

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO
TOCANTINS
CAMPUS ARAGUATINS

LEANDRO GOMES CARDOSO

**CONSULTA ONLINE: Arquitetura baseada em Web *Service* como apoio ao
SIEPEX Módulo Biblioteca**

ARAGUATINS
2017

LEANDRO GOMES CARDOSO

**CONSULTA ONLINE: Arquitetura baseada em Web *Service* como apoio ao
SIEPEX Módulo Biblioteca**

Trabalho de Conclusão de Curso
apresentado à Coordenação do Curso de
Licenciatura em Computação, do Instituto
Federal de Educação Ciência e Tecnologia
do Tocantins – Campus Araguatins, para
obtenção do grau de Licenciatura em
Computação.

Orientador: Prof. Me. Rogério Pereira de
Sousa

ARAGUATINS
2017

CARDOSO, Leandro Gomes.

Consulta Online: Arquitetura baseada em *Web Service* como apoio ao SIEPEX Módulo Biblioteca / Leandro Gomes Cardoso. - Araguatins, 2017

40 f.

Trabalho de Conclusão de Curso (Graduação em Licenciatura em Computação) - Instituto Federal de Educação, Ciências e Tecnologia do Tocantins - *Campus Araguatins*, 2017.

Orientador: Prof. Me. Rogério Pereira de Sousa.

1. *Web Service*. 2. SIEPEX. 3. Spring Boot. 4. Internet

LEANDRO GOMES CARDOSO

**CONSULTA ONLINE: Arquitetura baseada em Web *Service* como apoio ao
SIEPEX Módulo Biblioteca**

Trabalho de Conclusão de Curso
apresentado à Coordenação do Curso de
Licenciatura em Computação, do Instituto
Federal de Educação Ciência e Tecnologia
do Tocantins – Campus Araguatins, para
obtenção do grau de Licenciatura em
Computação.

Orientador: Prof. Me. Rogério Pereira de
Sousa

Aprovado em ____/____/ 2017

BANCA EXAMINADORA

Prof. Me. Rogério Pereira de Sousa (Orientador)
IFTO – Campus Araguatins

Prof. Esp. Jose Mendes de Menezes Junior
IFTO – Campus Araguatins

Prof. Me. Ramasio Ferreira de Melo
IFTO – Campus Araguatins

AGRADECIMENTOS

Agradeço a Deus primeiramente por ter me dado saúde e inteligência para trilha e passar por todas as dificuldades e conseguir esta conquista.

A minha família agradeço por estar sempre comigo participando de cada momento de minha trilha, mas em especial agradeço aos meus pais, Elizete da Silva Gomes e Juarez Dias Cardoso pela grande força e conselho como também sacrifícios para alcançar meus sonhos.

Ao meu professor orientador Rogério Pereira de Sousa por ter me ajudado e colaborado para tornar a elaboração deste trabalho possível, com suas orientações, paciência e dedicação.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Tocantins – Campus Araguatins por oferecer um ambiente excelente aos alunos assim tornando realidade um sonho.

Agradeço aos colegas de curso por contribuir com os momentos que tivemos juntos.

O pessoal da TI (tecnologia da informação) do *campus* Araguatins pela oportunidade de ter ótimos momentos de estágio e poder aprender como conviver em um ambiente profissional.

A todos da banca examinadora, que fizeram presença tornando possível essa avaliação do presente trabalho.

E a todos que de uma forma direta ou indireta colaborou com a vida acadêmica e a conclusão deste trabalho.

“A primeira regra de qualquer tecnologia utilizada nos negócios é que a automação aplicada a uma operação eficiente aumentará a eficiência. A segunda é que a automação aplicada a uma operação ineficiente aumentará a ineficiência”.

(Bill Gates)

RESUMO

Este trabalho apresenta um protótipo de *web service* como forma de apoio ao SIEPEX módulo Biblioteca. *Web service* é uma forma de comunicação entre dispositivos de vários através de requisições e respostas. O problema de pesquisa é: “Como desenvolver um *Web Service* de pesquisa de uma forma simples com base no SIEPEX Módulo Biblioteca?”. Para resposta deste questionamento utilizou-se de metodologia de pesquisa experimental, para isso foi desenvolvido um protótipo funcional de *web service* com simulação do ambiente real. O objetivo geral do estudo é fornecer uma forma simplificada de pesquisa ao acervo da biblioteca utilizando-se de tecnologias para facilitar o desenvolvimento. Com a análise dos resultados pode se constatar que é possível a criação de uma forma simples o *web service*, usando Spring Boot.

Palavras-chave: *Web Service*, SIEPEX, Spring Boot, Internet.

ABSTRACT

This work presents a prototype web service as a way to support the SIEPEX Library module. Web service is a form of communication between devices of various types through requests and responses. The research problem is: "How to develop a web service to search in a simple way based on library module of SIEPEX ?". In order to answer this question, we used an experimental research methodology, for this we developed a functional prototype of web service with simulation of the real environment. The overall goal of the study is to provide a simplified form of research to the library acquis using technologies to facilitate development. With the analysis of the results it can be seen that it is possible to create a simple web service using Spring Boot.

Keywords: *Web Service*, SIEPEX, Spring Boot, Internet.

LISTA DE ILUSTRAÇÕES

Figura 1. Funcionamento do Web <i>Service</i>	13
Figura 2. SOAP estrutura de mensagem	14
Figura 3. Exemplo de XML	15
Figura 4. Exemplo de URI	16
Figura 5. Exemplo de JSON	17
Figura 6. Visualização das tabelas relacionada ao livro	22
Figura 7. Iniciando um novo projeto Spring Boot	24
Figura 8. Modelos das tabelas do banco	27
Figura 9. Repositórios de acesso ao banco de dados	29
Figura 10. Funcionamento do MVC REST	31
Figura 11. Controles do protótipo	32
Figura 12. Postman	34
Figura 13. Resposta da pesquisa de livros por título	35

LISTA DE ABREVIATURAS E SIGLAS

CORBA	Common Object Request Broker Architecture
DAO	Objeto de Acesso a Dados
DNS	Domain Name System
FTP	Protocolo de Transferência de Arquivos
HTTP	Hypertext Transfer Protocol
IDE	Ambiente Integrado de Desenvolvimento
IP	Internet Protocol
IRC	Internet Relay Chat
ITU	União Internacional de Telecomunicações
JPA	API de Persistência Java
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
PIBID	Programa Institucional de Bolsa de Iniciação à Docência
POM	Modelo de Objeto de Projeto
REST	Representational State Transfer
RMI/IIOP	Remote Method Invocation/interface over the Internet Inter-Orb Protocol
RPC	Chamada remota de procedimento
SGBD	Sistema de Gerenciamento de Banco de Dados
SGML	Standard Generalized Markup Language
SIEPEX	Sistema Integrado de Ensino, Pesquisa e Extensão
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifiers
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WEB	World Wide Web
XML	eXtensible Markup Language

SUMARIO

1. INTRODUÇÃO	10
1.1 Objetivo Geral	10
1.2 Objetivos Específicos	10
1.3 Organização do Trabalho	11
2. REVISÃO BIBLIOGRÁFICA	12
2.1 Web Services	12
2.1.1 SOAP	13
2.1.1.1 Resposta	13
2.1.2 REST	15
2.1.2.1 Endereçamento	15
2.1.2.2 Resposta	16
2.2 Java	17
2.2.1 Java EE	17
2.2.2 Apache Tomcat	18
2.2.3 Spring Boot	18
2.2.3.1 Iniciadores	18
2.2.3.2 Servidor incorporado	19
2.2.4 PostgreSQL	19
2.2.5 Apache Maven	19
3. METODOLOGIA PARA O DESENVOLVIMENTO	20
4. RESULTADOS E DISCUSSÃO	20
4.1 Implementação do protótipo	20
4.1.1 Análise do SIEPEX Módulo Biblioteca	20
4.1.1.1 Analise do banco de dados	21
4.1.2 Tecnologias utilizadas na prototipação do Web Service	22
4.1.3 Delimitação as formas de pesquisa	23
4.1.4 Configuração é codificação do protótipo	23
4.1.4.1 Spring Data JPA	29
4.1.4.2 Spring MVC	30
4.1.4.3 Finalização do protótipo	33
4.1.5 Testes de validação	33
5. CONCLUSÃO E TRABALHOS FUTUROS	36
5.1 Trabalhos futuros	36
REFERENCIAS	37

1. INTRODUÇÃO

Com o passar dos tempos o acesso à internet vem crescendo e com isso a forma de como a população interage mudou, pois na WEB se encontra meios de fazer quase tudo na vida cotidiana.

