**INTRODUÇÃO**

A rede mundial de computadores e os diversos aparelhos eletrônicos, denominados *Smart*`*s*, se conectam através da internet onde existe um número incalculável de informações e softwares. E é preciso fazer com que essas informações se cruzem. O mesmo ocorre com as empresas que adquirem softwares de vários fornecedores e precisam de alguma forma fazer todos eles se comunicarem. Segundo Emilio Dias, o *WebService* ajuda como uma ponte de ligação entre os mesmos (DIAS, 2016, p.11).

Para fins de exemplificação, temos empresas como Twitter, Google e Facebook que dependem muito de um modelo computacional acessível e integrado que permita que seus produtos interajam com aplicações de toda a Web. Eles precisam disponibilizar meios para que outros sistemas se integrem com seus serviços e isso acontece por meio dos tipos de *WebService*.

Com isso, podemos dizer que a escolha do uso de tecnologia de *WebService* não se deve apenas ao fator técnico da questão, mas também à necessidade de se desenvolver sistemas que viabilizem uma relação cada vez melhor aos usuários.

Em tempos em que clientes se tornam cada vez mais exigentes é fundamental que as empresas estejam aptas para atender tais necessidades, caso contrario, o risco é de ficarem pra trás na concorrência.

Baseado na premissa de que a demanda dos usuários se torna maior a cada dia, podemos afirmar que o WebService utilizado deve evoluir na mesma velocidade para atendê-la, suportar e integrar tudo isso.

A esse respeito Emilio Dias nos esclarece que, as arquiteturas monolíticas e camadas acopladas podem não ser totalmente eficazes para lograr êxito nisso e é fundamentando-se neste fato que este trabalho pretende mostrar como o mo delo REST pode ajudar a programar soluções para atender a todos esses requisitos (DIAS, 2016, p.12).

**1. WEBSERVICE**

De acordo com informações do site DEVMEDIA (2009), a atividade de desenvolvimento de sistemas é naturalmente complexa, pois tem o escopo de criar artefatos para o atendimento dos requistos de negócio e, ainda, envolve requisitos de aplicação como protocolos de comunicação e gerenciamento de transações distribuídas.

Antes de começarmos a falar sobre conceito e a **tecnologia de um *webservice***, vamos entender um pouco de sua história e evolução. O W3C (*World Wide Web Consortium*) aceitou a submissão do *Simple Object Access Protocol* (SOAP) no início dos anos 2000. Este formato de mensagem baseado em XML estabeleceu uma nova estrutura de transmissão para comunicação entre aplicações (ou entre serviços) via HTTP. Sendo uma tecnologia não amarrada a fornecedor, o SOAP disponibilizou uma alternativa atrativa em relação aos protocolos proprietários tradicionais, tais como CORBA (*Common Object Request Broker Architecture*) e DCOM (*Distributed Component Object Model*).

O informativo DEVMIDIA (2009) ainda nos acrescenta que no decorrer do ano seguinte, o W3C publicou a especificação WSDL. Uma nova implementação do XML, este padrão forneceu uma linguagem para descrever a **interface dos *webservices***. Posteriormente suplementada pela especificação UDDI (*Universal Description, Discovery and Integration*), que proporcionou um mecanismo padrão para a descoberta dinâmica (*dynamic discovering*) de descrições de serviço, a primeira geração da **plataforma de *Webservices*** foi estabelecida.

A partir de então, os *webservices* foram aderidos em larga escala pelos vendedores e fabricantes. A indústria seguiu dando suporte se rendendo à popularidade e importância desta plataforma e de princípios de projeto orientados a serviço.

Analisando a questão teórica podemos dizer que, um *Webservice* é configurado por um conjunto de métodos acedidos a serem invocados por outros programas que utilizando tecnologias da *web*.

Assim sendo, Um *WebService* é utilizado para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente das linguagens de programação utilizadas nessas plataformas.

