

**Teste prático para vaga de Analista Desenvolvedor
NUTI (Núcleo de Tecnologia da Informação)**

Candidato: Rafael Ribeiro Estrela

Sumario

1. Descrição do desafio	1
2. Solução proposta	5
3. Implementação da solução	10
4. Conclusão	25

LISTA DE FIGURAS

Figura 1 - Diagrama entidade relacionamento

Figura 2 - Diagrama de classe

Figura 3 - Protótipo de baixo nível

Figura 4 - Protótipo de baixo nível

Figura 5 - Script de criação de usuários Oracle

Figura 6 - Script de criação de tabelas Oracle

Figura 7 - Script de inserção e busca no Oracle

Figura 8 - Estrutura de pacotes do projeto

Figura 9 - Configuração para perfil de teste

Figura 10 - Configuração para perfil de desenvolvimento

Figura 11 - Configuração para perfil de teste

Figura 12 - Classe de entidade Page

Figura 13 - Classe de entidade Tag

Figura 14 - Interface de repositório PageRepository

Figura 15 - Código HTML da página buscar_tags

Figura 16 - Interface web página home

Figura 17 - Interface web página identificar tags

Figura 18 - Interface web página com as tags identificadas de uma ou mais urls

Figura 19 - Interface web página buscar tags

Figura 20 - Interface web página com tags recuperadas do banco de dados

LISTA DE TABELAS

Tabela 1 - Caso de uso Identificar Tags

Tabela 2 - Caso de uso Buscar tags identificadas

Tabela 3 - Código Hibernate para buscar registros

Tabela 4 - Código Hibernate para inserir registros

1. Descrição do desafio

No código de uma página HTML há diversas tags para apresentar o conteúdo da melhor forma. Para uma análise mais cautelosa, há o interesse de contabilizar a quantidade de cada tag HTML em uma determinada página.

Sendo assim, é necessário criar um programa para identificar as tags HTML existentes nas páginas que forem carregadas por meio de uma lista de URL informada. Além disso, deve ser contado quantas vezes cada tag aparece em cada página.

É necessário mostrar as informações coletadas para possibilitar verificar os dados da URL informada. Assim, as URL, tags e as respectivas contagens devem ser armazenadas em um banco de dados.

Exemplo

Considere o código HTML abaixo.

```
<html>
  <head>
    <title>Teste prático</title>
  </head>
  <body>
    <h1>Olá</h1>
    <p>Teste 1</p>
    <p>Teste 2</p>
    <p>Teste 3</p>
  </body>
</html>
```

Ao processar essa página, o seguinte resultado seria apresentado:

tag	quantidade
html	1
head	1
title	1
body	1
h1	1
p	3

Entregas

Entregas relacionadas a desenvolvimento de sistemas:

- Código da aplicação que processa as páginas
- Código gerado para inserção e consulta no banco de dados
- Aplicação web para visualização das informações

Entregas relacionadas a análise de sistemas:

- Diagrama de Entidade Relacionamento (DER)
- Especificação de caso de uso
- Protótipo das páginas

2. Solução para o desafio

Com o intuito de solucionar o desafio proposto, será criada uma aplicação server-side utilizando a linguagem Java e o framework Spring Boot. Através da aplicação, o usuário poderá acessar as páginas web para analisar tags de uma página web ou buscar o resultados de análise já feitas, armazenar no banco de dados o resultado e retornar para o usuário.