Segundo a ITU (International Telecommunication Union ou União Internacional de Telecomunicações) a porcentagem de pessoas sem acesso à internet está em 53% pesquisa do final de 2016, com isso pode se ver que temos 47% da população mundial com acesso à internet.

Com isso, vem à necessidade de ter acesso a informação em qualquer lugar com utilização de plataformas diferentes. Mas como atender a todas elas em um único lugar ? Neste contexto entra o *Web Service*, que foi criada no intuito de se ter um padrão na forma de distribuição de informações e funcionalidades, sem dependência de plataforma.

Segundo o *World Wide Web Consortium - W3C* (2004) “Os *Web Services* fornecem um meio padrão de interoperação entre diferentes aplicativos de software, executando em uma variedade de plataformas e/ou *frameworks*”.

O desenvolvimento do trabalho é feito com base na seguinte questionamento “Como desenvolver um *Web Service* de pesquisa de uma forma simples com base no SIEPEX Módulo Biblioteca ?”. Como finalidade de ter mais de um meio e/ou local de se fazer pesquisas dos dados já cadastrados na biblioteca facilitando assim o acesso a todos. Com a pesquisa feita o usuário vai saber se o livro procurado contém no acervo da biblioteca.

1.1 Objetivo Geral

O objetivo geral deste trabalho é apresentar uma forma de pesquisa simplificada ao acervo da biblioteca, bem como desenvolvimento de um protótipo de *Web Service* com o *framework* Spring Boot, que possibilita o acesso do mesmo em qualquer lugar com um dispositivo com acesso à internet.

1.2 Objetivos Específicos

- Analisar o SIEPEX Módulo Biblioteca;
- Pesquisar e validar as tecnologias necessárias para a implementação de uma *Web Service*;

- Criar um protótipo de Web Service utilizando Spring Boot;

1.3 Organização do Trabalho

Ao término deste capítulo introdutório, o trabalho está dividido nas seguintes partes: no capítulo 2 são apresentados os aspectos do web service bem como a linguagem Java e *frameworks*, o capítulo 3 mostra a metodologia utilizada no desenvolvimento, em seguida, o capítulo 4 demonstra o processo de desenvolvimento e testes do protótipo, no capítulo 5 apresenta as considerações finais e trabalhos futuros, por fim temos as referências bibliográficas.

2. REVISÃO BIBLIOGRÁFICA

2.1 Web Services

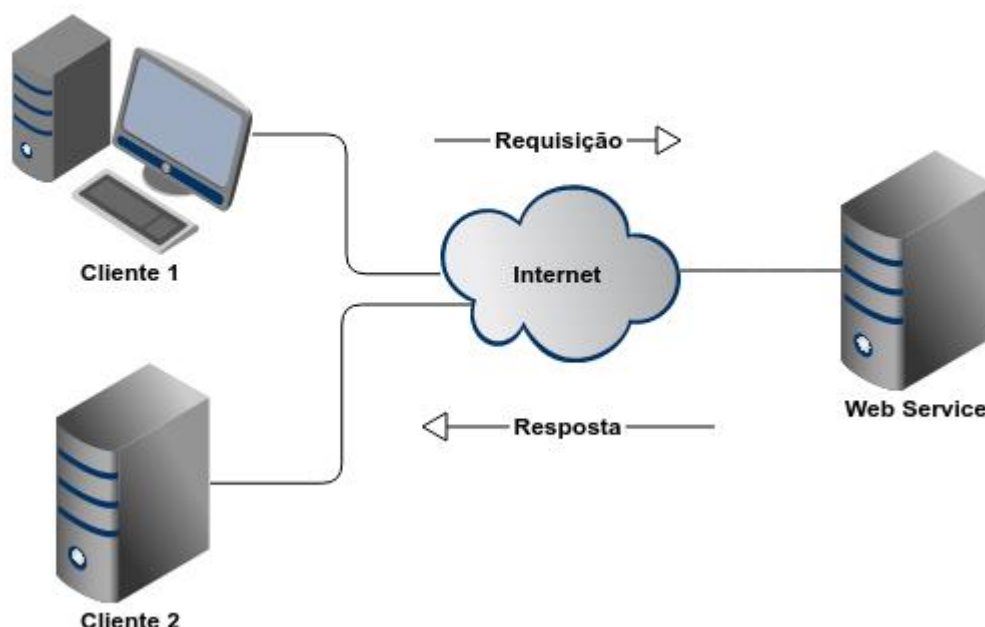
Quando se fala em interação entre sistemas de diferentes plataformas e/ou linguagens de programação, vem logo em mente os *Web Services* por se tratarem de uma tecnologia promissora nos dias atuais e por não se ter uma estrutura WEB homogênea de acesso à informação.

Um *Web Services* entre outras definições é o meio de comunicação via rede que pode comunicar-se com diferentes tipos de sistemas e recursos. Essa comunicação pode ser de vários tipos, o mais comum é pelo protocolo HTTP (Hypertext Transfer Protocol) responsável pela comunicação na WEB utilizando-se dos métodos como: get e post. Como definido pela W3C(2004) um *Web Service* é projetado para suportar a interação interoperável máquina-máquina sobre uma rede.

“A essência de *Web Services* está em permitir que diferentes aplicativos em ambientes distintos possam interoperar independente da plataforma ou linguagem em que foram escritos.” (CAMPOS, 2013).

Na prática, pode-se ter uma aplicação que se comunica com outra simplesmente para fazer um cálculo e devolver o resultado de forma que o sistema principal fique livre esperando o resultado. Essas ações disponíveis no *Web Service* são chamadas de recursos tendo como base de funcionamento requisição e respostas conforme ilustrado na Figura 1.

Figura 1. Funcionamento do Web Service



Fonte: O autor

Como pode ser observado as respostas de um Web Service podem ser para um usuário final com também a um outro servidor que consome os dados e processa para aí sim passar ao usuário.

Um web service pode ser criado usado com base em várias tecnologias (W3C, 2016). Atualmente existem dois tipos principais de arquitetura de Web Services: Simple Object Access Protocol (SOAP) e o Representational State Transfer (REST), são os mais utilizados na WEB.

