url}', '{nationalities_str}');"
        else:
            # Imprimir o status code da resposta HTTP em caso de erro
            print('Erro:', response.status_code)
```

FIGURA 4: Algoritmo para baixar os dados da api 2

O ALGORITMO REALIZA AS SEGUINTE AÇÕES

1. **Importação de bibliotecas:** As bibliotecas *requests* e *json* são importadas para lidar com as requisições *HTTP* e manipulação de dados *JSON*, respectivamente.
2. **Definição da URL da API:** É definida a *URL* da *API* de onde os dados serão obtidos. Nesse caso, a *URL* aponta para uma *API* que retorna

aviso vermelho da Interpol relacionados à nacionalidade dos Estados Unidos.

3. **Envio de solicitação HTTP:** O algoritmo faz uma requisição *GET* para a *URL* da *API* usando a função *requests.get()*. Isso envia a solicitação *HTTP* para a *API* e obtém a resposta.
4. **Verificação do status da resposta HTTP:** O algoritmo verifica se a solicitação *HTTP* foi bem-sucedida verificando o código de status da resposta. Se o código de status for 200 (indicando uma resposta bem-sucedida), o algoritmo continua o processamento. Caso contrário, é exibido o status code da resposta *HTTP* como uma mensagem de erro.
5. **Acesso aos dados JSON:** Se a resposta da *API* for bem-sucedida, o conteúdo *JSON* é extraído da resposta usando o método *json()* da biblioteca *requests*. Isso converte os dados *JSON* em um objeto *Python* para facilitar a manipulação.
6. **Iteração sobre os avisos vermelhos:** O algoritmo itera sobre cada aviso vermelho (*red notice*) presente nos dados *JSON*. Para cada aviso, são extraídas informações relevantes, como nome completo, data de nascimento, nacionalidades, gênero e *URL*.
7. **Manipulação dos dados:** As informações extraídas são tratadas conforme necessário. Por exemplo, o nome completo é obtido concatenando os campos "*forename*" e "*name*", a data de nascimento é convertida para o formato correto substituindo '/' por '-' e a lista de nacionalidades é convertida em uma string separada por vírgulas.
8. **Impressão do comando SQL de inserção:** Para cada aviso vermelho, o algoritmo gera um comando *SQL INSERT* correspondente e imprime no console. O comando *INSERT* contém os valores a serem inseridos nas colunas da tabela "*t_procurado*". O valor para a coluna "*id*" é gerado usando a sequência "*sq_t_proc.NEXTVAL*".

O algoritmo facilita a obtenção dos dados da *API* da Interpol relacionados aos avisos vermelhos e gera comandos *SQL* de inserção prontos para uso no banco de dados. Os comandos *INSERT* podem ser copiados e executados para inserir os registros correspondentes na tabela "*t_procurado*".

APLICAÇÃO JAVA

CLASSE PROCURADOS

```
package br.com.fiap.procurados.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.SequenceGenerator;
import jakarta.persistence.Table;

@Entity
@Table(name = "t_procurado")
public class Procurados {

    @Id
    @SequenceGenerator(name="proc",sequenceName="sq_t_proc",allocationSize
=1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator="proc")
    @Column(name = "id")
    private Long id;

    @Column(name = "nome")
    private String nome;

    @Column(name = "genero")
    private String sexo;
```

```
        @Column(name = "data_de_nascimento_utilizada")
        private String dataNascimento;

        @Column(name = "url")
        private String url;

        @Column(name = "nacionalidade")
        private String nacionalidade;

        public Procurados() {
        }

        public Procurados(String nome, String sexo, String dataNascimento, String url, String
nacionalidade) {
            this.nome = nome;
            this.sexo = sexo;
            this.dataNascimento = dataNascimento;
            this.url = url;
            this.nacionalidade = nacionalidade;
        }