2.1. Tecnologias utilizadas

- Linguagem de programação: Java 8;
- Build e deploy: Maven
- Frameworks:
 - Spring Boot: Framework principal que facilita a vida do desenvolvedor através de anotações e princípios como inversão de controle e injeção de dependências.
 - Spring Web: Framework para criação do servidor via tomcat para requisições http
 - Spring Data JPA: Framework que fornece abstração de camadas de acesso a dados, reduzindo o esforço ao valor realmente necessário.
 - jsoup: jsoup é uma biblioteca Java para trabalhar com HTML do mundo real. Ele fornece uma API muito conveniente para buscar URLs e extrair e manipular dados
 - h2: Biblioteca para conectar e manipular banco de dados relacional h2 em memória.
 - Oracle connector: Biblioteca para conectar no banco de dados Oracle
 - Thymeleaf: Framework que fornece templates e recursos de manipulação de páginas web para linguagem Java e Spring Boot
- Banco de dados: H2 em memória e Oracle;
- Versionador de código: GitHub Desktop
- IDE: IntelliJ, SQLDevelop

2.2. Artefatos do sistema

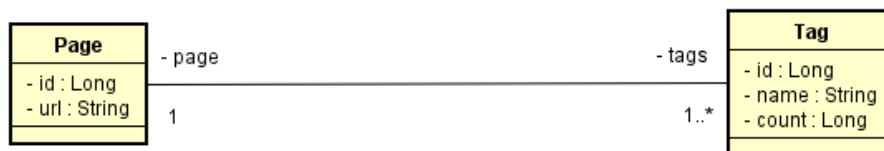
Diagrama entidade Relacionamento

Figura 1 - Diagrama entidade relacionamento



Diagrama de classe

Figura 2 - Diagrama de classe



Especificação de caso de uso

Tabela 1 - Caso de uso Identificar Tags

Caso de uso 1	Identificar tags
Descrição	Buscar e identificar tags de páginas da web através das urls.
Pré-condição	O usuário deve acessar a página principal: http://localhost:8080/api/v1/desafio/home
Fluxo Principal	<ol style="list-style-type: none">1. O usuário fornece uma lista de urls2. O sistema verifica no banco de dados se alguma url fornecida já foi analisada3. O sistema busca a página de acordo com a url e analisa as tags4. O sistema grava no banco de dados o resultado da análise5. O sistema retorna para o usuário o resultado.
Fluxo alternativo: O usuário fornece uma lista de urls vazia	O sistema emite uma mensagem de erro ao usuário, informando que é preciso no mínimo uma url na lista.
Fluxo alternativo: O sistema não pode conectar na url fornecida	O sistema emite uma mensagem de erro ao usuário, informando que não foi possível conectar em uma determinada url
Fluxo alternativo: O sistema detecta que uma análise já foi feita com URL informada	O sistema deve apagar os registros antigos e seguir para o item 3 do fluxo principal

Tabela 2 - Caso de uso Buscar tags identificadas

Caso de uso 2	Buscar tags identificadas
Descrição	Buscar páginas já analisadas e que estão no banco de dados
Pré-condição	O usuário deve acessar a página principal: http://localhost:8080/api/v1/desafio/home
Fluxo Principal	<ol style="list-style-type: none"> 6. O usuário fornece uma url 7. O sistema verifica se já existe uma análise da url fornecida no banco de dados 8. O sistema retorna para o usuário os dados encontrados.
Fluxo alternativo: O usuário não fornece nenhuma url	O sistema emite uma mensagem de erro ao usuário, informando que é preciso informar uma url
Fluxo alternativo: O sistema não encontra uma análise no banco de dados	O sistema emite uma mensagem de erro informando que não encontrou dados relacionados da url informada