  De acordo com o OPENSOFT (2007), os webservices são criados para funcionar com qualquer sistema operacional, plataforma de *hardware* ou linguagem de programação de suporte *web*. Sua finalidade consiste apenas em transmitir informação, ou seja, não são aplicações Web que suportam páginas que podem ser acedidas por utilizadores através de navegadores *web.*

 Eles permitem a reutilização de sistemas já existentes e acrescentam novas funcionalidades a eles, sem que seja necessário criar um sistema a partir do zero. Deste modo, é possível melhorar os sistemas já existentes, integrando mais informação e novas funcionalidades de forma simples e rápida.

É correto afirmar então que *WebService* é o termo utilizado para definir o recurso que soluciona a integração dos sistemas e a comunicação entre aplicações, mesmo sendo elas distintas, os novos programas podem se comunicar com aqueles que já existem. Tudo isso acontece através da comunicação do *WebService* com o banco de dados que torna possível o acesso e a alteração dos registros contidos neles. Para isso, é utilizada uma linguagem de marcação, como o XML (*Extensible Markup Language*) e o JSON (*JavaScript Object Notation*), que promovem a troca de informações.

**1.1. Como funciona um *webservice***

Considerando as operações disponíveis no *webservice*, a aplicação solicita uma dessas operações, o *webservice* efetua o processamento e envia os dados para a aplicação que requereu a operação. Por fim, a aplicação recebe os dados e faz a sua interpretação, convertendo-os para a sua linguagem própria (OPENSOFT, 2007, disponível em: < https://www.opensoft.pt/web-service/ > acesso em 22 de setembro de 2018).

Imagine que um cliente, utilizando um smartphone com um sistema operacional X, solicita uma informação de uma determinada empresa e que essa mesma informação seja solicitada por um cliente utilizando um aparelho com um sistema operacional Y, esta empresa não vai precisar de duas aplicações para promover a troca de informações, pois cabe ao *webservice* coletar os dados recebidos e traduzi-los para os dois clientes de acordo com o sistema de seus aparelhos.

**1.2. Tipos de Webservice**

Existem dois tipos de *webservice*, o SOAP (*Simple Object Access Protocol*), que é mais “maduro” no mercado e utiliza especificações de WSDL (*Web Services Description Language*) ou WADL (*Web ApplicationDescription Language*), que nada mais são que XML, para definir cada operação.

Já o REST (Representational State Transfer) se integra ao protocolo HTTP (*Hypertext Transfer Protocol*), seguindo seus métodos (GET, POST, PUT, DELETE, OPTIONS, HEAD) para facilitar nas operações. Por esta e outras características o REST vem sendo mais requerido que o SOAP.

**1.2.1. *Simple Object Access Protocol* (SOAP)**

O protocolo [SOAP](http://www.w3.org/TR/soap12-part1/) consiste na transferência de mensagens em formato XML utilizada em ambientes distribuídos. O padrão SOAP funciona como uma espécie de *framework*, que viabiliza a interação entre operações de diversas plataformas com mensagens personalizadas.

Aplicando este padrão em *webservice*, geralmente usa-se o WSDL (*Web Services Description Language*) para descrever a estrutura das mensagens SOAP e as ações possíveis em um *endpoint*. Uma das maiores vantagens é que várias linguagens e ferramentas conseguem ler e gerar mensagens facilmente. Várias linguagens de programação permitem a geração de objetos de domínio, *Stubs* e *Skeletons*, a partir da definição do WSDL, permitindo a comunicação remota via RPC (*Remote Procedure Call*) através de chamadas a métodos remotos, inclusive com argumentos complexos, como se fossem chamadas locais.

O grande problema desse padrão, é que ele adiciona um *overhead* considerável, tanto por ser em XML quanto por adicionar muitas *tags* de meta-informação. Além disso, a serialização e desserialização das mensagens podem consumir um tempo considerável, o tornando lento.