        // Getters and Setters

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }
    }
}
```

```
}
```

```
public String getNome() {
```

```
    return nome;
```

```
}
```

```
public void setNome(String nome) {
```

```
    this.nome = nome;
```

```
}
```

```
public String getSexo() {
```

```
    return sexo;
```

```
}
```

```
public void setSexo(String sexo) {
```

```
    this.sexo = sexo;
```

```
}
```

```
public String getDataNascimento() {
```

```
    return dataNascimento;
```

```
}
```

```
public void setDataNascimento(String dataNascimento) {
```

```
    this.dataNascimento = dataNascimento;
```

```
}
```

```
public String getUrl() {
```

```
    return url;
```

```
}
```

```
public void setUrl(String url) {  
    this.url = url;  
}  
  
public String getNacionalidade() {  
    return nacionalidade;  
}  
  
public void setNacionalidade(String nacionalidade) {  
    this.nacionalidade = nacionalidade;  
}  
  
@Override  
public String toString() {  
    return "Procurados [id=" + id + ", nome=" + nome + ", sexo=" + sexo + ",  
dataNascimento=" + dataNascimento  
        + ", url=" + url + ", nacionalidade=" + nacionalidade + "];"  
}  
  
}
```

DETALHAMENTO DOS PRINCIPAIS ELEMENTOS DA CLASSE

A classe Procurados é uma entidade de modelo que representa um registro na tabela `"t_procurado"` do banco de dados. A classe é anotada com várias anotações da especificação *JPA (Java Persistence API)*, que fornecem mapeamentos entre a classe e a tabela no banco de dados.

Segue abaixo uma explicação detalhada dos principais elementos da classe:

- **@Entity:** Essa anotação indica que a classe é uma entidade *JPA*, ou seja, ela representa uma tabela no banco de dados.
- **@Table(name = "t_procurado"):** Essa anotação especifica o nome da tabela associada à entidade. No caso, o nome da tabela é *"t_procurado"*.
- **@Id:** Essa anotação marca o campo *id* como a chave primária da tabela.
- **@SequenceGenerator:** Essa anotação especifica o uso de uma sequência para gerar os valores da coluna *id*. A sequência é definida com o nome *"proc"* e o nome da sequência no banco de dados é *"sq_t_proc"*.
- **@GeneratedValue:** Essa anotação especifica como os valores da coluna *id* são gerados. No caso, está configurado para usar *GenerationType.SEQUENCE*, indicando que os valores serão obtidos a partir da sequência especificada.
- **@Column:** Essa anotação mapeia o campo para uma coluna da tabela. É possível especificar o nome da coluna usando o parâmetro *name*. Por exemplo, o campo *nome* é mapeado para a coluna *"nome"*.
- **Campos da entidade:** Os campos da classe representam as colunas da tabela. Eles são mapeados para as colunas correspondentes usando a anotação *@Column*.
- **Construtores:** A classe possui dois construtores. O primeiro é o construtor padrão, sem argumentos, e o segundo é um construtor que aceita os valores para os campos da entidade.
- **Getters e Setters:** Para cada campo da entidade, existem métodos *getter* e *setter* correspondentes, permitindo acessar e modificar os valores dos campos.
- **toString():** O método *toString()* é sobrescrito para fornecer uma representação em formato de *string* da instância da classe. Ele retorna uma *string* contendo os valores dos campos da entidade.

Essa classe representa a estrutura do objeto *Procurados* no código Java e fornece métodos para acessar e manipular os dados dessa entidade. Ela é usada em conjunto com o *JPA* para persistir e recuperar os objetos *Procurados* no banco de dados.

REPOSITORY

```
package br.com.fiap.procurados.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import br.com.fiap.procurados.model.Procurados;

public interface ProcuradosRepository extends JpaRepository<Procurados, Long> {

    Procurados findByNome(String nome);

    List<Procurados> findByNomeContainingIgnoreCase(String nome);