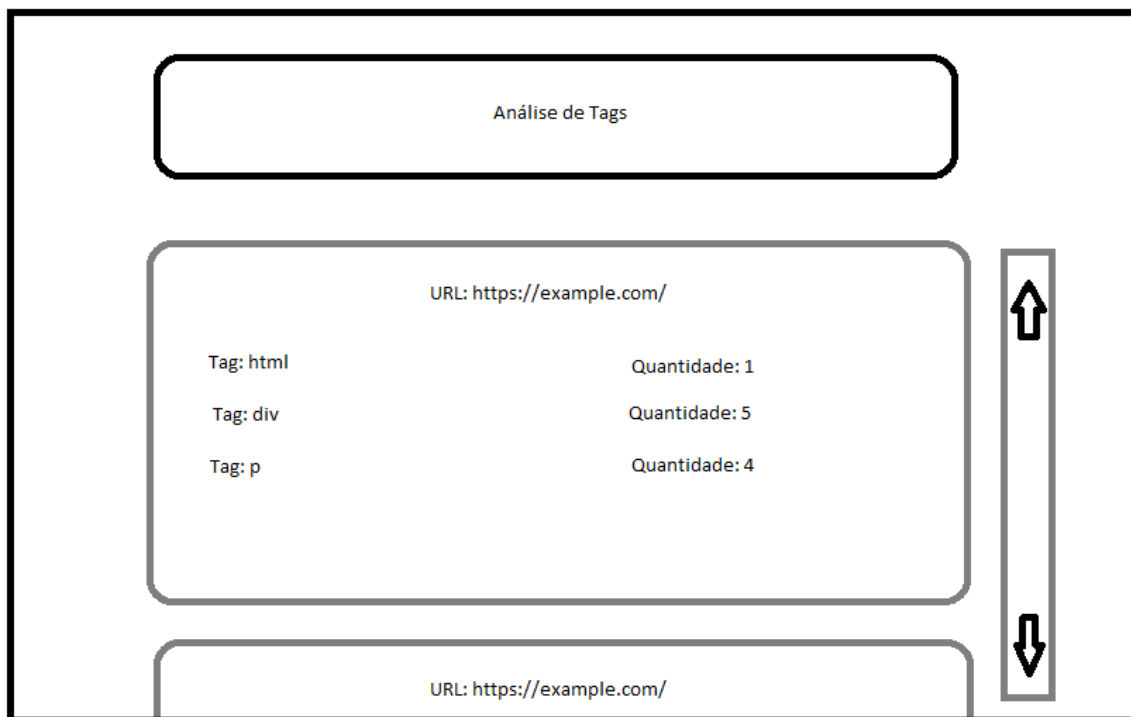
Protótipos de baixo nível

Figura 3 - Protótipo de baixo nível

The prototype is enclosed in a black rectangular border. At the top, there is a large, rounded rectangular box with the text "Análise de Tags" centered inside. Below this box, the text "Insira aqui as URLs:" is displayed. Underneath, there are two input fields, each containing a URL. The first input field contains the text "https://mylearn.oracle.com/ou/home" and has a red rectangular button labeled "Remover" to its right. The second input field contains the text "https://www.cebraspe.org.br/concursos/SERPRO_23" and also has a red rectangular button labeled "Remover" to its right. Below these two input fields, there is a green rectangular button labeled "Adicionar". At the bottom center of the prototype, there is a gray rectangular button labeled "Buscar e analisar".

Fonte: Imagem própria

Figura 4 - Protótipo de baixo nível



Fonte: Imagem própria

3. Implementação da solução

Dado a proposta de solução para resolver o desafio, será utilizado as tecnologias mencionadas no item 2.1 deste documento. A implementação de fato está armazenada no gerenciador de códigos GitHub.

3.1 Organização e configuração do ambiente de desenvolvimento

Antes de iniciar a implementação, foi feita a organização e configuração do ambiente de desenvolvimento de forma a garantir que todos os softwares necessários estivessem disponíveis para a implementação. Segue abaixo uma lista de atividades ordenadas para se iniciar a implementação:

1. Instalação do Java JDK
2. Instalação e configuração do banco de dados Oracle
3. Instalação da IDE IntelliJ
4. Instalação do versionador GitHub Desktop
5. Criação do repositório no GitHub
6. Download do projeto base no Spring Initializr
7. Abrir o projeto base na IDE
8. Associar o projeto com o repositório no GitHub

3.2 Estruturas do banco de dados

Para estruturar o banco de dados Oracle, primeiramente criamos um usuário somente para o projeto com todos os privilégios necessários para criar as tabelas, sequencias, índices, inserção, atualização, busca e deleção dos dados.