2.1.1 SOAP

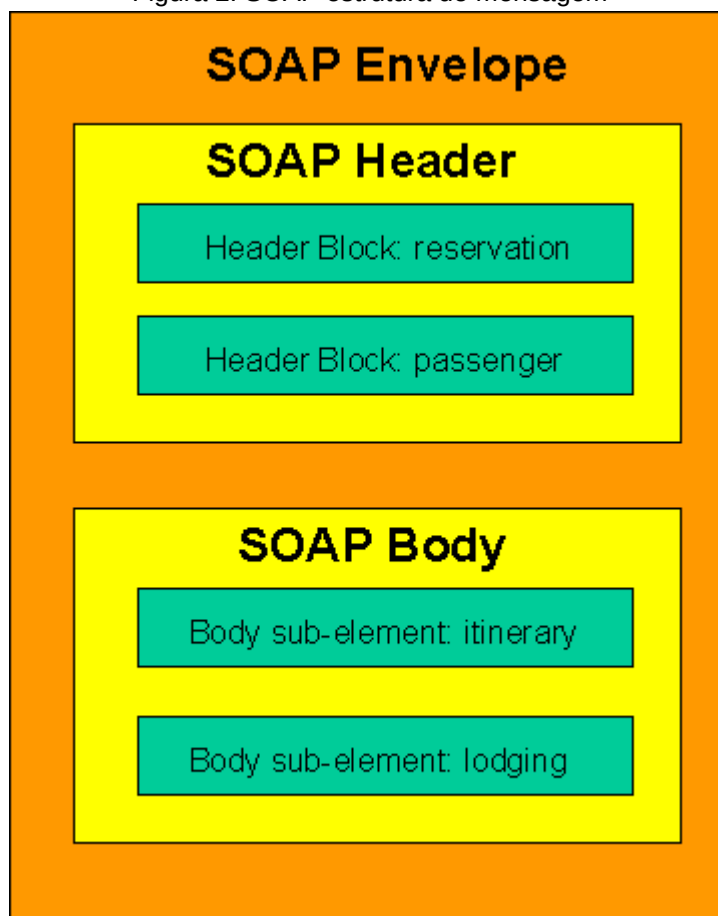
No português Protocolo Simples de Acesso a Objetos (SOAP), de acordo com a W3C(2004), é utilizado para troca de mensagens entre sistemas distribuídos de diferentes tipos. Possui uma estrutura padrão, extensível composta de empacotar e trocar mensagens com base no XML, tendo vários métodos de comunicação como HTTP, SMTP, FTP, RMI/IIOP. (W3C, 2016).

2.1.1.1 Resposta

A mensagem SOAP é formada por vários elementos como exemplificado a

seguir na Figura 2:

Figura 2. SOAP estrutura de mensagem



Fonte: <https://www.w3.org/TR/soap12-part0/primer-figure-1.png>

- Envelope: Onde contém a definição, que é uma mensagem transportada pelo SOAP.
- Cabeçalho (Header): Bloco contendo todas as informações necessárias para o transporte da mensagem como: autenticação, contexto de transação.
- Corpo (Body): Contém a parte mais importante, o conteúdo a ser entregue e processado que vem no formato XML.

O XML é um formato de texto simples recomendado pela W3C uma evolução do SGML utilizada com a SOAP para definir o padrão de Web Services da W3C, como desvantagem de seu uso pode ser citada a forma de sua estrutura que é um pouco mais complexa que algumas outras existentes.

Figura 3. Exemplo de XML

```
1 <livros>
2   <livro>
3     <titulo>titulo 1</titulo>
4     <autor>autor 1</autor>
5   </livro>
6   <livro>
7     <titulo>titulo 2</titulo>
8     <autor>autor 2</autor>
9   </livro>
10 </livros>
```

Fonte: O autor

2.1.2 REST

REST no português (Transferência de Estado Representacional) é um estilo de arquitetura para projetar aplicativos em rede (ELKSTEIN, 2016). Sendo completamente baseada no protocolo HTTP, assim mais simples que outros como CORBA, RPC e SOAP.

Este estilo foi primeiramente proposto por Roy Fielding na sua tese de doutorado, sendo também um dos autores do HyperText Transfer Protocol (HTTP). Por isso tem sua base estruturada no HTTP. As Trocas de mensagens utilizam-se de requisições HTTP para manipulação da informação (ELKSTEIN, 2016).

Conceito importante encontrado dentro do REST é o recurso. Recursos são conjuntos de informações que podem ser nomeada sendo que trafegam por meio da Uniform Resource Identifiers (URI). Para Fielding (2000) qualquer tipo de informação que pode ser nomeada é possível ser um recurso como imagens e documentos.

2.1.2.1 Endereçamento

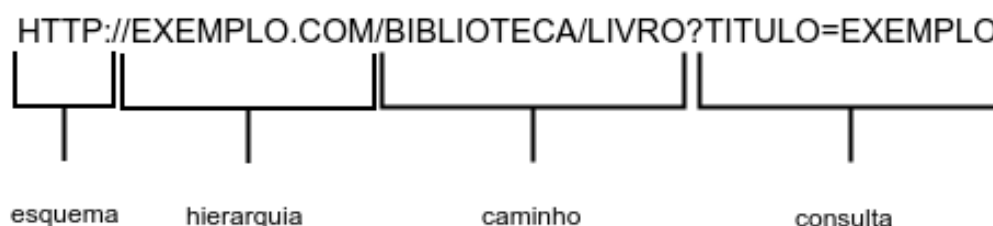
URI é um elemento simples na WEB, formada pela junção da *Uniform Resource Locator* (URL) é *Uniform Resource Name* (URN). REST define o termo recurso para padrão URI bem como semântica geral da interface genérica para manipulação de recursos pelas suas representações. (Fielding, 2000)

A RFC2396 descreve uma *Uniform Resource Identifier*(URI) como uma sequência de caracteres para identificar um recurso abstrato ou físico, esse identificador usando URL ou URN .Conforme Figura 4.

A composição de uma URI é basicamente:

- Nome do esquema identifica o protocolo (por exemplo: FTP, HTTP, HTTPS, IRC);
- Parte Hierárquica, referência ao DNS (*Domain Name System*) real da máquina (por exemplo: nome do domínio ou endereço de *Internet Protocol* - IP);
- Caminho faz parte de um conjunto de sequência separadas por uma barra (“ / ”);
- Consulta, contém informações adicionais que geralmente é antecedido por um interrogação(“ ? ”).

Figura 4. Exemplo de URI



Fonte: O autor

2.1.2.2 Resposta

Em um *web service* do tipo REST, pode-se ter vários tipos de formatos de respostas, pois o mesmo é baseado por completo no HTTP. O formato mais comum utilizado na comunicação entre sistemas pelo REST é o JSON (*JavaScript Object Notation*) que é um formato simples, leve na interação de dados, de fácil compreensão, sendo também independente de linguagens de programação usada, para tanto o Json é uma linguagem de transmissão de dados ideias, por ser leve e simples .(Json, 2016). Conforme Figura 5.

Figura 5. Exemplo de JSON

```
1 [
2   {
3     "id": 134,
4     "autor": {
5       "nome": "ACOPIARA, Moreira de"
6     },
7     "titulo": "titulo",
8     "subtitulo": "subtitulo",
9     "cdd": null
10  },
11  {
12    "id": 135,
13    "autor": {
14      "nome": "ACOPIARA, Moreira de"
15    },
16    "titulo": "titulo",
17    "subtitulo": "subtitulo",
18    "cdd": null
19  }
20 ]
```

Fonte: O autor

2.2 Java

Java não é uma simples linguagem de programação, e sim uma plataforma de desenvolvimento que inclui ferramentas livres e privadas que funcionam basicamente nos principais tipos de aplicações como: Incorporados com Java Embedded, Desktop com Java SE, Web com Java EE e Computação em Nuvem com Java Cloud.

Disponível para todos os ambientes de desenvolvimento como: Windows, Linux, Mac OS, pois utiliza uma forma de comunicação indireta por meio de uma máquina virtual ou JVM(Java Virtual Machine), considerada uma das plataformas mais usadas para desenvolvimento. Segundo a Oracle(2016) tem-se mais de 9 milhões de desenvolvedores em todo o mundo. (COSTA, 2008).

2.2.1 Java EE

Java EE (Java *Enterprise Edition*) é destinado a aplicações web tendo um grande conjunto de especificações destinado a padronizar a forma de desenvolvimento atualmente se encontra na versão 7. Tendo como objetivo simplificar a forma de desenvolvimento que hoje em dia é bastante complexa tendo várias regras de negócio.

De acordo com a Caelum (2016), Java EE é uma série de especificações

dando uma fórmula de como deve ser implementado cada parte do serviço do software.