}
```

DETALHAMENTO DOS ELEMENTOS PRESENTES NA INTERFACE

Esse código define uma interface *ProcuradosRepository* que estende a interface *JpaRepository* do *Spring Data JPA*. Essa interface é responsável por fornecer métodos para interagir com a entidade *Procurados* no banco de dados.

Segue abaixo uma explicação detalhada dos elementos presentes na interface:

- **ProcuradosRepository estende JpaRepository<Procurados, Long>:** Isso indica que *ProcuradosRepository* é uma interface de repositório JPA para a entidade *Procurados*. A interface *JpaRepository* fornece métodos *CRUD* (*Create, Read, Update, Delete*) e outras operações de consulta.
- **Procurados findByNome(String nome):** Esse método é uma operação de consulta personalizada que retorna um objeto *Procurados* com base no nome fornecido como parâmetro. Ele é implementado automaticamente pelo *Spring Data JPA* com base no nome do método.

Por exemplo, ao chamar `findByNome("João")`, o *Spring Data JPA* executará uma consulta para buscar um registro na tabela `t_procurado` com o nome "João".

- **List<Procurados> findByNomeContainingIgnoreCase(String nome):**
Esse método também é uma operação de consulta personalizada que retorna uma lista de objetos *Procurados* cujos nomes contenham o valor fornecido como parâmetro. A consulta é case-insensitive, ou seja, não diferencia letras maiúsculas de minúsculas. Por exemplo, ao chamar `findByNomeContainingIgnoreCase("Silva")`, o *Spring Data JPA* executará uma consulta para buscar todos os registros na tabela `t_procurado` com o nome contendo "Silva", independentemente de estar em maiúsculas ou minúsculas.

Essa interface *ProcuradosRepository* define métodos de consulta personalizados que permitem buscar objetos *Procurados* no banco de dados com base em critérios específicos. O *Spring Data JPA* é responsável por implementar automaticamente esses métodos de acordo com as convenções de nomenclatura e os tipos de retorno especificados na interface. Isso simplifica o desenvolvimento do acesso ao banco de dados, permitindo que faça consultas personalizadas de forma fácil e eficiente.

RESOURCE

```
package br.com.fiap.procurados.resources;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.fiap.procurados.model.Procurados;
import br.com.fiap.procurados.repository.ProcuradosRepository;
```

```
@RestController
@RequestMapping("/procurados")
public class ProcuradosResource {

    private final ProcuradosRepository procuradosRepository;

    public ProcuradosResource(ProcuradosRepository procuradosRepository) {
        this.procuradosRepository = procuradosRepository;
    }

    @GetMapping("/{nome}")
    public ResponseEntity<Object> getProcuradoPorNome(@PathVariable
String nome) {
        Procurados procurado = procuradosRepository.findByNome(nome);
        if (procurado != null) {
            return ResponseEntity.ok(procurado);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Não
consta como procurado na base de dados");
        }
    }
}
```

DETALHAMENTO ELEMENTOS PRESENTES NA CLASSE

Esse código define uma classe `ProcuradosResource` que é responsável por lidar com as requisições relacionadas aos recursos dos procurados. Essa classe é anotada com `@RestController`, indicando que é um controlador de *API REST*.

Segue abaixo uma explicação detalhada dos elementos presentes na classe:

- **ProcuradosResource possui uma dependência injetada de ProcuradosRepository:** O construtor da classe recebe uma instância de *ProcuradosRepository*, que é usado para acessar e manipular os dados dos procurados no banco de dados.
- **@RequestMapping("/procurados"):** Essa anotação define o mapeamento base para todas as requisições relacionadas aos procurados. Ou seja, todas as rotas definidas neste controlador terão */procurados* como prefixo de *URL*.
- **@GetMapping("/{nome}"):** Essa anotação indica que o método *getProcuradoPorNome* irá lidar com requisições *GET* para a rota */procurados/{nome}*, onde *{nome}* é um parâmetro de caminho (*path variable*) que representa o nome do procurado a ser buscado.
- **public ResponseEntity<Object>
getProcuradoPorNome(@PathVariable String nome):** Esse método é responsável por buscar um procurado pelo nome fornecido como parâmetro na *URL*. Ele retorna uma instância de *ResponseEntity<Object>*, que encapsula a resposta *HTTP*.
- **Procurados procurado = procuradosRepository.findByNome(nome):** Esse trecho de código utiliza o *ProcuradosRepository* para buscar um procurado pelo nome no banco de dados. O resultado é atribuído à variável *procurado*.
- **if (procurado != null) { return ResponseEntity.ok(procurado); }:** Verifica se um procurado foi encontrado. Se sim, retorna uma resposta *HTTP* com status 200 (OK) e o objeto procurado no corpo da resposta.
- **Else { return
ResponseEntity.status(HttpStatus.NOT_FOUND).body("Não consta como procurado na base de dados"); }:** Caso o procurado não seja encontrado, retorna uma resposta *HTTP* com status 404 (NOT FOUND) e uma mensagem de erro informando que o procurado não consta na base de dados.

Essa classe `ProcuradosResource` define um endpoint de *API REST* para buscar um procurado pelo nome. Ao receber uma requisição *GET* para a rota `/procurados/{nome}`, ela utiliza o `ProcuradosRepository` para buscar o procurado correspondente no banco de dados e retorna uma resposta *HTTP* com o resultado da busca.

CONTROLLER