Segue o script de criação de usuários:

Figura 5 - Script de criação de usuários Oracle

```
-- SCRIPT USUARIO  
  
create user nuti identified by nuti;  
grant dba to nuti;  
grant connect to nuti;
```

Fonte: Imagem própria

Em seguida criamos as estruturas para comportar os dados do sistema de acordo com o DER

Segue o script das estruturas:

Figura 6 - Script de criação de tabelas Oracle

```
-- SCRIPT ESTRUTURAS

create sequence seq_page;

create table tb_page(
    id number default seq_page.nextval,
    url varchar2(255) not null,
    constraint tb_page_id_pk primary key(id),
    constraint tb_page_url_un unique(url)
);

create sequence seq_tag;

create table tb_tag(
    id number default seq_tag.nextval,
    name varchar2(255) not null,
    count number not null,
    id_page number not null,
    constraint tb_tag_id_pk primary key(id),
    constraint tb_tag_id_page_fk foreign key(id_page) references tb_page(id)
);
```

Fonte: Imagem própria

Por ultimo e opcional, podemos inserir e buscar dados de teste:

Segue o script de inserção e busca:

Figura 7 - Script de inserção e busca no Oracle

```
-- SCRIPT SEED

insert into tb_page (url)
values ('https://example.com/');

insert into tb_tag (name, count, id_page)
values ('html', 1, 1);

-- SCRIPT BUSCA

select pa.url, tg.name tag_name, tg.count tag_count
from tb_page pa
join tb_tag tg on (tg.id_page = pa.id)
where pa.url = 'https://example.com/';
```

Fonte: Imagem própria

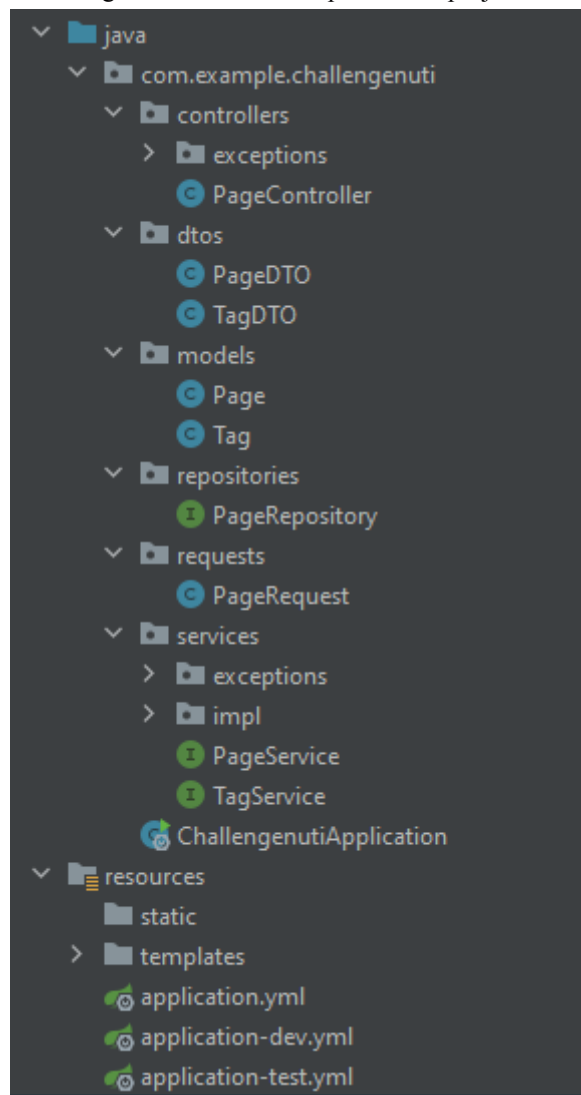
3.3 Implementação

Pacotes

Ao começar a implementar, foi seguido alguns padrões de projeto que serão descritos mais à frente. Um dos padrões de projeto define que devemos organizar nosso código em camadas, sendo que parte dessas camadas seguem o padrão MVC, tendo seu proprio objetivo e responsabilidade. Essas camadas podem ser representadas em forma de pacotes.