2.2.2 Apache Tomcat

Tomcat é um servidor de aplicações Java EE, desenvolvido pela Fundação Apache. Sua principal característica é a centralização na linguagem JAVA, seguindo as especificações proposta pelo Oracle para o Java EE. Há várias formas de se utilizar o Tomcat que pode ser isoladamente ou em conjunto com outro servidor de páginas estáticas com o Apache. O Tomcat é uma implementação de código aberto das tecnologias Java Servlet, JavaServer Pages, Java Expression Language e Java WebSocket, (Tomcat, 2016)

2.2.3 Spring Boot

O Spring Boot é um projeto idealizado pela Pivotal, empresa responsável pela criação e mantimento de vários frameworks e projetos do mundo Java. O Boot surgiu como facilitador no desenvolvimento de aplicações Web Java com a base nos *frameworks* Spring. O *framework* tem sua base no já conhecido Spring é acumula todos os benefícios, diminuindo a complexidade dessa estrutura de configuração, (Silva, 2015).

De acordo com a Spring Boot, 2016, os objetivos principais são:

- Ter uma forma rápida e acessível de iniciar o desenvolvimento com Spring
- Executar com os requisitos padrões
- Fornecimento de recursos comuns a maior parte dos projetos Java Web (servidores incorporados, segurança, métricas)
- Sem geração de código ou configuração de XML

As aplicações podem ser criadas pode ser no formato autocontido com jar e então no formato padrão war, podendo ser executado em por meio de scripts.

2.2.3.1 Iniciadores

O Spring Boot a baseado em iniciadores que são descritores de dependências que se pode incluir a seu projeto. Já se tem boa parte dessas tecnologias Spring com iniciadores que você pode precisar. Os iniciadores contêm muitas das dependências que você precisa para obter um projeto em execução

rapidamente, (Spring Boot, 2016).

Alguns exemplos de iniciadores:

- *spring-boot-starter-web*(inclui todos os *frameworks* Spring necessários para trabalhar web)
- *spring-boot-starter-data-jpa*(inclui meios de acesso a bancos de dados)

2.2.3.2 Servidor incorporado

O servidor de aplicação no Spring Boot é Incorporado, sendo necessário somente iniciar a aplicação para funcionar, pois o mesmo já vem com configurações predefinidas. Tem suporte a servidores incorporado: Tomcat, Jetty é Undertow, (Spring Boot, 2016).

2.2.4 PostgreSQL

PostgreSQL é um sistema de gerenciamento de banco de dados completamente livre sendo bastante utilizado com o Java para desenvolvimento de aplicações. O PostgreSQL é um poderoso sistema de banco de dados objeto-relacional de código aberto, (POSTGRESQL, 2016).

2.2.5 Apache Maven

Maven é um gerenciador é compressor de projetos de software, tendo sua base no conceito Modelo de Objeto de Projeto (POM). Sendo que pode ser utilizado na gerência é ou compilação de projeto, sua funcionalidade é simplificar no gerenciamento de dependências nos projetos Java facilitando seu desenvolvimento e compartilhamento. Não sendo necessário fazer download individual de cada biblioteca a ser utilizada que era muito comum antes destes tipos de gerenciadores. Apache Maven é uma ferramenta que visa ter um padrão na construção dos projetos Java, definido de uma forma clara o que é projeto é suas dependências, (Maven, 2016).

3. METODOLOGIA PARA O DESENVOLVIMENTO

O projeto tem como objetivo a criação de um protótipo de *web service* para utilização no sistema bibliotecário já existente no Instituto Federal de Educação Ciência e Tecnologia do Tocantins- *Campus* Araguatins, tendo uma visão de facilitar o acesso dos usuários a pesquisa no acervo da biblioteca.

Foi apresentado primeiramente o que é um *web service*. Logo em sequência foram mostrados os tipos de *web service*, assim, mostrando o porquê da escolha do REST como modelo.

Na modelagem e execução do protótipo foi utilizando de tecnologias gratuitas, como plataforma de desenvolvimento IDE (*Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento) Eclipse Neon versão Java EE, linguagem de programação Java, SGBD PostgreSQL similar ao encontrado no SIEPEX Módulo Biblioteca, para testes de pesquisa utilizou-se uma aplicativo Postman ele possui uma versão gratuita multiplataforma executado por meio de uma aplicativo do Google Chrome, com ele pode-se fazer requisições do HTTP de todos os tipos com GET e POST e facilita a leitura da resposta.

4. RESULTADOS E DISCUSSÃO

4.1 Implementação do protótipo

Neste capítulo é apresentada a análise do SIEPEX Módulo Biblioteca existente, como também o processo no desenvolvimento do *web service* de pesquisa e testes de validação.

4.1.1 Análise do SIEPEX Módulo Biblioteca

O SIEPEX Módulo Biblioteca é um software existente no Instituto Federal de Educação, Ciência e Tecnologia do Tocantins *campus* Araguatins, desenvolvido por acadêmicos do curso de Licenciatura em computação. O projeto surgiu após um trabalho da disciplina de Análise II, nele o objetivo era fazer o levantamento de requisitos e fazer toda a modelagem na UML. Mas com o tempo os acadêmicos responsáveis resolveram dar prosseguimento no projeto assim desenvolver o

sistema com apoio dos professores coordenadores do PIBID (Programa Institucional de Bolsa de Iniciação à Docência).

A necessidade inicial do projeto era que os funcionários da biblioteca fizessem o gerenciamento do acervo e que alguns alunos e professores pudessem fazer consulta de livros cadastrados no sistema só que com uma pequena limitação, o sistema é acessado localmente por ser uma aplicação desktop não acessível fora da biblioteca.

Mas após um tempo surgiu uma nova necessidade de ter acesso à pesquisa de livros no acervo em qualquer lugar por meio da web, essa é uma funcionalidade bastante importante para os professores que quando quiser montar seu material vai poder saber de onde estiver quais os materiais de apoio na biblioteca para suas disciplinas.

O sistema da biblioteca está estruturado da seguinte forma, um servidor de banco de dados no caso PostgreSQL que visa ter uma base única para gerenciamento do acervo e do outro lado sistema desktop desenvolvido em Java SE com toda estrutura para que mais de um usuário tenha acesso a funcionalidade do banco ao mesmo tempo.

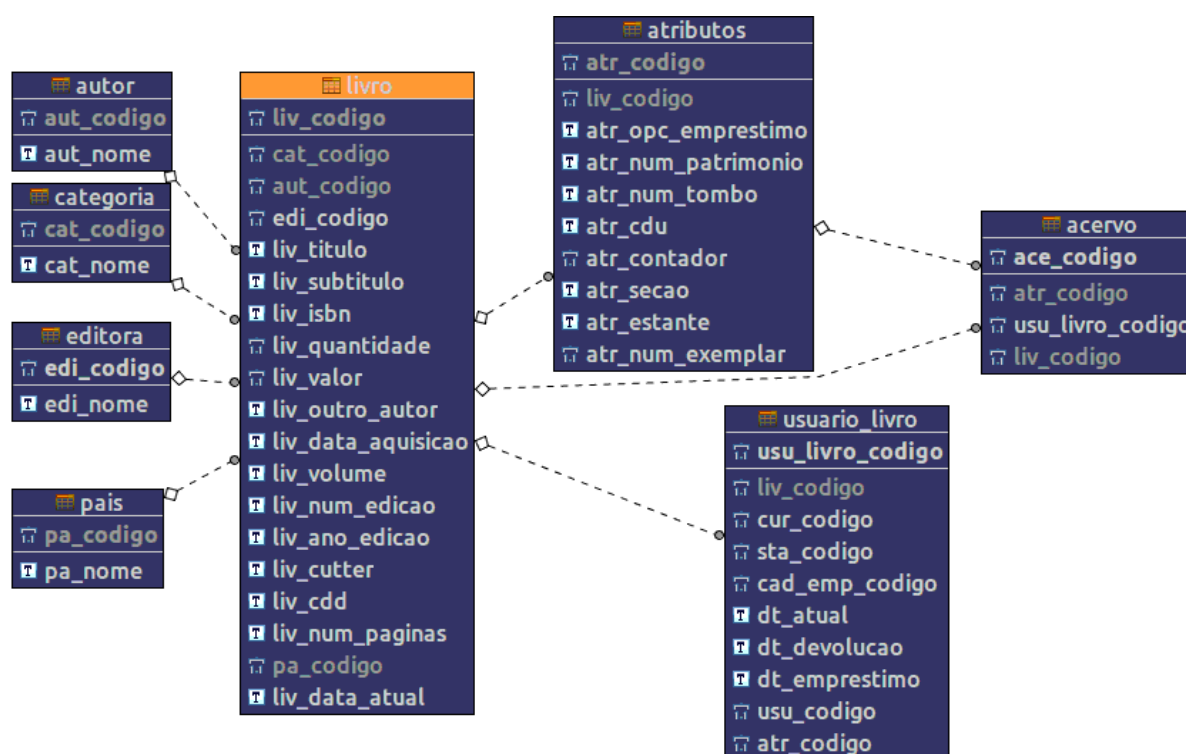
Com base nessas informações a parte importante para o protótipo é o banco de dados, pois o mesmo pode ser acessado de uma forma direta com isso foi feito um backup com a estrutura do banco e mesmo foi usado como base para o desenvolvimento do protótipo.