**1.2.2. *Representational State Transfer* (REST)**

O REST é outro protocolo de comunicação diferente do SOAP. Este, por sua vez, baseia-se no protocolo de hipermídia HTTP. Porém, ele não impõe restrições ao formato da mensagem, mas sim ao comportamento dos componentes envolvidos.

A maior vantagem neste protocolo é sua flexibilidade. Neste caso, existe a possibilidade de o desenvolvedor optar pelo formato mais adequado para as mensagens do sistema em conformidade com uma necessidade específica. Os formatos mais comuns são: JSON, XML e texto puro; mas, em teoria, qualquer formato pode vir a ser utilizado.

Temos assim, outra vantagem: quase sempre os *webservices* que usam REST são mais "leves" e, portanto, mais rápidos.

No entanto, existe uma desvantagem. O problema na utilização do REST pode se derivar justamente de suas vantagens. Como a definição do corpo de dados fica totalmente a cargo do desenvolvedor, os problemas de interoperabilidade podem ser mais comuns.

**2. ARQUITETURA CLIENTE-SERVIDOR**

O fundamento desta arquitetura é bem simples, ela é dividida em interface (Usuário, Cliente) e a parte onde são armazenados os dados (Servidor, Aplicação). A aplicação roda no servidor que passa para a API onde está contido seu WebService e então ele completa a ponte até os usuários finais, seja ele um *Desktop*, Android ou um site *web*.

Camila Camargo (2008) apresenta uma definição bem simples do que seria e como funciona a tecnologia cliente servidor. Segundo ela, diversos são os tipos de servidores, sendo que os mais conhecidos são: Servidor de Fax, de arquivos, *web*, e-mail, imagens e FTP. Cada um destes servidores tem uma funcionalidade específica, por exemplo, para que seja possível visualizar um site na internet, é preciso utilizar o Servidor *web*, o qual é responsável pelo armazenamento das páginas do mesmo. Para abrir estas páginas *web*, utilizamos um navegador, portanto o navegador é o cliente e o site é o servidor, uma vez que o primeiro acessa informações disponibilizadas pelo segundo. Desta forma, as redes que utilizam servidores são chamadas do tipo Cliente-Servidor.

**Vale dizer também que a tecnologia cliente/servidor é uma arquitetura na qual o processamento da informação é dividido em dois módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes). Assim, podemos dizer de maneira simplificada que, os processos cliente enviam pedidos para o processo servidor, e este por sua vez processa e envia os resultados dos pedidos. (WORDPRESS, 2017, disponível em: <https://arqserv.wordpress. com/2012/03/17/como-funciona-a-arquitetura-cliente-servidor/> acesso em 1º de outubro de 2018)**

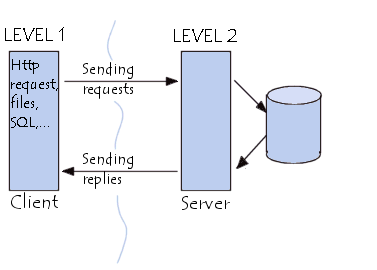
**Os sistemas mais pesados da rede, como, por exemplo, os bancos de dados, geralmente ficam alojados no servidor por este ter maior capacidade de armazenamento. Já as máquinas clientes, costumam ser menos potentes, uma vez que não precisam executar tantos aplicativos que requerem tantos recursos das máquinas.**

**O que mais importa entender em questões de funcionamento é que em uma máquina em arquitetura**Cliente/Servidor**não é necessário que todas as máquinas sejam do mesmo fabricante ou do mesmo tipo. O que se precisa realmente é que todas as máquinas possam se comunicar através da rede pelo mesmo tipo de protocolo de acesso.**

Abaixo veremos os tipos de modalidades de arquitetura cliente servidor:

**2.1. Arquitetura de dois níveis:**

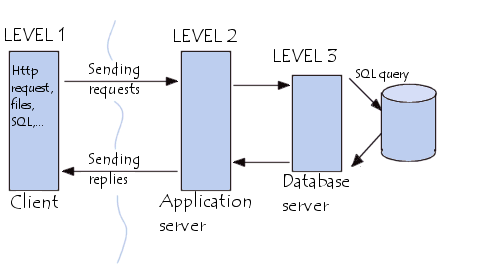
A arquitetura em dois níveis, que também pode ser chamada de arquitetura 2-*tier* caracteriza os sistemas cliente/servidor através dos quais o cliente pede um recurso e o servidor responde diretamente ao pedido, utilizando seus próprios recursos. O que significa que o servidor não requer outro aplicativo para proporcionar parte deste serviço:



Fonte: https://br.ccm.net/contents/149-a-tecnologia-da-arquitetura-cliente-servidor

**2.2. Arquitetura de três níveis:**

Já na arquitetura em três níveis, existe um nível intermediário, o que significa que ela está compartilhada entre um cliente, ou seja, o computador que solicita recursos, equipado com uma interface de usuário, geralmente um navegador, que é encarregado da apresentação; o servidor de aplicativo, também chamado de *software* intermediário, cuja tarefa é proporcionar os recursos solicitados e o servidor de dados, que fornece os dados necessários ao servidor de aplicativo:



Fonte: https://br.ccm.net/contents/149-a-tecnologia-da-arquitetura-cliente-servidor

O emprego maciço do termo arquitetura em três níveis designa as arquiteturas que envolvem servidores de empresa ou de bancos de dados.

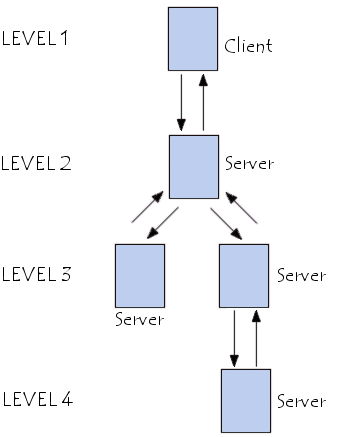
**2.3. Comparação entre os tipos de arquitetura:**

Na arquitetura em dois níveis o servidor pode fornecer todos os recursos solicitados pelo cliente diretamente, o que faz dele um servidor polivalente. Já nos casos da arquitetura em três níveis, os aplicativos do servidor são descentralizados, que é o mesmo que dizer que cada servidor é incumbido especialmente de uma tarefa. Imagine uma arquitetura em que temos um servidor para *web* e servidor para o banco de dados.

De acordo com o informativo CCM (2017), uma das vantagens da arquitetura em três níveis é que ela permite maior flexibilidade e segurança, pois ela pode ser definida de forma independente para cada serviço e em cada nível. Ela também apresenta melhor desempenho em função da divisão de tarefas entre os servidores.

**2.4. Arquitetura multiníveis:**

Nos casos de arquitetura em três níveis, cada um dos servidores, fica incumbido de efetuar uma tarefa específica. Sendo assim, um servidor pode oferecer o seu próprio serviço utilizando os serviços de um ou vários servidores ao mesmo tempo. O que torna possível que a arquitetura em três níveis possua vários níveis:



Fonte: https://br.ccm.net/contents/149-a-tecnologia-da-arquitetura-cliente-servidor)

**3. REST**

Antes de falarmos sobre o REST, vamos explicar uma questão que, em alguns casos, pode gerar certa confusão que é a diferença entre os termos REST e RESTful. Está é, inclusive, uma confusão muito comum, e consiste na forma como devem ser utilizados esses termos.

Segundo Emílio Dias (2016), o correto é a utilização do termo API RESTful. Também é importante entender o motivo dessa nomenclatura e qual o momento certo para usar cada uma. No momento sobre a discussão sobre o modelo e características, o correto é utilizar o termo REST, já nos casos em que estivermos tratando de uma implementação que usa essas mesmas características, deve-se usar RESTful. Entender essa diferença nos auxilia a sempre utilizar as nomenclaturas corretas. Isso também nos ajuda a deixar claro que REST nada mais é que um conjunto de boas práticas.