Segue abaixo os pacotes do projeto:

Figura 8 - Estrutura de pacotes do projeto



Fonte: Imagem própria

Descrição dos pacotes

- controllers: Pacote responsável por expor as chamadas das páginas web de acordo com endpoints.
- dtos: Pacote de classes que utilizamos para transitar dados entre as camadas do modelo MVC.
- models: Camada que contém classes de dados que representam entidades do banco de dados ou objetos do mundo real.
- repositories: Pacote com interfaces que fazem operações com banco de dados.
- services: Pacote com todas as regras de negócio
- utils: Pacote com classes utilitárias que auxiliam em algum processamento
- resources: Pacote com arquivos relacionados às páginas web

Configurações

Além da questão das camadas e pacotes, também foi definido arquivos de configuração de acordo com o ambiente de execução. Para o projeto foi criado dois ambientes, o de teste e o de desenvolvimento.

Para o ambiente de teste, utilizamos a seguinte configuração:

Figura 9 - Configuração para perfil de teste

```
spring:|
  datasource:
    url: jdbc:h2:mem:test
    driver-class-name: org.h2.Driver
    username: user
    password:
  jpa:
    database-platform: org.hibernate.dialect.H2Dialect
    show-sql: true
  h2:
    console:
      enabled: true
```

Fonte: Imagem própria

O ambiente de teste utiliza o banco de dados H2 em memória, oque é muito útil em casos que queremos testar se o mapeamento objeto relacional foi criado da forma correta, se os dados estão sendo armazenados e se as operações do banco de dados ocorrem de forma ininterrupta. Na configuração do ambiente, estamos configurando parâmetros de conexão com o banco de dados H2 e permitindo que quando houver uma operação com o banco de dados, o hibernate mostre através dos logs o comando e operação que foram executados.

Este ambiente é propicio para um momento inicial e não deve ser usado como parâmetro de testes mais profundos. **Como não há garantias de que o avaliador deste desafio terá o banco de dados Oracle instalado na maquina, o projeto será entregue com o perfil de teste, utilizando o banco de dados H2.**

Para o ambiente de desenvolvimento, utilizamos a seguinte configuração:

Figura 10 - Configuração para perfil de desenvolvimento

```
spring:
  datasource:
    url: jdbc:oracle:thin:@//localhost:1521/XEPDB1
    driver-class-name: oracle.jdbc.OracleDriver
    username: nuti
    password: nuti
  jpa:
    database-platform: org.hibernate.dialect.Oracle12cDialect
    show-sql: true
    hibernate:
      ddl-auto: none
```

Fonte: Imagem própria

Na configuração do ambiente, estamos configurando parâmetros de conexão com o banco de dados Oracle e permitindo que quando houver uma operação com o banco de dados, o hibernate mostre através dos logs o comando e operação que foram executados. Além disso existe a configuração do hibernate indicando que ele não deve criar as estruturas no banco de dados Oracle, uma vez que esta atividade já foi feita diretamente no banco de dados.

No ambiente default, utilizamos a seguinte configuração:

Figura 11 - Configuração para perfil de teste

```
spring:
  profiles:
    active: dev
  jpa:
    open-in-view: false
server:
  servlet:
    context-path: /api/${API_VERSION:v1}
```

Fonte: Imagem própria

Nessa configuração padrão definimos qual perfil de desenvolvimento usaremos e qual a URL base a aplicação estará funcionando. Além disso definimos uma configuração do framework JPA para que não permita que uma sessão com banco de dados fique aberta durante todo o processamento da requisição http, pois nesse momento podem ocorrer outras operações e processamentos, causando problemas de desempenho.

Classes de entidade

As classes que representam as entidades estão no pacote models. Essas classes possuem uma estrutura que representa as tabelas do banco de dados. No caso temos duas classes, a classe Page e Tag. Essas classes seguem o diagrama de classe mencionado no item 2.2.2. deste documento.