4.1.1.1 Análise do banco de dados

Antes de partir para o desenvolvimento foi feita uma análise dos dados que são necessário para construção do web service com isso utilizou-se o programa DBeaver um gerenciador universal de banco de dados multiplataforma no modo visual facilitando a compreensão dessas tabelas.

A seguir um exemplo de tabelas que se pode ver com DBeaver na Figura 6.

Figura 6. Visualização das tabelas relacionada ao livro



Fonte: O autor

Há várias tabelas dentro do banco de dados, mas foi definido que no momento a necessidade é que tenha-se uma forma de pesquisar por livros, autores, categorias, editora, que estão sendo demonstrado na Figura 6.

4.1.2 Tecnologias utilizadas na prototipação do Web Service

Para fazer o desenvolvimento foi definido primeiramente que o web service seria feito com a linguagem Java. Escolhido pela facilidade no desenvolvimento de aplicativos e ter mais facilidade de ter ajuda por parte dos professores.

Mas com a linguagem definida vem uma questão desenvolver puramente ou fazer utilização de algum *framework* para facilitar no momento de criação, com isso foi feito um levantamento das principais tecnologias para se criar um web service do tipo REST chegando-se a duas alternativa de *framework* uma o Jersey e outro Spring Boot.

Após análise ficou claro as grandes facilidades que o *framework* Spring Boot traz por já ter uma configuração inicial que pode ser utilizado em produção algumas coisas como servidor embutido, e ter complementos chamado de iniciadores que ao se incluir configurar todo o projeto restando apenas executar.

O que chamou atenção e que com somente dois iniciadores já se tem uma estrutura completa para desenvolvimento de um *web service*. Eles são o de trabalhar com a web fornecendo toda a base necessária para se ter um *web service* e o outro responsável por toda a manipulação de dados.

Para a codificação foi escolhido a IDE Eclipse Neon Java EE ele é um software livre facilitador da programação por trazer recurso como *autocomplete* que dar sugestão de código, sendo também recomendado pelos desenvolvedores do Spring.

Na questão dos testes para não ter que criar um cliente para *web service* será utilizado um aplicativo gratuito em sua versão gratuita encontrado no Google Chrome chamado Postman fornecendo a base necessária para fazer testes com o protocolo HTTP.

4.1.3 Delimitação as formas de pesquisa

Quando se está trabalhando com a arquitetura REST as URI tem importância pelo conceito que uma URI está ligado a um recurso. Para que o protótipo siga este modelo serão definidos quatro recursos que serão mapeados, que são: Livro, Autor, Categoria e Editora, assim tendo uma melhor separação.

Ao se pesquisar, por exemplo, por um título de livro então teremos a seguinte URI `http://localhost/livros/` seguido do termo a ser pesquisado e por último tipo de pesquisa a ser realizada ficando no final assim `http://localhost/livros/titulo?tipo=titulo`. Para o livro por ter maior quantidade de informações foi delimitado três tipos de pesquisas por título, subtítulo e isbn.

4.1.4 Configuração e codificação do protótipo

Após toda definição da parte não funcional do protótipo, chegou o momento da configuração e codificação. Para isso o primeiro passo é a criação de um projeto Spring Boot, no momento há diferentes formas como: utilizando-se uma IDE com plugin do Spring e outra a mais recomendada por não está ligado diretamente a IDE que é pelo site de iniciação de novas aplicações acesso por meio `http://start.spring.io/`.

Dentro do site tem várias opções uma dessa mais importante é a forma de gerenciamento que pode ser pelo Maven ou Grandle, que são gerenciadores de projeto da plataforma Java. Com isso pode se definir as dependências dos projetos

como no exemplo da Figura 7.

Figura 7. Iniciando um novo projeto Spring Boot

The screenshot shows the Spring Initializr web application interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there's a header "Generate a Maven Project with Spring Boot 1.4.3". The main content is divided into two columns: "Project Metadata" and "Dependencies".

Project Metadata:

- Artifact coordinates:**
 - Group:**
 - Artifact:**
 - Name:**
 - Description:**
 - Package Name:**
 - Packaging:**
 - Java Version:**

Dependencies:

- Add Spring Boot Starters and dependencies to your application**
- Search for dependencies:**
- Selected Dependencies:**
 - Web JPA PostgreSQL

Fonte: O autor

Com a configuração completa tem que se gerar um arquivo compactado é importá-lo na IDE. Esse processo pode ser diferente de acordo com a IDE utilizada no Eclipse utiliza-se do tem que se extrai o projeto para uma pasta de sua preferência acesse a IDE no menu *File -> Import -> Maven -> Existing Maven Project*, com essa opção selecione a pasta em que contendo o projeto após finalizar é eclipse vai fazer toda a configuração do projeto com base no arquivo pom.xml.

A arquivo pom.xml é o responsável pela padronização de projeto com que tenha o maven como gerenciado como no exemplo a seguir de um fragmento pom.xml gerado:

1. `<parent>`
2. `<groupId>org.springframework.boot</groupId>`
3. `<artifactId>spring-boot-starter-parent</artifactId>`
4. `<version>1.4.2.RELEASE</version>`
5. `<relativePath />`
6. `</parent>`
7. `<properties>`
8. `<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>`
9. `<project.reporting.outputEncoding>`
 UTF-8

```

        </project.reporting.outputEncoding>
10.    <java.version>1.8</java.version>
11. </properties>
12. <dependencies>
13.    <dependency>
14.        <groupId>org.springframework.boot</groupId>
15.        <artifactId>spring-boot-starter-data-jpa</artifactId>
16.    </dependency>
17.    <dependency>
18.        <groupId>org.springframework.boot</groupId>
19.        <artifactId>spring-boot-starter-web</artifactId>
20.    </dependency>
21.    <dependency>
22.        <groupId>org.postgresql</groupId>
23.        <artifactId>postgresql</artifactId>
24.        <scope>runtime</scope>
25.    </dependency>
26.</dependencies>

```

Alguma das configurações são muito importante como relatado abaixo:

- Da linha 1 a 6 é definida a versão em que o Spring Boot trabalhar é também como todas outras dependências Spring está ligada a ela.
- Da linha 7 a 11 é configurado as propriedades do projeto com Encoding que é o formato dos caracteres de resposta, também define a versão do Java assim criando um padrão.
- Da linha 12 a 26 onde encontra-se todas as dependências relacionadas ao projeto.
- Da linha 13 a 16 é referenciado a dependência de um iniciador referente ao JPA(Java Persistence API ou API de Persistência Java) que gerencia todas ações de banco de dados.
- Da linha 17 a 20 é referenciado a dependência de um iniciador para web controla toda parte de web como requisição e resposta.
- Da linha 21 a 25 tem a dependência do SGBD PostgreSQL bem como a definição que uma dependência utilizada em tempo de execução.