Para Rodrigo Calado (2013), o **REST** é um paradigma arquitetônico e o **RESTful** é um serviço web que usufrui deste paradigma. Pode ser conceituado também como um conjunto de restrições que define um padrão arquitetural com características específicas: entendo que benefícios e dificuldades aparecerão ao implementar um sistema seguindo o padrão **REST**.

Agora que definimos a diferença ente REST e RESTful podemos seguir explicando suas características.

De acordo com entendimento extraído das palavras de Emílio Dias (2016), embora pareça uma proposta recente, o REST teve seu surgimento no início dos anos 2000, a partir da tese de Ph.D de um cientista chamado Roy Fielding. Sua principal funcionalidade era a formalização de um conjunto de melhores práticas chamadas *constraints*. As *constraints* tinham como objetivo determinar a forma com que padrões como HTTP e URL deveriam ser modelados, aproveitando de fato todos os recursos oferecidos pelos mesmos.

Podemos dizer que, conforme aprendemos com o DEVMEDIA (2013), o REST não é um padrão ou uma arquitetura, e sim uma forma diferente e nova de se pensar sobre HTTP. É composto por um grupo de princípios e restrições que definem como padrões *web*, HTTP e URLs, podem e devem ser utilizados. Aqueles que defendem o REST mostram que ele apresenta uma forma mais simples de resolver problemas de integração entre aplicativos e simplifica a estrutura necessária para construir arquiteturas orientadas a serviços.

É considerado mais sofisticado por utilizar de forma vasta, cerca de 80%, o protocolo HTTP, aproveitando seus recursos o que o torna mais descomplicado. Utiliza armazenamento em cachê, para eliminar algumas interações entre cliente e o servidor permitindo um aumento de desempenho, podendo assim ter mais componentes e interações sem perda de velocidade e torna os dados mais confiáveis e seguros.

Alguns conceitos utilizados no REST:

1. **RECURSO:** São os itens que representam informações dentro da aplicação. Exemplo: Produtos de uma loja.
2. **IDENTIFICADORES:** Consiste em especificações dos recursos. Exemplo: Tendo como recurso uma camisa e a cor verde como um identificador, é uma camisa de cor específica.
3. **REPRESENTAÇÕES DE RECURSOS:** São as maneiras com as quais se podem obter determinadas recursos. Exemplo: Pode-se obter a camisa com ou sem embrulho para presente.
4. **METADADOS:** São descrições mais detalhadas sobre como se obter tal recurso ou representação. Exemplo: Enviar via SEDEX ou PAC.

Numa aplicação deste tipo, os recursos são os itens principais, onde sua representação vem do servidor para a aplicação cliente, mudando-a de estado. Seguindo o exemplo das roupas, o cliente acessa o site “loja.com/camisa/verde” então, ao comprar, o produto será enviado e ao receber na sua casa, em um embrulho para presente, o estado de sua entrega mudará de “a caminho” para “entregue”.

**4. PROTOCOLO HTTP**

O HTTP, *HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto), é um protocolo da camada de Aplicação do modelo OSI utilizado para transferência de dados na rede mundial de computadores, a *World Wide Web.* (OFICINADAET,2007 disponível em: <https://www.oficinadanet.com.br/artigo /459/o\_protocolo\_http> acesso em 25 de setembro de 2018).

Segundo Ludovico Fisher (2013), o HTTP é o protocolo que permite o envio e o recebimento de informações na *web*. Sendo que, um protocolo é um conjunto de regras que determinam quais os tipos de informações podem ser trocadas, e que mensagens são apropriadas para encaminhar a terceiros.

Ainda segundo FISHER (2013), no HTTP, existem dois papeis responsáveis pelo funcionamento correto do protocolo: o servidor e o cliente. Por padrão, o cliente sempre inicia a conversa; e então o servidor responde. Criando uma comunicação. O HTTP é baseado em texto; ou seja, mensagens são essencialmente uma fração de bits de dados, mas o corpo da mensagem pode conter outros arquivos de mídia.  A utilização de texto facilita a forma como monitoramos requisições HTTP.