Segue abaixo as classes mencionadas:

Figura 12 - Classe de entidade Page

```
@Entity
@Table(name = "tb_page")
public class Page {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    2 usages
    private String url;
    1 usage
    @OneToMany(mappedBy = "page", fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private List<Tag> tags = new ArrayList<>();
}
```

Fonte: Imagem própria

Figura 13 - Classe de entidade Tag

```
@Entity
@Table(name = "tb_tag")
public class Tag {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    5 usages
    private String name;
    2 usages
    private int count;

    2 usages
    @ManyToOne
    @JoinColumn(name = "id_page")
    private Page page;
}
```

Fonte: Imagem própria

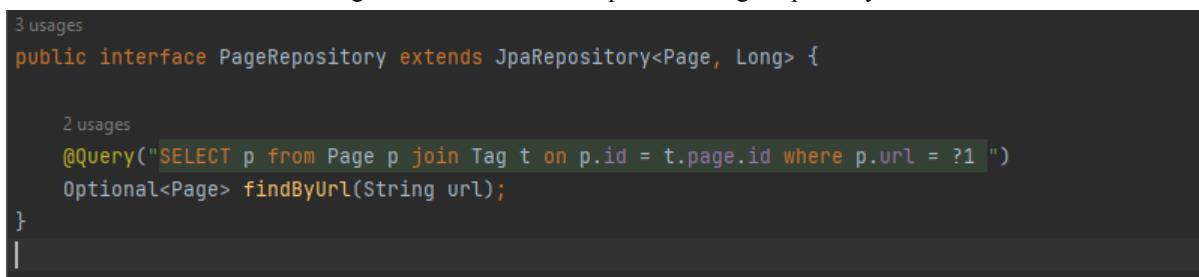
Como é possível observar, as classes utilizam anotações do Spring Data JPA para o correto mapeamento objeto relacional.

Interfaces de repositório

As interfaces de repositório é onde ocorrem as operações com o banco de dados. A maior parte das operações está encapsulada no framework Spring Data, de forma a abstrair implementações complexas desnecessárias no momento.

Segue abaixo a interface PageRepository

Figura 14 - Interface de repositório PageRepository



```
3 usages
public interface PageRepository extends JpaRepository<Page, Long> {

    2 usages
    @Query("SELECT p from Page p join Tag t on p.id = t.page.id where p.url = ?1 ")
    Optional<Page> findByUrl(String url);
}
|
```

Fonte: Imagem própria

A interface possui um método “findByUrl”, que permite buscar um registro do banco de dados através de uma URL. Essa operação segue o comando informado na anotação @Query, que utiliza a linguagem JPQL, que é uma linguagem de consulta do Hibernate mais próxima da Orientação a Objetos.

Outros métodos de busca, inserção e atualização, como mencionado, estão encapsulados no Spring Data.

Classes de serviços

As classes no pacote services possuem as regras de negócio explanadas na descrição do desafio. Todas as classes neste pacote possuem interfaces, que atuam como contratos dizendo quais os comportamentos essas classes devem ter e o retorno de cada comportamento. Além disso é uma boa prática para que possamos seguir um dos princípios do SOLID, uma vez que devemos depender de abstrações e não de implementações, o que permite que ao fazermos uma injeção de dependência, a classe que possui a implementação de fato, não cause problemas para a classe dependente.

A seguir será explicado o conceito e lógica aplicada para resolver de fato o desafio. Como a implementação possui código muito extenso, não será mostrado o código de fato neste documento mas sim uma explicação dos conceitos e lógica.

A classe `PageServiceImpl` possui dois métodos definidos de acordo com sua interface, “`buscarTags`” e “`identificarTagsPorUrl`”.