```
package br.com.fiap.procurados.controller;

import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import br.com.fiap.procurados.model.Procurados;
import br.com.fiap.procurados.repository.ProcuradosRepository;

@Controller
public class ProcuradosController {

    private final ProcuradosRepository procuradosRepository;

    public ProcuradosController(ProcuradosRepository procuradosRepository) {
        this.procuradosRepository = procuradosRepository;
    }

    @GetMapping("/procurados")
    public String pesquisarProcurado(@RequestParam("nome") String nome,
        Model model) {
        List<Procurados> procurados =
            procuradosRepository.findByNomeContainingIgnoreCase(nome);
```

```
        model.addAttribute("procurados", procurados);
        model.addAttribute("temProcurados", !procurados.isEmpty()); // Verifica se
a lista de procurados não está vazia
        return "Procurados/Procurados";
    }

    @GetMapping("/home")
    public String exibirHome(Model model) {
        model.addAttribute("temProcurados", false); // Define como false para
ocultar a mensagem de "Não consta como procurado na base de dados"
        return "Procurados/Procurados";
    }
}
```

DETALHAMENTO DOS ELEMENTOS PRESENTES NA CLASSE

Esse código define a classe `ProcuradosController`, que é um controlador responsável por lidar com as requisições relacionadas aos procurados. Essa classe é anotada com `@Controller`, indicando que é um controlador *MVC*.

Segue abaixo uma explicação detalhada dos elementos presentes na classe:

- **ProcuradosController possui uma dependência injetada de ProcuradosRepository:** O construtor da classe recebe uma instância de *ProcuradosRepository*, que é usado para acessar e manipular os dados dos procurados no banco de dados.
- **@GetMapping("/procurados"):** Essa anotação indica que o método `pesquisarProcurado` irá lidar com requisições *GET* para a rota `/procurados`.
- **public String pesquisarProcurado(@RequestParam("nome") String nome, Model model):** Esse método é responsável por pesquisar procurados com base em um nome fornecido como parâmetro na *URL*. Ele retorna uma string que representa o nome da página de destino a ser exibida.

- **@RequestParam("nome"):** Essa anotação indica que o parâmetro nome é um parâmetro de requisição. Ele será preenchido com o valor fornecido na URL, seguindo o padrão *?nome=valor*.
- **List<Procurados> procurados = procuradosRepository.findByNomeContainingIgnoreCase(nome):**
Esse trecho de código utiliza o ProcuradosRepository para buscar uma lista de procurados cujo nome contenha o valor fornecido. O resultado é atribuído à lista procurados.
- **model.addAttribute("procurados", procurados):** Adiciona o atributo "procurados" ao modelo, tornando a lista de procurados disponível na página de destino.
- **model.addAttribute("temProcurados", !procurados.isEmpty()):**
Adiciona o atributo *temProcurados* ao modelo, que é usado para verificar se a lista de procurados não está vazia. Esse atributo é usado na página de destino para decidir se exibe ou não a mensagem "Não consta como procurado na base de dados".
- **return "Procurados/Procurados":** Retorna o nome da página de destino que será exibida. Nesse caso, a página é *Procurados/Procurados*, que provavelmente está dentro da pasta de visualizações (*views*) do projeto.
- **@GetMapping("/home"):** Essa anotação indica que o método *exibirHome* irá lidar com requisições *GET* para a rota */home*.
- **public String exibirHome(Model model):** Esse método é responsável por exibir a página inicial. Ele adiciona o atributo "temProcurados" ao modelo e define seu valor como false, para ocultar a mensagem "Não consta como procurado na base de dados".