Geralmente, este protocolo utiliza a porta 80 e é usado para a comunicação de sites, através da linguagem HTML (*Hipertext Markup Language*, ou Linguagem de Marcação de Hipertexto). No entanto, para que exista comunicação com o servidor do site é necessária à utilização de comandos adequados, que não estão em linguagem HTML.

Protocolo é um conjunto de regras passadas para informar quais tipos de dados irão ser trocados e que mensagens serão enviadas. O protocolo HTTP é baseado em duas importantes partes, o cliente, que por padrão solicita os dados, e o servidor que fornece os dados, como uma conversa onde o cliente pergunta e o servidor responde. Ele utiliza de mensagens em forma de bits para se comunicar, podendo conter nelas mídias variadas. Para compor essas mensagens são utilizadas duas *tags*, o *head* onde são informadas especificações de metadata e métodos HTTP, e o *body* onde estão os dados que deseja transmitir pela rede, este pode ser vazio, diferente do *head*.

De acordo com Nando Vieira (2007) podemos entender que, o protocolo HTTP baseia-se em requisições e respostas entre clientes e servidores. O cliente, navegador ou dispositivo que fará a requisição, também chamado de user agent, solicita um determinado recurso (*resource*), enviando um pacote de informações contendo alguns cabeçalhos (*headers*) a um URI (*Uniform Resource Identifier*, ou Identificador de Recursos Universal) ou, mais especificamente, URL (*Uniform Resource Locator*,ou Localizador de Recursos Universal). São identificadores de recursos, por exemplo, identifica uma página da web. Produtos/camisa identifica o produto camisa. Ao receber essas informações o servidor envia uma resposta, que pode ser um recurso ou um simplesmente outro cabeçalho.

**4.1. Funcionamento do protocolo HTTP:**

Um sistema de comunicação em rede possui diversos protocolos que trabalham em conjunto para o fornecimento de serviços. Para que o protocolo HTTP consiga transferir seus dados pela Web, é necessária atuação dos protocolos TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*) que tornam possível a conexão entre clientes e servidores através de *sockets* TCP/IP.

Segundo conceito extraído do entendimento de Kemuel Rocha (2015), o HTTP utiliza o modelo cliente-servidor, como a maioria dos protocolos de rede, baseando-se no paradigma de requisição e resposta. Um programa requisitante (cliente) estabelece uma conexão com um outro programa receptor (servidor) e envia-lhe uma requisição, contendo a URI, a versão do protocolo, uma mensagem MIME (padrão utilizado para codificar dados em formato de textos ASCII para serem transmitidos pela Internet) contendo os modificadores da requisição, informações sobre o cliente e, possivelmente, o conteúdo no corpo da mensagem.

De acordo com os ensinamentos de FOSCARINE (2001), o protocolo HTTP faz a comunicação entre o cliente e o servidor por meio de mensagens. O cliente envia uma mensagem de requerendo um recurso e o servidor, por sua vez, envia uma mensagem de resposta ao cliente atendendo a solicitação. Segundo, os dois tipos de mensagens existentes no protocolo utilizam um formato genérico, definido na RFC 822, para a transferência de entidades.

Em conformidade com o entendimento de FIELDING (1999), podemos concluir que, uma mensagem, seja ela de requisição ou de resposta, será composta, de acordo com a definição estabelecida na RFC 2616, se apresentando como uma linha inicial, não apresentando nenhuma outra linha de cabeçalhos, uma linha em branco obrigatória finalizando o cabeçalho e, finalmente, o corpo da mensagem, que é opcional em alguns casos. Nesta seção serão apresentados os campos que compõem uma mensagem mais detalhadamente; ou seja, o HTTP apresenta o sítio ou local onde está a página da Internet.