Com o projeto importado já se tem uma base completa para o desenvolvimento, com todos os arquivos necessário podendo ser até executado que levantará o servidor normalmente.

O arquivo principal de uma aplicação do Spring Boot é a que contém a anotação `@SpringBootApplication`, a classe que a contenha será responsável pela execução do sistema de com algumas modificações se necessário.

```

1. @SpringBootApplication
2. public class BibliotecaApiApplication {
3.     public static void main(String[] args) {
4.         SpringApplication.run(BibliotecaApiApplication.class, args);
5.     }
6. }
```

- Na linha 1 tem uma anotação que indica que esse é uma classe principal do projeto, O `@SpringBootApplication` é equivalente a `@Configuration`, `@EnableAutoConfiguration` e `@ComponentScan`. (Spring, 2016). O seja ativa toda a configuração automática é indica que é a classe principal.
- Na linha 3 é indicado que esse é o primeiro método a ser executado na classe, pois o mesmo tem o nome de `main`.
- Na linha 4 é chamado o método `run` responsável por iniciar a aplicação, ele recebe dois parâmetros a classe principal que no caso está adicionando ela mesma e o segundo e os argumentos que pode ser passado no momento da execução.

Agora é necessário configurar o acesso ao banco de dados para que o aplicativo já comece com uma conexão estabelecida, essa e outras configurações podem ser feitas no arquivo `application.properties` encontrado dentro da pasta `resource` da aplicação.

Ele é um arquivo de propriedades onde pode ser feita alterações dos valores padrões que o *framework* traz. Algumas destas configurações estão relacionadas a banco de dados que serão configurados como demonstrado abaixo.

1. `spring.jpa.database=POSTGRESQL`
2. `spring.datasource.platform=postgres`
3. `spring.jpa.hibernate.ddl-auto=none`
4. `spring.database.driverClassName=org.postgresql.Driver`

5. `spring.datasource.url=jdbc:postgresql://localhost:5432/controlebiblioteca`
6. `spring.datasource.username=postgres`
7. `spring.datasource.password=postgres`

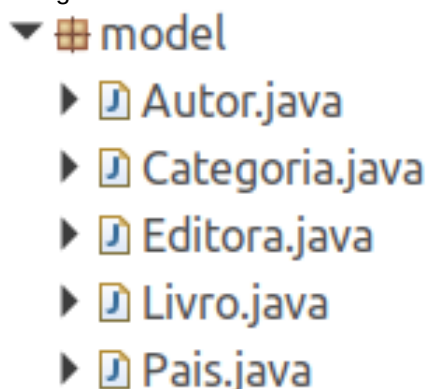
Acima e listado a configuração básica para banco de dados PostgreSQL tendo as seguintes funções:

- Linha 1 define o SGBD para postgresql.
- Linha 2 define a plataforma de dados.
- Linha 3 define qual tipo de ação pode ser executado em banco, essa é uma configuração importante pois se estiver com um banco de dados já em produção com dados reais se essa configuração estiver errada pode causar a perda de todos os dados, para não executar esses tipo de ação definir “none” e seja nenhuma.
- Linha 4 indica o drive de conexão ao banco.
- Linha 5 define a URL de conexão ao banco.
- Linha 6 define usuário do banco.
- Linha 7 define a senha do banco.

Com a conexão ao banco de dados estabelecida o próximo passo é criar as classes de modelos também conhecida como entidade que faz a ligação direta com as tabelas em banco. Um modelo representa uma tabela do banco de dados com todos os campos relacionados, contendo anotações que indica o que ele e em banco.

Para se trabalhar com as tabelas em banco são necessários modelos e para o protótipo foi definido 5 tabelas a serem usadas, conforme representado na Figura 8.

Figura 8. Modelos das tabelas do banco



Fonte: O autor

As configurações das classes são feitas por meio de *annotation* que definem o que cada campo representa, de acordo com o exemplo a seguir na demonstração de uma dessas classes.

```

1. @Entity
2. @Table(name="livro")
3. public class Livro implements Serializable{

4.     private static final long serialVersionUID = 1L;

5.     @Id
6.     @Column(name="liv_codigo")
7.     private int id;

8.     @ManyToOne
9.     @JoinColumn(name="cat_codigo")
10.    @JsonIgnore
11.    private Categoria categoria;

12.    @Column(name="liv_titulo")
13.    @JsonInclude(Include.NON_NULL)
14.    private String titulo;

```

Basicamente o que está sendo configurado acima é:

- Linha 1: declarando que essa classe é uma entidade referente ao banco.
- Linha 2 : dá –se o nome da tabela que está vinculado com a classe.
- Linha 3: diz que esse é uma classe serializada podendo ser trafegada em rede.
- Linha 5: indica que é um campo primário da tabela em banco o identificador.
- Linha 6 : vincula com o nome que está no banco.
- Linha 7 : define o tipo é nome do campo.
- Linha 8: mostra que essa classe está vinculada com categoria em uma relação de um livro para muitas categorias.
- Linha 9: indica qual o nome do campo responsável pela vinculação das tabelas no caso cat_codigo que será salvo no livro.
- Linha 10: fala ao Spring que é para ignorar se estiver sendo feito um retorno de dados do tipo JSON.
- Linha 11: cria uma variável privada do tipo Categoria.
- Linha 13: fala ao Spring que é para incluir o valor no retorno somente se não for nulo ou null.

Cada um dos modelos é responsável por uma tabela em banco. Mas não executa as ações em banco para isso existem alguns padrões muito conhecido e utilizado com DAO (*Data Access Object* ou Objeto de Acesso a Dados) e o recomendado pelo Spring Data JPA padrão repository (repositório). Na Figura 9 temos os repositórios dos modelos que vão executar ações em banco de dados.

Figura 9. Repositórios de acesso ao banco de dados



Fonte: O autor

4.1.4.1 Spring Data JPA

Esse é um *framework* responsável por simplificar o acesso a tecnologias de armazenamento de dados. Podendo ser relacional (Postgres, MySQL, etc) ou não (MongoDB, Redis). Responsável pelo controle de transação em banco onde o desenvolvedor não necessita ter preocupação em abrir e fechar transações deixando mais livre trabalhando em cima da lógica.

Além disso, facilitar a ter um padrão por trazer algumas funcionalidades prontas com *save*, *delete*, *findOne* como também meios de criar *queries* (consulta em banco) por meio de nomes de métodos como por exemplo no repositório de autor:

1. `List<Autor> findByNomeContaining(String nome);`
2. `Autor findByIdAndNome(int id, String nome);`

Acima tem-se dois exemplos de como fazer queries com base no nome. Na linha 1 é feito uma busca pelo nome do autor que contenha o nome passado, na 2 é feito busca pelo id e nome.

Os repositórios para essas funcionalidades estende-se de outra interface genérica *JpaRepository* que define acesso ao banco. A interface repositório do Spring Data JPA cria acesso aos dados tendo facilidade de criar queries automáticas baseadas no nome, (Spring, 2016). Como exemplo no arquivo *LivroRepository.java*.

```
1. public interface LivroRepository extends JpaRepository<Livro, Integer>{  
2.     List<Livro> findByTituloContaining(String titulo);  
3.     List<Livro> findByAutorNomeContaining(String nome);  
4. }
```

Esse é um exemplo de uma interface repositório com algumas queries criadas automáticas com base no nome. Podendo-se compreender da seguinte forma:

- Linha 1 estende da interface JpaRepository passando para o mesmo o modelo e tipo da chave primária (id).
- Linha 2 é declarando um método que significa buscar todos os livros que tenha o título que contenha no seu título uma parte do valor passado.
- Linha 3 é declarando um método com significado de buscar todos os livros que tenha o nome do autor que contenha uma parte do valor passado.