Em resumo, esse controlador ProcuradosController lida com a pesquisa de procurados com base no nome fornecido, faz consultas no banco de dados usando *ProcuradosRepository* e retorna o resultado para uma página de destino. Ele também possui um método para exibir a página inicial sem resultados de pesquisa.

FRONT-END

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="style.css">
  <title>Página de Pesquisa de Procurados</title>
  <!-- Importando os estilos do Bootstrap -->
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
  <!-- Importando os scripts do Bootstrap -->
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></scri
pt>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.j
s"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></s
cript>
</head>
<body style="background-color: rgba(165, 209, 241, 1)">
  <div class="container">
    <h1 class="mt-5">Pesquisa de Procurados</h1>
    <form action="#" th:action="@{/procurados}" method="get">
      <div class="form-group mt-5">
        <label class="mb-2" for="nome">Nome do Procurado:</label>
        <input type="text" class="form-control" id="nome" name="nome"
placeholder="Digite o nome do procurado">
      </div>
      <button type="submit" class="btn btn-primary mt-
2">Pesquisar</button>
    </form>
    <div th:if="${procurados != null}">
      <h2 class="mt-5 mb-5">Informações dos Procurados:</h2>
      <div th:each="procurado : ${procurados}">
        <p><strong>Nome:</strong> <span
th:text="${procurado.nome}"></span></p>
        <p><strong>Nacionalidade:</strong> <span
th:text="${procurado.nacionalidade}"></span></p>
        <p><strong>Sexo:</strong> <span
th:text="${procurado.sexo}"></span></p>
        <p><strong>Data de Nascimento:</strong> <span
th:text="${procurado.dataNascimento}"></span></p>
        <p><strong>URL:</strong> <a th:href="${procurado.url}"
th:text="${procurado.url}"></a></p>
      <hr>
    </div>
    </div>
    <div th:if="${temProcurados != true and procurados != null}"
class="alert alert-danger">
      <h2>Não consta como procurado na base de dados!</h2>
    </div>
  </div>
</body>
<footer th:if="${temProcurados != true}" class="text-center">

```

```

```

```
</footer>  
</html>
```

DETALHAMENTO FRONT-END

Página *HTML* com marcação *Thymeleaf*, uma linguagem de modelagem para construir páginas da *web* com base em modelos. Vamos analisar os elementos principais do código:

- **<!DOCTYPE html>**: Define o tipo de documento como *HTML5*.
- **<html xmlns:th="http://www.thymeleaf.org">**: Define o namespace *th* como um namespace *Thymeleaf*.
- **<head>**: Contém informações e metadados da página.
- **<meta charset="UTF-8">**: Define a codificação de caracteres como *UTF-8*.
- **<link rel="stylesheet" href="style.css">**: Importa um arquivo CSS chamado *style.css* para estilizar a página.
- **<title>Página de Pesquisa de Procurados</title>**: Define o título da página exibido na barra de título do navegador.
- **Importações do Bootstrap**: Importa estilos CSS e scripts *JavaScript* do Bootstrap para aplicar estilos e funcionalidades adicionais à página.
- **<body style="background-color: rgba(165, 209, 241, 1)">**: Define o corpo da página.
- **<div class="container">**: Cria um container Bootstrap para organizar o conteúdo da página.
- **<h1 class="mt-5">Pesquisa de Procurados</h1>**: Título principal da página.
- **<form action="#" th:action="@{/procurados}" method="get">**: Formulário para pesquisar procurados.
- **<div class="form-group mt-5">**: Grupo de formulário.