**5. MÉTODOS HTTP (VERBOS):**

Os métodos HTTP, também conhecidos como verbos, nada mais são que os comandos que informam ao servidor o que deve ser feito com o conteúdo informado na URL.

Fazendo alusão ao conceito estabelecido por FISHER (2013), podemos dizer, em outras palavras, que os verbos HTTP dizem ao servidor o que ele deve fazer com a informação identificada na URL. A requisição pode, por opção, conter informações adicionais no corpo/*body*, que podem ser necessárias para executar a operação.

Como já foi dito, a tecnologia REST utiliza dos métodos HTTP para promover a comunicação entre cliente-servidor, eles são:

1. **GET:** pegar informações já existentes de um recurso. Exemplo: mostrar os pedidos de um determinado cliente, ou mostrar todos os produtos de uma categoria. Contudo o GET não faz alterações nas informações apresentadas.
2. **POST:** adiciona informações utilizando algum recurso. Exemplo: adicionar um novo cliente, pedido ou produto ao banco de dados.
3. **PUT:** altera um recurso já existente. Exemplo: editar um produto, cliente, etc, se ele não existir também pode criar o recurso. A maior diferença entre POST e PUT é que o anterior pode carregar informações também além de usar recursos.
4. **DELETE:** apaga um recurso. Exemplo: excluir um produto categoria, etc.
5. **OPTIONS:** mostra aos clientes quais os tipos de requisições são permitidas a um dado recurso. Exemplo: se o POST é permitido, ou se o DELETE é permitido.

**6. FERRAMENTAS E TECNOLOGIAS**

**6.1 NetBeans**

O NetBeans é uma IDE (*Integrated Development Environment*), que em português significa Ambiente de Desenvolvimento Integrado, que possibilita no desenvolvimento de aplicações utilizando várias linguagens de programação, entre elas: Java, HTML5, C++ e PHP, que foi usada no desenvolvimento do *webservice* que será apresentado no próximo capítulo.

É gratuito e de código aberto, o tornando bem popular entre desenvolvedores, tendo como principal objetivo a reutilização de código para agilizar e simplificar o desenvolvimento, uma outra vantagem é que por se tratar de uma aplicação Java ele roda em qualquer SO (Sistema operacional) que possui instalado a JVM (*Java Virtual Machine*).

Segundo o próprio site, a IDE surgiu como um projeto de faculdade na república Tcheca em 1996, e era conhecido como “Xelfi”, tendo como objetivo se tornar uma IDE que utilizasse Java, porém parecida com o Delphi, outra IDE muito popular que utiliza as linguagens C++ e Object Pascal. Xelfi foi a primeira IDE de Java que foi escrita com a Linguagem Java.

O NetBeans propriamente dito foi lançado em dezembro de 2000 pela Sun Microsystems e em 2010 se tornou parte da Oracle, todo este período sendo uma IDE totalmente de código aberto.

**6.2 PHP**

De acordo com o site oficial, a sigla PHP faz um acrônimo recursivo de Hypertext Preprocesso, e o denomina como uma linguagem de programação de código aberto. O PHP é mais utilizado em desenvolvimento web, pois pode ser utilizado juntamente com o HTML, economizando vários comandos necessários em outras linguagens, como C e Perl, que o próprio site cita.

Um de seus destaques é o motivo de todo o código ser gerado através do servidor, fazendo com que o código fonte não fique a mostra para os clientes daquele HTML retornado, outro motivo de destaque é sua curva de aprendizagem, que é consideravelmente pequena para um iniciante, porém oferece uma gama de possibilidades para desenvolvedores profissionais.