Essa é uma forma bem simples de criar queries, pois não está vinculado diretamente ao SQL e/ou banco de dados o próprio Spring Data JPA se encarrega de fazer toda a conversão para o banco configurado.

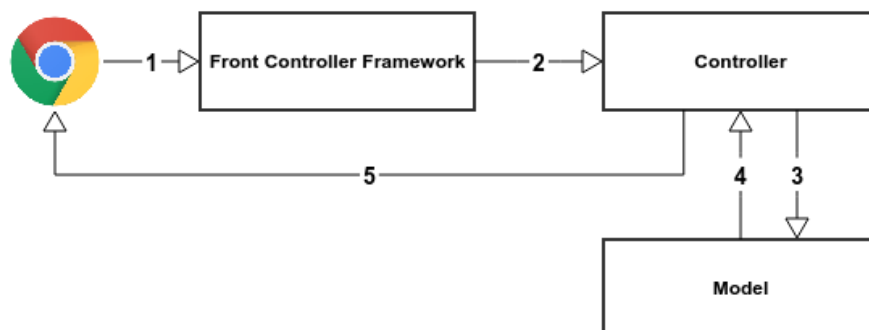
Todos os modelos são repositórios prontos o próximo passo é a criação dos controles.

4.1.4.2 Spring MVC

Entre os projetos Spring, o Spring MVC é um dos que mais facilita no desenvolvimento Java web, pois pode ser utilizado em pequenos e grandes projetos já que ele traz em si uma separação de responsabilidades bem como todo o controle dessas requisições e respostas e como um centro de toda a aplicação. O MVC é acrônimo de Model, View e Controller, esse um padrão de projeto bastante utilizado em várias tecnologias de programação.

Como se dá o funcionamento do Spring MVC, basicamente é acessado pela URI e o MVC se encarrega do funcionamento com apresentado na Figura 10.

Figura 10. Funcionamento do MVC REST



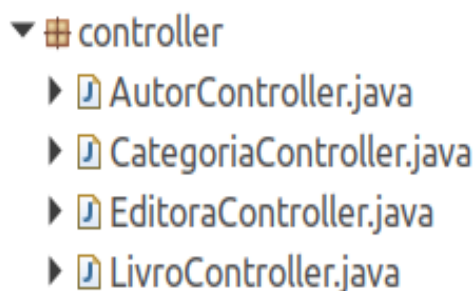
Fonte: O autor

Na figura 10, tem-se a demonstração dos passos a ser executado pelo Spring MVC com uma diferença por ser REST e não ter a camada de view(visão). Os passos são de acordo da seguinte forma:

- Passo 1 usuário fazer requisição que é controlada pelo front controle que define que é responsável por responder a requisição.
- Passo 2 a requisição é mandada ao controle (*controller*) da aplicação que verificará a ação a ser executada.
- Passo 3 faz acesso ao banco de dados como base no modelo (*model*) por meio do repositório.
- Passo 4 retorna os dados do banco para o controle (*controller*) que faz o tratamento necessário.
- Passo 5 devolve ao usuário a resposta no caso do REST o mais comum e no formato JSON.

No REST os recursos são comparados a controles no Spring MVC, para o protótipo será criado 4 controles representando os recursos como na Figura 11.

Figura 11. Controles do protótipo



Fonte: O autor

O controle no padrão MVC é responsável pela ligação dos modelos com a visualização. É no caso do REST que retorna os dados diretamente ao cliente, mas como isso ocorre dentro do controle para fazer o controle do livro no protótipo, utilizou-se o seguinte código.

```

1. @RestController
2. @RequestMapping("/livros")
3. public class LivroController {

4.     @Autowired
5.     private LivroRepository livroRepository;

6.     @GetMapping("/{search}")
7.     public ResponseEntity<?> search(@PathVariable(value = "search") String search,
8.                                     @RequestParam(value = "tipo", required = true) String tipo) {
9.         List<Livro> livros;
10.         if (tipo.equals("titulo")) {
11.             livros = livroRepository.findByTituloContaining(search);
12.             return ResponseEntity.status(HttpStatus.OK).body(livros);
13.         } else if (tipo.equals("subtitulo")) {
14.             livros = livroRepository.findBySubtituloContaining(search);
15.             return ResponseEntity.status(HttpStatus.OK).body(livros);
16.         }
17.         return ResponseEntity.notFound().build();
18.     }

```

As configurações da classe `LivroController` tem as seguintes partes importantes:

- Linha 1 define que esse é um controle do tipo REST que não necessita de uma visualização (*view*).
- Linha 2 indica como vai ser acessado a classe por meio do caminho (*path*) `livros/`.
- Linha 4 e 5 carrega de forma automática uma instância do repositório de livro por meio da anotação `@Autowired`.
- Linha 6 define que todas as requisições do tipo HTTP GET para o caminho `/livros` passando um valor a ser buscado vai ser respondido por esse método.
- Linha 7 recebe o valor é o parâmetro necessário para execução do método. Como também configura o tipo de resposta para o tipo `ResponseEntity<?>` podendo devolver qualquer tipo de modelo(entidade).
- Linha 8 cria uma lista do tipo modelo livro.
- Linha 9 a 15 faz verificações para identificar a forma de pesquisa a ser realizada.
- Linha 10 insere na lista o valor retornado do método de pesquisa.
- Linha 11 retorna uma resposta status da requisição e também seu corpo.
- Linha 16 retorna que o recurso não foi encontrado.

4.1.4.3 Finalização do protótipo

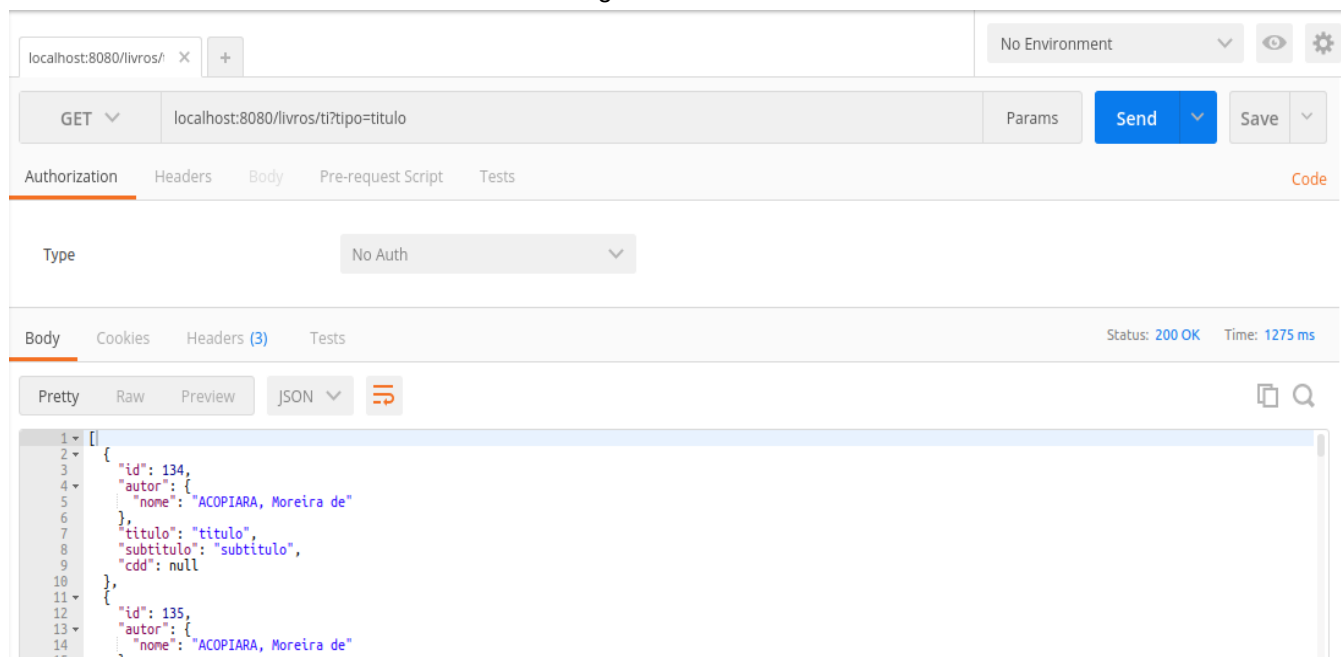
Após ter criado todas as classes de controle, modelo e repositório, tem-se uma estrutura completa de um Web Service da arquitetura REST de uma forma simples aproveitando todos os benefícios do Spring Boot.