- `<label class="mb-2" for="nome">Nome do Procurado:</label>`: Rótulo para o campo de entrada de nome.
- `<input type="text" class="form-control" id="nome" name="nome" placeholder="Digite o nome do procurado">`: Campo de entrada de texto para digitar o nome do procurado.
- `<button type="submit" class="btn btn-primary mt-2">Pesquisar</button>`: Botão de envio do formulário.
- `<div th:if="{procurados != null}">`: Verifica se a lista de procurados não é nula.
- `<h2 class="mt-5 mb-5">Informações dos Procurados:</h2>`: Título para as informações dos procurados.
- `<div th:each="procurado : {procurados}">`: Loop que itera sobre cada objeto "procurado" na lista de procurados.
- `<p>Nome: </p>`: Exibe o nome do procurado.
- `<p>Nacionalidade: </p>`: Exibe a nacionalidade do procurado.
- `<p>Sexo: </p>`: Exibe o sexo do procurado.
- `<p>Data de Nascimento: </p>`: Exibe a data de nascimento do procurado.
- `<p>URL: <a th:href="{procurado.url}" th:text="{procurado.url}"></p>`: Exibe a URL relacionada ao procurado como um link.

SYSTEM PRACTICE - PROTOTIPAGEM

PÁGINA HOME

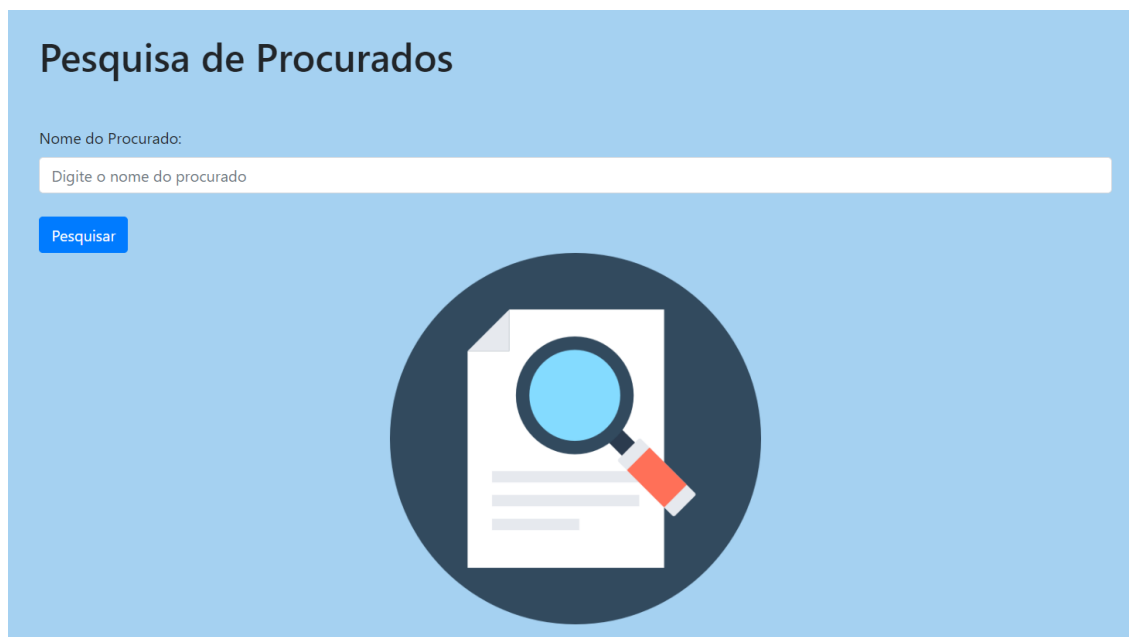


Figura 5: Página com resultado positivo para pesquisa de procurado

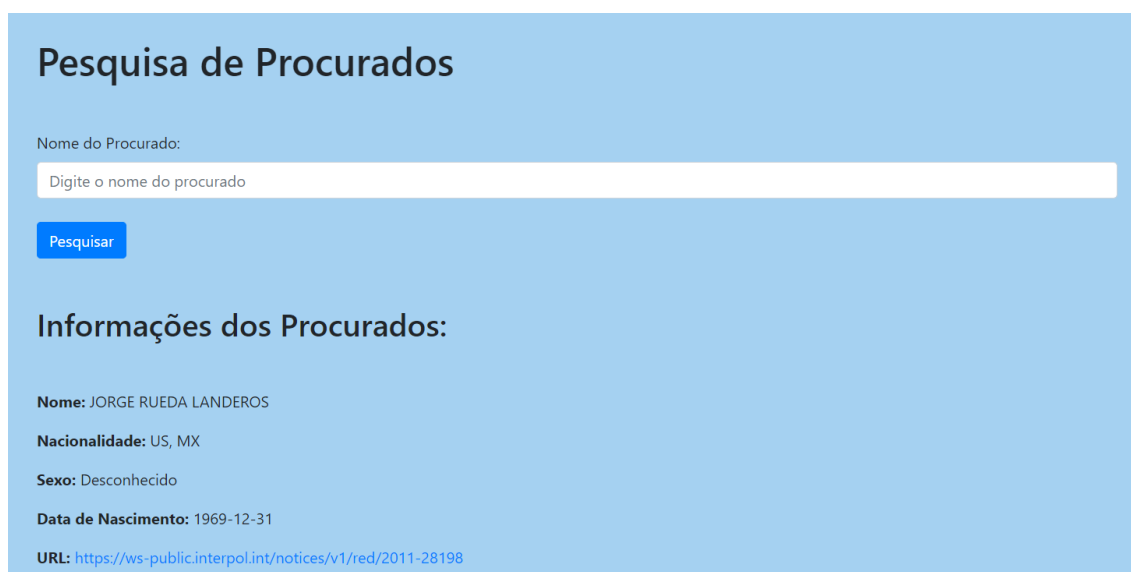


Figura 6: Página com resultado negativo para pesquisa de procurado

Pesquisa de Procurados

Nome do Procurado:

Pesquisar

Informações dos Procurados:

Não consta como procurado na base de dados!