- a. “`buscarTags`”: O método `buscar tags por url` possui uma implementação simples, onde o método recebe como parâmetro uma url e chama o método “`findByUrl`” na camada `repository` para buscar um registro no banco de dados. Caso não retorne registros, será lançada uma exceção do tipo “`ResourceNotFoundException`” que significa que um determinado recurso não foi encontrado. Caso retorne registros, o mesmo será transportado para camadas superiores e mostrado ao usuário.
- b. “`identificarTags`”: O método `identificar tags` possui uma implementação onde o método recebe uma lista de urls e verifica se essa lista está vazia. Se estiver vazia ele lança uma exceção do tipo “`RequestInvalidException`” que significa que a requisição feita é inválida pois não atende a uma das regras. Após isso é feita uma iteração na lista de urls verificando se a url informada já possui uma análise no banco de dados. Se já existe uma análise, os registros são apagados e é feita uma nova análise. A implementação que faz a análise e processa as páginas está na classe “`TagService`”

A classe `TagServiceImpl` possui um método de acordo com sua interface, “`analisaTags`”.

- a. “`analisaTags`”: O método `analisa tags` possui uma implementação onde o método recebe uma url e através dessa url podemos conectar, extrair e retornar as tags da página. Para esse processo utilizamos a biblioteca do `jsoup`. Com a biblioteca conectamos na página através da url, extraímos todos os elementos encontrados, a partir disso, obtemos as tags e os elementos relacionados a essa tag, como nome e quantidade. Com esses dados, cumprimos com um dos requisitos de negócio que é obter as tags e quantidade de cada uma em uma página web. Após o processo retornamos uma lista de objetos do tipo “`Tag`” e posteriormente são salvos no banco de dados e retornados para o usuário.

Classes de exceção

As classes de exceção são responsáveis por armazenar os erros que ocorrem durante o processamento das requisições. Para cada tipo de erro usamos uma classe específica, propícia para aquele erro. Isso é recomendando pois é uma boa prática separar exceções para um melhor tratamento e visualização do usuário.

Classes de controle

As classes no pacote controllers expõem endpoints que podem ser acessados pelo usuário. Ao digitar o endpoint no navegador ou em algum outro software que permite requisições http, será retornado uma página de visualização. Nesta página o usuário poderá realizar interações com o sistema, inserido e buscando dados.

Classes de controle de exceções

Quando uma exceção ocorre na aplicação, essa exceção pode ser capturada, tratada ou exposta ao usuário. Dependendo da exceção, capturamos ela, estruturamos com o erro que ocorreu e retornamos ao usuário, para que o mesmo entenda o ocorrido e execute a operação da forma correta. Uma exceção personalizada que pode ser capturada e retornada ao usuário é a de "RequestInvalidException", que informa ao usuário que a forma que foi feita a requisição ao sistema possui algo inválido, por exemplo, passar uma lista de urls vazia.

Devido a falta de conhecimento e experiência com o framework Thymeleaf, não foi possível estruturar um procedimento que retorne ao usuário as exceções da forma correta. Portanto, ao ocorrer uma exceção, será retornada uma json ao usuário informando qual a exceção e problema ocorrido.

Arquivos de template

Os arquivos no pacote template é que representam a camada de visualização. Os arquivos html são estruturados nesse pacote com anotações do framework Thymeleaf, que nos permite exibir os dados da aplicação na página web. Essas anotações são como referências, ou seja, a anotação aponta diretamente para um objeto ou atributo presente na classe Model, que possui uma estrutura de dados do tipo chave-valor que armazena os dados que podem ser retornados na visualização da página.

Segue abaixo um exemplo da página html com as referências dos objetos utilizando Thymeleaf.