4.1.5 Testes de validação

Testes são formas simples de averiguar se há algum problema na implementação dos protótipos antes de entrar em fase de produção. Para esse projeto utilizou-se Postman como meio de testes pela sua facilidade e por ser multiplataforma.

O Postman basicamente oferece uma interface mais limpa é simples onde se pode fazer as requisições é ver as respostas de diversas formas, também podendo salvar as formas de pesquisa e refazer de uma forma rápida. Conforme Figura 12.

Figura 12. Postman

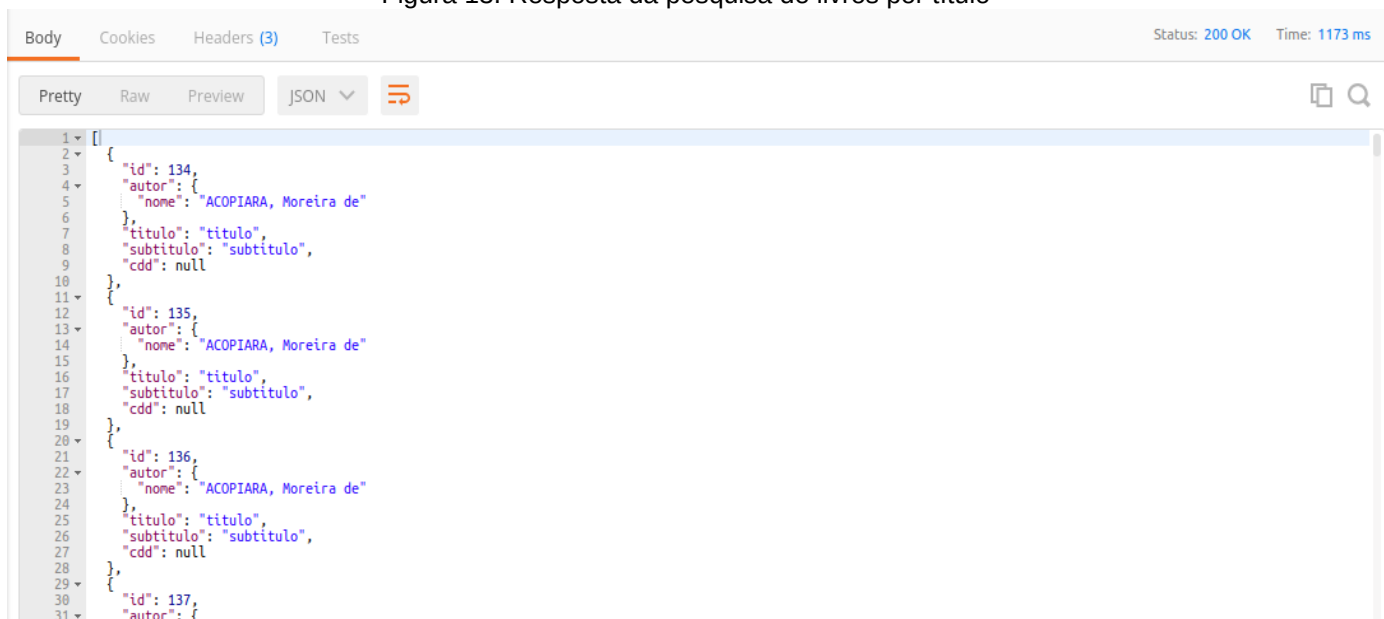


Fonte: O autor

A base para os testes foram os recursos ou controles do protótipo bem como todos os métodos de busca aceita por cada um, no intuito de ter certeza dos dados de resposta estavam corretos foi conferido com os valores do banco de dados.

Um exemplo de resposta pode se ver na Figura 13.

Figura 13. Resposta da pesquisa de livros por título



Fonte: O autor

Alguns dados que importante na análise é o status que foi devolvido no caso deste teste foi 200 OK, o formato de resposta JSON e o tempo de resposta de cada requisição.

5. CONCLUSÃO E TRABALHOS FUTUROS

Ao analisar o SIEPEX módulo Biblioteca, obteve-se a estrutura necessária que as classes de modelo de banco de dados teriam que ter, assim vinculando ao sistema atual com os dados.

A partir da pesquisa e validação das tecnologias com base na qualidade e facilidade de utilização da mesma, optou-se pela escolha de *framework* como padrão para o desenvolvimento do protótipo. Este por sua vez, entrega ao desenvolvedor uma maneira simples e objetiva para construção de aplicações baseadas na arquitetura REST.

A prototipação não ofereceu dificuldades, por ser totalmente baseado no *framework* Spring Boot, e com isso pode-se construir um *web service* REST, utilizando-se padrões de projetos.

Com tudo, obteve-se uma forma simplificada de pesquisa ao acervo bibliográfico utilizando o protótipo de *web service*, resultado final deste trabalho, podendo ser comprovado com base em testes de validação por qualquer dispositivo com acesso à internet.

5.1 Trabalhos futuros

Para futuros trabalhos, sugere-se baseado nos resultados obtidos, a implementação de uma complementação do sistema de forma que ele tenha além da pesquisa, ações que possam ser feitas como a inclusão, alteração e exclusão de dados.

REFERENCIAS

Caelum. et al. **Java para Desenvolvimento Web**. Disponível em: <https://www.caelum.com.br/apostila-java-web/>. Acesso em: 29 dez. 2016.

CAMPUS, J. **RESTMB: API RESTful para Android**. 68p. Monografia. Centro Universitário Eurípides de Marília. Marília, 2013.

COSTA, D. G. **Java em rede-recursos avançados de programação**. Brasport, 2008.

ELKSTEIN, M. **Learn REST: a tutorial**. Em: <<http://rest.elkstein.org/>>. Acesso em: 28 dez. 2016.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 180p. Dissertação. University of California, Irvine. California, 2000.

ITU (2016). **ICT Facts Figures 2016**. Disponível em: <http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf>. Acesso em: 26 dez. 2016.

JSON. **Introducing JSON**. Disponível em: <http://www.json.org/>. Acesso em: 28 dez. 2016.

Maven, Apache. **Welcome to Apache Maven**. Disponível em: <https://maven.apache.org/>. Acesso em: 28 dez. 2016.

Oracle. et al. **Tecnologias Java**. Disponível em: <https://www.oracle.com/br/java/technologies/index.html>. Acesso em: 29 dez. 2016.

PostgreSQL. **About**. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 27 dez. 2016.

RFC2396. **Uniform Resource Identifiers (URI): Generic Syntax**. Disponível em: <https://tools.ietf.org/html/rfc2396>. Acesso em: 29 dez. 2016

SILVA, J. L. L. **Sistema de gestão pública para o cadastramento de ocorrências utilizado arquitetura de microserviços**. 34p. Artigo. Faculdade 7 de Setembro. Fortaleza, 2015.

Spring Boot, Pivotal Software inc. **Spring Boot Reference Guide**. Disponível em: <http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>. Acesso em: 27 dez. 2016.

Tomcat, Apache. **Apache Tomcat Welcome**. Disponível em:

<https://tomcat.apache.org/>. Acesso em: 28 dez. 2016.

W3C (2004). **Web Services Architecture**. Disponível em:
<https://www.w3.org/TR/ws-arch/> Acesso em: 27 dez. 2016.