TESTES REALIZADOS NA API UTILIZANDO POSTMAN

REGISTRO INEXISTENTE NO BD

The screenshot shows the Postman interface for a GET request to `http://localhost:8080/procurados/jorge`. The request is saved and has a status of 404 Not Found. The response body is displayed in the 'Body' tab, showing the message 'Não consta como procurado na base de dados'.

Overview GET http://localhost:8080/ No Environment

http://localhost:8080/procurados/jorge Save

GET http://localhost:8080/procurados/jorge Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results Status: 404 Not Found Time: 38 ms Size: 214 B Save Response

Pretty Raw Preview Visualize Text

1 Não consta como procurado na base de dados

REGISTRO EXISTENTE NO BD

Overview GET http://localhost:8080/ No Environment

http://localhost:8080/procurados/MANUEL Save

GET http://localhost:8080/procurados/MANUEL Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 40 ms Size: 326 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "nome": "MANUEL",
4   "sexo": "Desconhecido",
5   "dataNascimento": "1967-10-20",
6   "url": "https://ws-public.interpol.int/notices/v1/red/2022-35500",
7   "nacionalidade": "US"
8 }
```

DOCUMENTAÇÃO DA API

1. GET /procurados/{nome}

Recupera informações sobre uma pessoa procurada com base no nome.

Parâmetros da Solicitação

- **{nome}**: O nome da pessoa procurada para pesquisar (parâmetro de caminho).

Respostas

- 200 OK: Retorna as informações da pessoa procurada.

- Corpo:

```
json

{
  "id": 1,
  "nome": "John Doe",
  "sexo": "Masculino",
  "dataNascimento": "1980-01-01",
  "url": "https://example.com/procurado/johndoe",
  "nacionalidade": "Estados Unidos"
}
```

1.

- 404 Not Found: Se a pessoa procurada não for encontrada no banco de dados.
 - Corpo: "Não consta como procurado na base de dados"

2. GET /procurados

Pesquisa pessoas procuradas com base no nome.

Parâmetros de Consulta

- **nome:** O nome da pessoa procurada para pesquisar (parâmetro de consulta).

Respostas

- 200 OK: Retorna a lista de pessoas procuradas que correspondem aos critérios de pesquisa.
 - Corpo:

json

```
[
  {
    "id": 1,
    "nome": "John Doe",
    "sexo": "Masculino",
    "dataNascimento": "1980-01-01",
    "url": "https://example.com/procurado/johndoe",
    "nacionalidade": "Estados Unidos"
  },
  {
    "id": 2,
    "nome": "Jane Smith",
    "sexo": "Feminino",
    "dataNascimento": "1990-02-15",
    "url": "https://example.com/procurado/janesmith",
    "nacionalidade": "Canadá"
  }
]
```

1.

- 404 Not Found: Se não forem encontradas pessoas procuradas correspondentes aos critérios de pesquisa.
 - Corpo: "Não consta como procurado na base de dados"

Modelos de Dados

1. Pessoa Procurada (Procurados)

Representa uma pessoa procurada.

- **id** (inteiro): O identificador único da pessoa procurada.

- **nome** (string): O nome da pessoa procurada.
- **sexo** (string): O gênero da pessoa procurada.
- **dataNascimento** (string): A data de nascimento da pessoa procurada.
- **url** (string): A URL associada à pessoa procurada.
- **nacionalidade** (string): A nacionalidade da pessoa procurada.

Esta API fornece endpoints para recuperar informações sobre pessoas procuradas. O endpoint **GET /procurados/{nome}** permite procurar uma pessoa procurada específica pelo nome, enquanto o endpoint **GET /procurados** permite procurar pessoas procuradas com base em correspondência parcial de nomes.

CONCLUSÃO

Este projeto é uma aplicação web voltada para a pesquisa de pessoas procuradas. Vamos fazer uma análise geral do projeto com base nas informações fornecidas:

ARQUITETURA:

O projeto segue uma arquitetura em camadas, com uma separação clara entre a camada de apresentação (resources e controller) e a camada de persistência (model e repository). Isso permite uma melhor organização e manutenção do código.

TECNOLOGIAS UTILIZADAS:

- O projeto utiliza a linguagem Java e o framework Spring Boot para o desenvolvimento do backend.
- No banco de dados está sendo utilizado por meio de uma biblioteca de persistência JPA / Hibernate.
- O Thymeleaf é utilizado como mecanismo de modelagem para renderizar as páginas HTML.
- O Bootstrap é utilizado para fornecer estilos e componentes visuais para a interface do usuário.

FUNCIONALIDADES:

- A pesquisa de procurados é realizada por meio de um formulário onde o usuário pode digitar o nome do procurado.
- A página exibe as informações dos procurados encontrados, como nome, nacionalidade, sexo, data de nascimento e URL relacionada.
- Caso não haja procurados correspondentes à pesquisa, uma mensagem de "Não consta como procurado na base de dados" é exibida.
- O projeto também possui uma página inicial (home) que não exibe informações de procurados.

INTEGRAÇÃO COM A API:

O projeto contém um código Python que consome uma API (Interpol) para obter informações sobre procurados. Esse código faz uma solicitação HTTP para a API, obtém os dados no formato JSON e os processa para inserir no banco de dados.

Em resumo, o projeto apresenta uma interface de pesquisa de procurados, onde o usuário pode pesquisar por nome e obter informações sobre os procurados encontrados. Ele integra-se a uma API externa para obter os dados dos procurados e os armazena em um banco de dados por meio da entidade Procurados. O Thymeleaf é utilizado para renderizar as páginas HTML, e o Bootstrap é utilizado para fornecer estilos visuais. A arquitetura em camadas e a utilização do Spring Boot permitem uma estrutura organizada e modularizada do projeto.