**6.3 MySQL Workbench**

SQL (*Structured Query Language*, em português: Linguagem de Consulta Estruturada) é uma linguagem padrão para se comunicar com um banco de dados relacional. Segundo o site oficial, o MySQL é uma ferramenta unificada, que fornece:

* Modelagem de dados
* Criação de SQL
* Ferramenta de administração para configurar servidores
* Administração de Usuários
* Backup (Cópia de segurança) e muitos outros recursos

O programa foi utilizado no projeto para a criação do banco de dados “topgas” juntamente de suas tabelas, fazendo uma conexão direta com o PHP utilizando a IDE NetBeans.

**CONCLUSÃO:**

**REFERÊNCIAS BIBLIOGRAFICAS:**

CCM, 2017 disponível em: <https://br.ccm.net/contents/149-a-tecnologia-da-arquitetura-cliente-servidor> acesso em: 15 de setembro de 2018.

CAMARGO, Camila – *O que é Cliente-Servidor?* – TECMUNDO, 2008. Disponível em: <https://www.tecmundo.com.br/ 982-o-que-e-cliente-servidor-. htm/> acesso em 23 de setembro de 2018.

CALADO, Rodrigo – *O Que É REST? Um Resumo Do Assunto: Características, Conceitos, Vantagens E Desvantagens… Prefiro Dizer Que É Uma Rápida Introdução Ao Assunto…* .– 2013Disponível em: <http://www.rodrigocalado.com. br/o-que-e-rest-um-resumo-do-assunto-caracteristicas-conceitos-vantagens-e- desvantagens-prefiro-dizer- que-e-uma-rapida-introducao-ao-assunto/> acesso em 20 de setembro de 2018.

DEVMIDIA, 2009, disponível em: <https://www.devmedia.com.br/web-services/2873> acesso em 01 de outubro de 2018

DIAS, Emílio – *Desmistificando REST com JAVA* – 1º Edição, ALAWORKS, Uberlândia, 2016, disponível em: <file:///C:/Users/fdbbbbbbbbbbbbbbbbb /Downloads/algaworks-livreto-desmistificando-rest-com-java-v1.1%20(4).pdf> acesso em: 05 de setembro de 2018. – passou o caminho errado , tem que falar sobre o livro passou o caminho da pasta do computador

FISCHER, Ludovico – *Guia de Introdução aos conceitos HTTP e REST* – ENVATOTUTS 2013. Disponível em: <https://code.tutsplus.com/pt/ tutorials/a-beginners-guide- to-http-and-rest--net-16340/>.

OFICINADAET 2007 – *Protocolo HTTP* – disponível em: <https://www.oficinadanet.com.br/artigo /459/o\_protocolo\_http> acesso em 25 de setembro de 2018.

OPENSOFT, 2007, disponível em: < https://www.opensoft.pt/web-service/> acesso em 22 de setembro de 2018.

ROCHA, Kemuel – *Funcionamento do protocolo HTTP* – 2015, disponível em: < https://kemuelrocha.wordpress.com/2015/02/04 /funcionamento-do-protocolo-http/> acesso em : 19 de setembro de 2018.

STACKOVERFLOW – *Principais diferenças entre SOPA e REST* – R2014, disponível em: <https://pt.stackoverflow.com/questions /11183/quais-as-principais-diferen%C3%A7as-entre-soap-rest>, acesso em: 20 de setembro de 2018.

VIEIRA, Nando – *Um pouco mais sobre o protocolo HTTP* – 2007, disponível em: <https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http> acesso em 25 de setembro de 2018.

**WORDPRESS – Como funciona a arquitetura cliente servidor – 2017, disponível em: <https://arqserv.wordpress.com/2012/03/17/como-funciona-a-arquitetura-cliente-servidor/> acesso em 01 de outubro de 2018.**

**SUMÁRIO:**

|  |  |
| --- | --- |
| **1. INTRODUÇÃO:** .................................................................................................. | 00 |
| **2. WEBSERVICE:** ................................................................................................. | 00 |
| **3. ARQUITETURA CLIENTE-SERVIDOR:** ........................................................... | 00 |
| **4. REST** ................................................................................................................. | 00 |
| **5. PROTOCOLO HTTP:** ...................................................................................... | 00 |