Figura 15 - Código HTML da página buscar_tags

```

</head>
<body>
<div class="container my-4">
  <h1 class="mb-4">Tags das páginas</h1>
  <div th:each="page : ${pages}" class="mb-4">
    <h2 th:text="${page.url}" class="mb-3"></h2>
    <table class="table table-striped">
      <thead>
        <tr>
          <th>Tag</th>
          <th>Contagem</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="tag : ${page.tags}">
          <td th:text="${tag.name}"></td>
          <td th:text="${tag.count}"></td>
        </tr>
      </tbody>
    </table>
  </div>
</div>

```

Fonte: Imagem própria

Código gerado na inserção e busca

Como mencionado neste documento, estamos utilizando o Hibernate para fazer as operações no banco de dados e logando o comando de execução. Ao fazer operação de consulta, temos o seguinte comando hibernate gerado:

Tabela 3 - Código Hibernate para buscar registros

```

select page0_.id as id1_0_, page0_.url as url2_0_ from tb_page page0_ inner join tb_tag
tag1_ on (page0_.id=tag1_.id_page) where page0_.url=?

```

Já na operação de inserção, temos o seguinte comando hibernate:

Tabela 4 - Código Hibernate para inserir registros

```

insert into tb_page (id, url) values (default, ?)
insert into tb_tag (id, count, name, id_page) values (default, ?, ?, ?)

```

Estes comandos equivalem as instruções SQL mencionadas na figura 7 - Script de inserção e busca no Oracle.

4. Conclusão

Diante de todo contexto e implementação explicados, temos como resultado uma aplicação que busca e identifica tags de páginas url através de uma lista de urls. Não só identifica como também busca no banco de dados análises já feitas de uma determinada url.

Segue abaixo as interfaces web resultantes:

Figura 16 - Interface web página home

Desafio NUTI

Identificar Tags

Buscar Tags por URL

Fonte: Imagem própria

Para consultar a página principal, basta executar a aplicação e acessar:

<http://localhost:8080/api/v1/desafio/home>

Quando clicamos no botão Identificar Tags, é apresentado a tela onde inserimos uma lista de urls para serem identificadas as tags das mesmas.

Figura 17 - Interface web página identificar tags

Identificar Tags

Insira as URLs (separadas por vírgula):

Enviar

Fonte: Imagem própria

Ao clicar no botão Enviar, é carregado a lista de tags de cada url inserida.

Figura 18 - Interface web página com as tags identificadas de uma ou mais urls

Tags das páginas

https://example.com/

Tag	Contagem
#root	0
html	1
head	1
title	1
meta	3
style	1
body	1
div	1
h1	1
p	2
a	1

Fonte: Imagem própria

Retornando à página principal, quando clicamos no botão Buscar Tags por URL, é carregado a página web onde podemos inserir uma URL e buscar a análise feita no banco de dados.

Figura 19 - Interface web página buscar tags

Buscar Tags de uma URL

Insira uma URL:

https://www.techonthenet.com/oracle/functions/index.php

Enviar

Fonte: Imagem própria

Quando clicamos no botão enviar, é carregado a lista de tags da url especificada.

Figura 20 - Interface web página com tags recuperadas do banco de dados

Tags das páginas

<https://www.techonthenet.com/oracle/functions/index.php>

Tag	Contagem
#root	0
html	1
head	1
meta	7
title	1
link	13
style	3
script	4
body	1
a	431
div	80
p	9
svg	9
path	13
img	17

Fonte: Imagem própria

O desafio de buscar e identificar tags em uma página web é algo interessante de se usar para avaliar candidatos e identificar seus níveis de análise e programação.

A intenção inicial que eu gostaria de fazer é de dividir o projeto em dois microserviços. Um microserviço backend baseado em uma api rest com Java e Spring Boot retornando objetos json através de endpoints e um outro microserviço frontend utilizando JavaScript e frameworks como React ou VueJS, consumindo a api rest. Porém devido as tarefas do dia, pouco tempo e falta de skill para o desenvolvimento frontend, pensei e fazer tudo na aplicação backend como um server-side, pois seria mais rápido e conciliado ao meu conhecimento de Java e o framework Spring Boot. Nunca havia trabalhado com o framework Thymeleaf, mas com a necessidade de interfaces web para apresentação dos dados, foi feito algo simples, ainda incompleto mas que atenda aos requisitos mencionados no desafio.